

A PROTOTYPE SYSTEM FOR LLM-BASED AUTOMATIC GENERATION OF MULTIPLE-CHOICE QUESTIONS IN MACHINE LEARNING PROGRAMMING

Wakatani A^{1*} and Maeda T²

¹*Faculty of Intelligence and Informatics, Konan University, Japan*

²*Faculty of Science and Technology, Seikei University, Japan*

Abstract: In recent years, the importance of mathematical, data science, and AI education has grown significantly, making it necessary to learn machine learning programming. In programming education, while acquiring knowledge through lectures is important, the experience of actually creating programs helps solidify that knowledge. However, preparing numerous programming assignments places a significant burden on educators. This study proposes a system that utilizes large language models (LLMs) to generate programming tasks for scikit-learn, a machine learning library written in Python. It describes the implementation and evaluation of a prototype version of this system. The programming exercises used in that system implement a multiple-choice question format where part of the program is left blank, and the student must fill in the missing text. This serves as an alternative implementation for programming exercises. Our preliminary evaluation confirmed that by refining the prompt, tasks suitable for the purpose are automatically generated. However, to achieve accurate task generation, additional instructions to the prompt are required for each target problem category. Therefore, establishing a universal prompt generation method for this purpose is a future challenge

Keywords: python language, self-learning, AI, auto-generation

Introduction

The Mathematical · Data Science · AI Education Program Accreditation System, initiated in fiscal year 2021 under the leadership of Japan's Ministry of Education, Culture, Sports, Science and Technology, is a system designed to promote universities and other institutions' engagement in education related to mathematical · data science · AI. As this situation indicates, we have entered an era where all working adults must learn data science as a fundamental literacy for the digital society. In this learning process, programming education has become an indispensable element for the utilization and implementation of applications such as machine learning. Furthermore, in programming education, experiential learning through actual program creation is indispensable, not merely learning programming language syntax or how to use necessary libraries. However, creating a large volume of programming assignments and grading them is extremely cumbersome, placing a significant burden on educators. Therefore, to simulate a

*Corresponding Author's Email: wakatani@konan-u.ac.jp

programming experience, we create multiple-choice questions where part of a program with specified content is left blank. We provide options to fill in the blanks and ask the student to select the correct answer. Multiple-choice questions are a frequently used method in learning, and by having learners consider phrases to fill in parts of a program, they are expected to serve as a substitute for programming experience. However, as mentioned earlier, preparing a large number of multiple-choice questions in advance places a significant burden on educators.

Meanwhile, the technology of generative AI based on LLMs (Large Language Models) is advancing at a tremendous pace and is being applied to new applications. Interactive AI and generative AI such as OpenAI's ChatGPT and Google's Gemini are used by many people to create sentences and communicate naturally with a high degree of accuracy, but there are still issues such as incorrect information being provided and some LLMs learning information that is not always up to date. However, the technology to create the programs is highly developed. For example, Code Interpreter, based on OpenAI's GPT-4, can not only automatically generate Python programs upon receiving instructions written in natural language, but also execute them using appropriate data sets. In other words, the functions of the generative AI, such as implementing software using a programming language and understanding the contents of the program, have achieved a very high level of perfection.

Therefore, this study attempts to automatically generate multiple-choice questions by leveraging the capabilities of Large Language Models. To achieve this objective, we prototype a system designed to support this process in a structured and systematic manner. Specifically, the system follows these steps: 1) provide a theme sentence that clearly specifies the outline and scope of the target program, 2) generate a multiple-choice question in JSON format by using an appropriate prompt in conjunction with the given theme sentence, and 3) generate an appropriate learning webpage using Streamlit based on the generated JSON data, thereby enabling users to interact with and study the content effectively.

Related research has seen numerous attempts to automatically generate multiple-choice questions using large language models (LLMs), with a variety of approaches proposed to address the inherent difficulty of generating plausible yet incorrect distractor options. In particular, Lee et al. propose a system in which humans collaborate with LLMs to generate options for mathematical calculation problems, emphasizing that the fully automatic generation of such problems remains inherently challenging when relying solely on an LLM. They report on the development of a prototype system and demonstrate improvements in the accuracy and quality of the generated options (Lee et al., 2024). Furthermore, Bitew et al. showed that, in the process of generating options, presenting multiple example options for different problems—an approach known as few-shot prompting—can improve the usefulness rate by increasing the likelihood of producing

plausible yet incorrect distractors that are beneficial for learning (Bitew et al., 2023). Building upon these prior studies, this study investigates whether refining the wording and structure of prompts can further enhance the accuracy of automatically generated multiple-choice questions, particularly for program fill-in-the-blank problems in programming education. Specifically, it examines whether such prompt refinements can increase the proportion of generated questions that are not only valid and viable as options but also pedagogically useful for learners.

Materials and Methods

To generate Japanese multiple-choice questions with code generation, several candidate LLMs with publicly available APIs exist, such as Anthropic Claude, Google Gemini, and OpenAI. However, this study will use OpenAI, which is estimated to have a large user base. Additionally, the target for problem generation is to create multiple-choice programming exercises for learning how to use commonly employed functions and options in scikit-learn, a machine learning library in Python, specifically for programming regression analysis (linear regression, Ridge regression, Lasso regression)

OpenAI API and prompt

This system prepares a base text (theme sentence) and generates multiple-choice questions based on it. This prototype is limited to multiple-choice questions concerning linear regression, Ridge regression, and Lasso regression in scikit-learn. Note that for questions covering broader fields beyond regression, a wider range of base theme sentences would need to be prepared accordingly. However, implementation using the same methodology is possible, and expansion is straightforward. The theme sentences used in this prototype are shown in Table 1.

Table 1: Theme statements for prompt

1	When using Lasso regression in scikit-learn, the Lasso function is employed. The alpha option controls regularization. The fit method and predict method are used. The model is represented by the intercept attribute and the coef attribute.
2	When using Ridge regression in scikit-learn, the Ridge function is employed. The alpha option controls the regularization. The fit method and predict method are used. The model is represented by the intercept attribute and the coef attribute.
3	When using linear regression in scikit-learn, the LinearRegression function is employed. The fit method and predict method are utilized. The model is represented by the intercept attribute and the coef attribute.

In problem generation, one theme sentence is randomly selected (denoted as A), a language is chosen (either English or Japanese, denoted as B), and a generation guideline sentence is provided (denoted as C). These are combined into a prompt as follows. The generation guideline sentence will be described later. The prompt is shown below. “Regarding text (A), consider a fill-in-the-blank problem where one part of the Python code is left blank. Furthermore, the problem will be a multiple-choice question with four options. Show the Python code for the problem, the problem statement, and the wording of the four options along with their answer numbers.

The baseline prompt is “The wording of the options does not need to include the option numbers. Also, line breaks are included in the Python code. Also, the data is initialized in the Python code. Comply with (B). Ensure the wording of the incorrect choices appears incorrect. Use (C) as the language.” It is noted that our system provides the prompt in Japanese. Among these, generation guideline B provides hints for problem generation, thereby enhancing the diversity of problem generation. The generation guidelines used are shown in Table 2.

Table 2: Problem generation statements for prompt

1	Ask for the function name.
2	Ask about the attributes.
3	Ask about the method.
4	Ask about the option settings.
5	Ask about how to use the options.
6	Make it a program combined with Numpy.
7	Make it a program combined with pandas.

For the response output, it should be obtained in JSON format to facilitate generating web pages for multiple-choice questions. Therefore, we utilize the structured output feature in the OpenAI API. Structured output is a feature available since August 2024 for models like GPT-4o, which guarantees that the model output will perfectly match the JSON schema specified by the developer. This allows us to obtain the output results of the generated multiple-choice questions, including the question stem and the correct answer (key)

along with three incorrect answers (distractors) in JSON format, making it easy to generate web application source code. In this prototype, the response_format format was used. In this format, by passing the response_format argument to the client.chat.completions.create function and describing the JSON schema within its value, the output becomes JSON. The main description is shown in Figure 1.

```
1 "schema": {  
2   "type": "object",  
3   "properties": {  
4     "stem": {"type": "string"},  
5     "Python code": {"type": "string"},  
6     "choice 1": {"type": "string"},  
7     "choice 2": {"type": "string"},  
8     "choice 3": {"type": "string"},  
9     "choice 4": {"type": "string"},  
10    "answer": {"type": "number"},  
11  },  
12  "required": ["stem", "Python code", "choice 1",  
13    "choice 2", "choice 3", "choice 4", "answer"],  
14  "additionalProperties": False,  
15 },
```

Figure 1: JSON schema in client.chat.completions.create (selected)

From the obtained response, quiz response = json.loads(response1.choices[0].message.content) stores the information in the dictionary variable quiz response, which is then used to generate the appropriate web source code.

Implementation on Streamlit Cloud

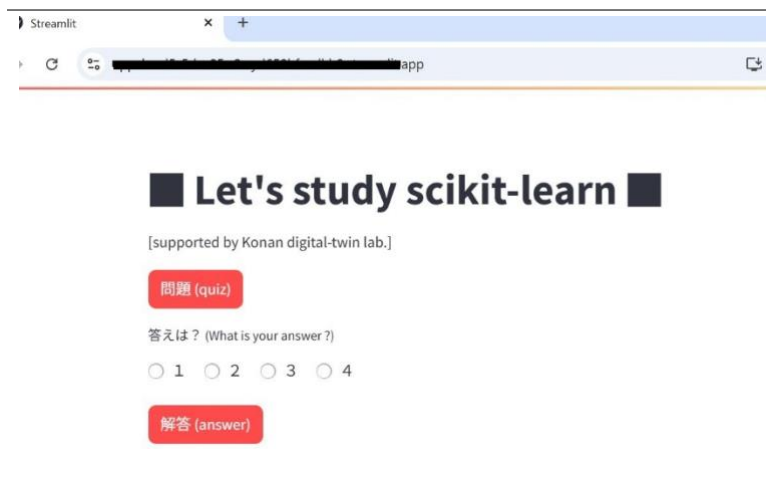


Figure 2: Four-choice questions for scikit-learn programming

For this prototype, the web source code utilizes Streamlit, and the web system is built on Streamlit Cloud. Streamlit is a framework for creating web applications within a Python environment. While its usage is simpler compared to other frameworks (such as Flask), its low learning cost makes it suitable for creating practical web applications in a straightforward and accessible manner. Additionally, Streamlit Cloud is a hosting service that allows you to easily publish and share web applications created with Streamlit on the cloud. It automatically deploys by simply linking to your GitHub repository and can be used free of charge under certain conditions (As of April 12, 2026). It has a low learning cost, making it well-suited for visualizing research results and quickly sharing prototypes. It is widely used for presenting analysis results and for educational and experimental purposes.

Figure 2 shows the initial screen of this system using Streamlit. Clicking the quiz button on this screen displays the problem screen, where learners fill in the blanks within the program with the appropriate answers. Figure 3 shows the screen displaying the problem statement. In this problem, the blank space is indicated by `__`, and the task is to provide the method used for training in linear regression. The options list plausible names that appear to be common English words, but for learners who have studied scikit-learn, `fit` is the obvious answer, thus promoting the consolidation of knowledge gained through learning.

以下のPythonコードを完成させるために、空欄に入る最も適切な選択肢を選びなさい。

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# サンプルデータの作成
data = {'面積': [30, 50, 70, 90, 110], '価格': [300, 500, 700, 900, 1100]}
df = pd.DataFrame(data)

# 線形回帰モデルの作成
X = df[['面積']]
y = df['価格']
model = LinearRegression()
model.___(X, y)

# モデルの係数と切片の表示
print('係数:', model.coef_)
print('切片:', model.intercept_)
```

次の選択肢から正しいものを選べ (Choose the correct one)

- 1 : compute
 - 2 : train
 - 3 : adapt
 - 4 : fit
-

Figure 3: Example of four-choice questions

Results and Discussion

In this preliminary experiment, we applied gpt-5, gpt-5-mini, gpt-4o, and gpt-4o-mini as large language models (LLMs), generating 20 multiple-choice questions each. We compared the response time required for generation and subjectively evaluated the quality of the generated multiple-choice questions. The usage fees for gpt-5, gpt-5-mini, gpt-4o, and gpt-4o-mini are \$1.25, \$0.25, \$2.50, and \$0.15 per 1 million input tokens, respectively. Therefore, considering running costs, selecting gpt-4o-mini or gpt-5-mini is preferable. Figure 4 shows the response times from OpenAI for each LLM. As shown in the figure, gpt-5 takes 5.7 times longer than gpt-4o and others. However, since gpt-5 was released very recently, it is unclear whether this is an effect of that or an inherent characteristic. Additionally, gpt-5-mini has approximately half the response time of gpt-5.

On the other hand, the response times for gpt-4o and gpt-4o-mini are nearly identical. In the initial implementation, for all APIs, over 90% of the problem generation results designated option 1 as the correct answer out of the four choices. Therefore, after receiving the response, it was necessary to perform operations such as randomly swapping the correct answer with the others within the Python program. Note that response times marked with “(rev)” indicate the response times after implementing the prompt improvements described later.

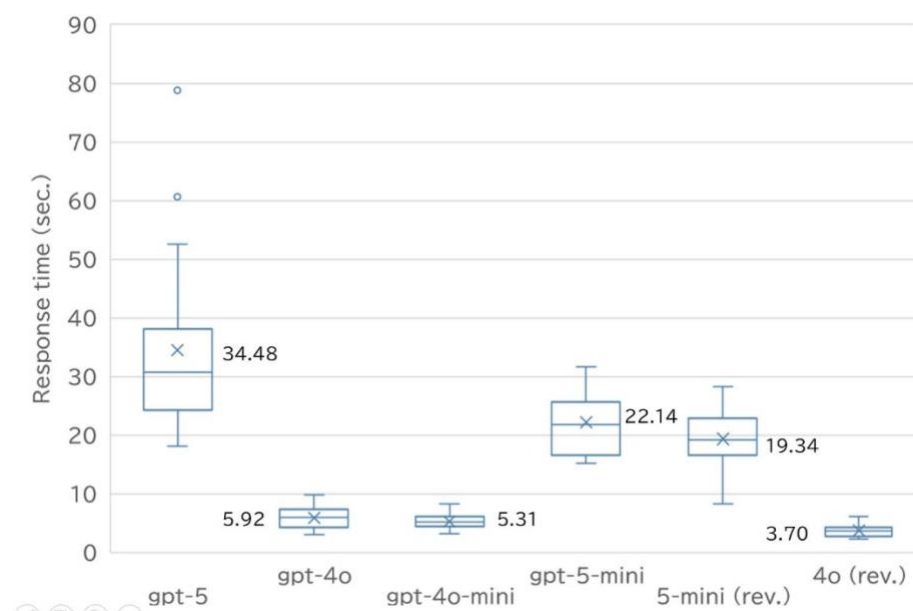


Figure 4: Response time. (sec.)

Table 3 shows a comparison of quality for automatically generated tasks. As shown in the table, gpt-4o-mini demonstrates insufficient accuracy in nearly all problem generations. For example, since these are fill-in-the-blank programming problems, only fragments of code conforming to the programming language's syntax should be provided as options.

Table 3: *Quality of the generated questions*

LLM	baseline	revised
gpt-5	100%	-
gpt-5-mini	60%	100%
gpt-4o	85%	90%
gpt-4o-mini	0%	-

However, in some cases, a brief explanation is appended alongside the code fragment (e.g. “2: ridge.coef indicates the model’s coefficient.” “1: Use the predict method to forecast the test data.”). While these additional explanations may contain useful information for learning, they serve as redundant text in a multiple-choice question. Furthermore, portions of the program were mixed within the question text, resulting in some elements failing to display correctly. Although the question text and program are separated as JSON elements, the response returned lacks sufficient structure. Additionally, in questions about the regularization parameter for Lasso regression, some asked for the specific value of alpha itself—that is, which value to choose from 0.1, 1.0, 10, or 100—and some could not be answered correctly. It is likely that the learned examples included instances with alpha=1.0, which it recognizes as correct answers, and considers any different value as incorrect. Note that other APIs also had problems asking for alpha values, but in those cases, the conditions were clearly specified, such as “The current alpha value is 5, but how can we strengthen regularization?” making them valid questions.

GPT-4o generated inadequate problems in approximately 15% of cases. For example, a problem like “print(Coefficients:', model.)” alone, which asks for “coef”, goes beyond the scope of a simple fill-in-the-blank question. Additionally, some displayed blanks like “model. a (X, y),” where the symbol “a” was provided in the blank, potentially confusing solvers. Other problems lacked blank displays altogether and contained deficiencies. However, compared to GPT-4o, the proportion of inadequate problems was

significantly lower. GPT-5 has no deficiencies or inadequacies, and its automatic question generation accuracy is extremely high.

However, its response time is long, making it less practical. While prompt refinement is desired to improve accuracy over GPT-4o, no effective improvement measures have been found at this time. How to evaluate the trade-off between accuracy and usability remains a future challenge.

In gpt-5-mini, 40% of the generated questions were inadequate, with most cases involving supplementary explanations added beyond the program text for filling in the blanks. For example, in a question asking to select “predict,” it stated: “predict — A method that returns a predicted value for the input from a trained model.” Additionally, for questions like “model. (X, y),” there were cases where the option was “model.fit(X, y).” Furthermore, overall, both the problem statements and the programs generated were shorter than those produced by GPT-5.

For GPT-5-mini and GPT-4o, which generate insufficient problems compared to GPT-5, we added text to the prompt. Namely, we appended: “Options should only include the text to be inserted into the blank spaces. Do not include any other text. Explicitly indicate the blank spaces within the Python program.” The results are shown in the “revised” column of Table 3. By adding additional wording to the prompt, the rate of insufficient problem generation for gpt-5-mini dropped to 0%. For gpt-4o, while issues remained—such as marking 0 as the correct answer when options ranged from 1 to 4, or asking for an alpha value without specifying conditions—overall improvement was observed. Therefore, it was confirmed that adding prompts to eliminate patterns exhibiting deficiencies is somewhat effective. Consequently, considering running costs, problem generation time, and ensuring a certain level of quality, this experiment determined that using gpt-5-mini for problem generation, with appropriate prompt additions, is the optimal approach.

However, the approach used this time — namely, first automatically generating tasks using a baseline prompt, then improving quality by adding prompt text to address the resulting errors — is costly. In other words, for various problem areas, the task of identifying additional prompt text becomes cumbersome, making the establishment of universal prompt text for high-quality automatic task generation a future challenge.

Conclusion

In recent years, machine learning programming has gained increasing importance within mathematical, data science, and AI education. Focusing on scikit-learn within Python-based machine learning programming, we propose a system that utilizes large language models (LLMs) to generate programming assignments.

This paper describes the implementation and evaluation of a prototype version of this system. For the programming assignment, rather than having students write actual code, we implemented it by automatically generating fill-in-the-blank multiple-choice questions using an LLM. These questions target specific sections of a program where parts are left blank. We then implemented this as a web application using Streamlit and Streamlit Cloud. Preliminary evaluations confirmed that by refining the prompts, problems suitable for the purpose were automatically generated. However, since prompt refinement is required for each problem category, establishing a general prompt generation method for this purpose is necessary in the future. Furthermore, evaluating the usefulness of this system in actual educational settings remains a future research challenge.

Acknowledgements

Part of this research was supported by JSPS KAKENHI Grant Number 23K02671. This research was also supported in part by Konan digital twin laboratory and MEXT, Japan.

References

- Bitew, S. K., Deleu, J., Develder, C., & Demeester, T. (2023). Distractor generation for multiple-choice questions with predictive prompting and large language models. *arXiv:2307.16338* Retrieved from <https://arxiv.org/abs/2307.16338>.
- Lee, J., Smith, D., Woodhead, S., & La, A. (2024). Math Multiple Choice Question Generation via Human-Large Language Model Collaboration. *arXiv:2405.00864*. Retrieved from <https://arxiv.org/abs/2405.00864>.