New Perspectives in Science Education
International Conference
NEW PERSPECTIVES in SCIENCE EDUCATION

Improvement of Advisory System Using LLM for Run-Time Errors in C Programming Learning

Akiyoshi Wakatani[1], Toshiyuki Maeda[2]

Konan University, Japan[1]
Hannan University, Japan[2]

Abstract

Generative AI technology based on LLM (Large-scale Language Models) has been applied to various fields, and one of them is an attempt to apply the technology of automatic program generation to programming education. By using OpenAI's API, we developed and evaluated a prototype VTA (Virtual Teaching Assistant) system that takes C language programs with errors and error messages as input to LLM and outputs appropriate advice for beginner-level learners. The results showed that the VTA produced appropriate advice for syntactic and semantic programming errors, but for logical errors, the VTA gave only general explanations in some cases. We focused on floating-point errors among logic errors and found that the main causes were "denominator of division is zero," "divisor of modulus calculation is zero," and "result of integer multiplication overflows". Therefore, for each candidate error factor, appropriate auxiliary information, including line numbers, could be extracted from the program and added to the prompts to generate more appropriate advice and then we confirmed that the effectiveness of the VTA system was improved.

Keywords: *beginners, floating point exception, logical error, python language, self-study*

1. Introduction

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 [1] and Google's Gemini [2] 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.

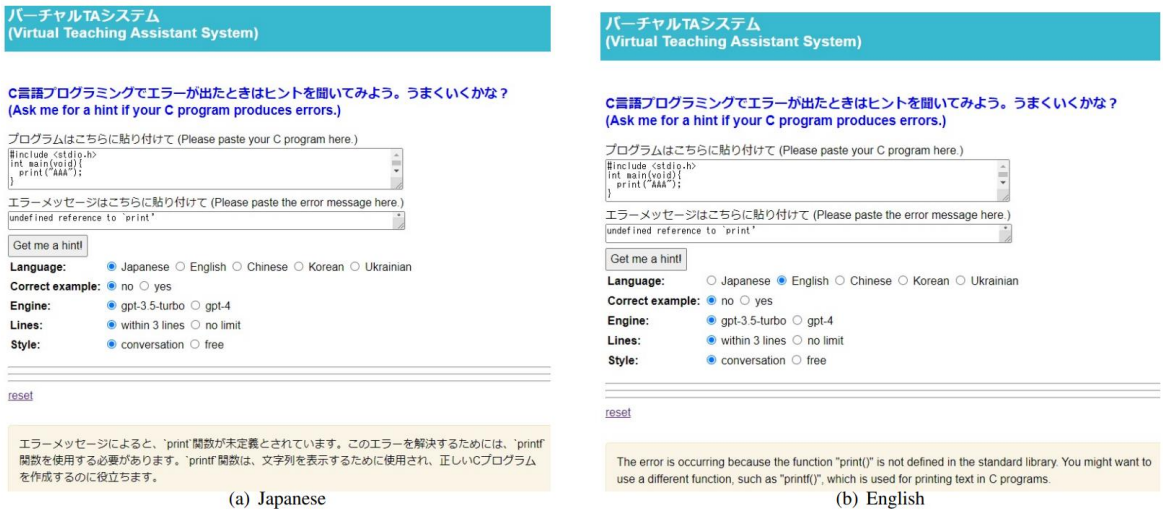
On the other hand, the training of human resources in IT and AI is an urgent issue. One of the requirements for IT and AI engineers is the ability to write programs, which is costly to acquire. In learning programming, it is necessary for students to know how to correct a program they have written by themselves if it is wrong, rather than to have a program automatically created by a programmer such as Code Interpreter. In such a case, advice from a teaching assistant (TA) is useful for beginners in learning programming, and it is not always necessary to show them the correct program. For example, in reference [3], it is stated that simply showing the correct answer is problematic for learners' understanding, and that it is important to help learners to reach the correct answer, rather than to show the correct answer using LLMs. Thus, LLMs are considered to have extensive experience with codes, but little understanding of good pedagogy.

The use of AI to improve the efficiency of programming learning has already begun. In reference [4], Kazemitabaar et al. examined the effectiveness of the OpenAI Codex in helping novice programmers learn programming, and reported a 1.15-fold improvement in the completion rate of programs, and a 1.8-fold improvement in their grades of paper tests. In the other literature [5], Leinonen et al. showed that GPT-3's explanations of error messages generated in Python programming helped learners to understand the error content, and that adjusting the temperature parameter appropriately made it more useful. However, since the Python language that this study targets is an interpreter language that interprets and executes the program, the error messages can be generated using both types of information, and the explanations output by the LLM can be considered to be generated in a prepared and appropriate manner. Therefore, since C is a compiler language, it is not clear from the results of

this study alone whether the LLM can generate useful advice for compiler languages that have both compile-time and run-time errors.
In a previous study by the authors [6], they divided errors in C into three types: grammatical errors, semantic errors, and logical errors, and showed that LLM generates appropriate advice for grammatical and semantic errors, but the advice generated is insufficient for logical errors. Therefore, this paper aims to improve the accuracy of advice generated by the LLM for logical errors, which are also runtime errors, by adding auxiliary information extracted from the program to the prompts. Although there are a variety of runtime errors, we focus here on floating-point errors, and after showing what causes such errors, we clarify the information needed to extract auxiliary information.

2. Programming Errors and Virtual TA System

In the execution of a C language program, there are three factors that cause programming errors [7]: the first is a syntax error, which is caused by a grammatical mistake, such as a misspelling of a keyword. The second is a semantic error, which occurs when the syntax is correct but the meaning of the program is not, for example, declaring a variable with the same name multiple times with different types, etc. The third is a logical error, which is often discovered at runtime and is also called a runtime error, for example, a floating-point error when the divisor of a division becomes zero as a result of the calculation.
As already mentioned above, appropriate advice given by a TA is useful for beginners to correct mistakes in their programming programs by themselves. Therefore, we have developed a prototype of a virtual teaching assistant system (hereinafter referred to as the VTA system) that generates an OpenAI API based on the TA's advice. Figure 1 shows a screen capture of the system. The left side is the output screen in Japanese, and the right side is the output screen in English. Note that the figure shows our previous system, and the gpt-4o option is also available for the current system. This VTA system is compatible with the C language, and appropriate advice is displayed when the target program and the error messages generated for the program are entered



(a) Japanese (b) English

Fig. 1. Virtual TA system

.
In the authors' previous study [6], among the three types of programming errors, the advice generated for syntactic and semantic errors was generally appropriate and sufficient for learners to eliminate the errors by themselves. However, since C language is a compiler language, information generated at compile time, such as the correspondence between the line number in the program and the instruction number of a machine instruction, or between a variable declared in the program and its address in memory, is not stored at run time. Thus, logical errors are runtime errors, and in some cases for logical errors, the advice was only a general explanation.

3. Floating Point Error

To find out what causes floating point exceptions among runtime errors and what questions can be asked about them, we searched the Internet for “floating point exception (core dumped) C” and investigated the parts that cause them. The principal factors can be categorized as follows:

1. denominator of (integer) division is zero,
2. the divisor of a modulus calculation (i.e., the right-hand side of the % operator is zero), and
3. integer multiplication (e.g., “result *= k;”) results in overflow

The first two factors are easy to detect, but the last one is expected to be difficult to find. For example, “result *= k;” can be found by searching for the string “*=”, but some parsing may be required to find “result = result*k;”.

As shown in the previous literature [6], advice generation for logical errors with prompting only by the original program and error messages is only of a certain level of advice and is especially inadequate for floating-point exceptions. In particular, pointing out line numbers is rarely correct.

Therefore, when an error occurs due to a floating-point exception, it would be better to generate advice by mixing the line numbers corresponding to the factors mentioned above and the program part in question in the prompt.

For example, consider the following program example.

```
:
   36     printf ("3");
   37     while (number != 0)
   38     {
   39
   40         remain = number % mod;
   41         remain1 = constant - remain * mod;
   42
   43         switch (remain)
   44         {
          :
```

For this example, since “remain = number % mod;” is considered a candidate for the cause of the error, the phrase “The divisor of the modulus operation on line 40 may be zero” should be added. Furthermore, consider the following example.

```
          :
   20  int fac(int n) {
   21         int result = 1;
   22         int k;
   23
   24         for (k = 1; k <= n; k++) {
   25               result *= k;
   26         }
   27         return result;
   28  }
      :
```

For this example, “result *= k;” is considered a candidate for the error factor, so the statement “The result of the multiplication on line 25 may have overflowed” should be added for more appropriate advice.

4. Prompt Generation

For each cause of floating point error, a string search is performed for the corresponding character string, and the following words are added to the basic prompt to generate appropriate advice using OpenAI's API. Note that the choice of the sequence of the phrases is important, since the advice generated will vary greatly depending on the sequence of the phrases. For example, “Factor YY on line XX may cause a floating-point exception” and “Factor YY that may cause a floating-point exception is on line XX” may generate different advice. Mostly the former is better in accordance with our experiments.

4.1 The Denominator of Division Is Zero.

Lines containing the string “/” are candidates for the cause of the error, but it is necessary to distinguish them from lines containing “/*”, “*/”, or “//”, which represent comments. If the line number at which the candidate factor is found is n, the statement “The divisor of the division operation on line {n} may be zero, so there may be a floating-point exception, please look closely at line {n}” is added to the prompt.

4.2 The Divisor in the Modulus Calculation Is Zero.

A line containing the string "%" is a candidate for a cause of error. If the number of that line is n, the statement "The divisor of the modulus operation on line {n} may be zero, so there may be a floating-point exception, please look at line {n} carefully" is added to the prompt.

4.3 Integer Multiplication Results in Overflow.

Lines containing the strings "*=" and "XXX=XXX*" (where XXX is an arbitrary string) are candidates for error causes. If the line number is n, the statement "The result of the multiplication on line {n} may have overflowed, so there may be a floating-point exception, please look at line {n} carefully" is added to the prompt.

5. Evaluation

Error pattern	No. of lines	Factor of error	No. of error sources
1	17	Zero division	2
2	84	Zero modulus operation	2
3	83	Overflow of multiplication	2
4	38	Overflow of multiplication	4
5	26	Zero division	2
6	11	Zero modulus operation	1
7	55	Zero division	1

Table 1. Error patterns

After implementing the prompt generation as described above, we evaluate whether appropriate advice is generated for the seven programming examples that contain errors. Table 1 shows a summary of the seven programming error patterns included in this section, which are the targets of advice generation. These programs were obtained from the results of a search on the Internet using the keywords "floating point error." Note that No. of lines indicates the number of lines that make up the target program, Factor of error indicates the factor of error, and No. of error sources indicates the number of candidate error sources.
At this point, we manually generated auxiliary information in accordance with the previously described policy and generated advice in gpt-4o with temperature=0.5, using the auxiliary information and the program and error messages (in this case, "floating-point exception (core dump)") as prompts. The following is a list of the evaluations of the generated advice. One advice pattern was executed with no auxiliary information, and three advice patterns were executed with prompts with auxiliary information. "S" indicates that sufficient advice including line numbers was generated, "A" indicates that correct advice was generated although line numbers were not provided, "B" indicates that only general advice was generated, and "X" indicates that off-the-wall advice was generated. "No extra" indicates the evaluation of the advice generated without auxiliary information, and 'Turn 1, 2, 3' indicates the evaluation of the advice generated three times with prompts with auxiliary information.

Error pattern	No extra	Turn 1	Turn 2	Turn 3
1	B	S	A	S
2	B	S	S	S
3	B	S	S	S
4	A	S	S	S
5	B	S	S	S
6	X	S	S	S
7	X	A	A	S

Table 2. Results

For error pattern 1, there are two possible error locations due to zero division, and the auxiliary information gives the locations of the two divisions. Therefore, in the case without auxiliary information, the advice is just a general statement, but in the case with auxiliary information, although

there are references to locations that do not need to be pointed out, the advice is appropriate enough for the user to understand the program locations to be searched for. Similar results were obtained for Error Patterns 2, 3, and 5. Although the causes of the errors are different, the auxiliary information is utilized and effective advice is successfully generated.
Error pattern 4 is caused by an overflow of integer multiplication calculation in the part where “resultW *= i;” is written. The auxiliary information points out four potential error locations, including the above location, and provides valid advice based on them. In particular, one of the three pieces of advice generated not only points out the program location, but also points out that "if the variables resultW, resultH, and resultWH have large values, they may exceed the integer range" This is a good example of how the auxiliary information can be used to point out the program position. Thus, the target variable names can be identified from the information on the “number of lines in the program” and the “multiplication calculation” pointed out in the auxiliary information. Therefore, it seems that the auxiliary information improved the LLM's reasoning ability.
In the case of error patterns 6 and 7, only inappropriate advice could be generated in the absence of auxiliary information, indicating that the use of auxiliary information is essential for advice generation for floating-point errors.
In general, the advice including the program location is successfully generated by using the auxiliary information extracted from the program, and the information is sufficient for the user's program debugging. However, if the number of error occurrence candidates increases, the advice will include many points of irrelevant locations, which may cause confusion to the user. However, for beginners, the program length is short, and programs of less than 100 lines have at most four potential error locations.

6. Conclusion

The development of applied systems using LLM is evolving. We have developed a prototype system that generates appropriate advice for error messages that beginners need to deal with when programming the C language, and we have made suggestions on how to deal with logical errors, which are runtime errors. In the error pattern of floating-point errors used in this study, we were able to confirm the improvement of the effectiveness of the system by adding auxiliary information extracted from the program to the prompts. Note that this experiment was conducted using OpenAI's API and evaluated with gpt-3.5-turbo at temperature=0.5, but now gpt-4o is available, so the level of advice seems to have increased.
In the future, it will be necessary to evaluate the effectiveness of advice generation with other settings and other LLMs. In addition to floating-point errors, it is also an urgent issue to investigate the accuracy of advice generation for other runtime errors, such as segmentation errors and address errors, by using the same approach.

Acknowledgment

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

REFERENCES

[1] OpenAI. “ChatGPT”, Available: https://chat.openai.com/, 2024.
[2] Gemini, “Gemini”, Available: https://gemini.google.com/app, 2024.
[3] Hellas, A., Leinonen, J., Sarsa, S., Koutcheme, C., Kujanpaa, L., & Sorva, J. “Exploring the Responses of Large Language Models to Beginner Programmers’ Help Requests.”, https://arxiv.org/pdf/2306.05715.pdf, 13 pages, 2023.
[4] Kazemitabaar, M., Chow, J., Ma, C. K. T., Ericson,B. J., Weintrop, D., & Grossman, T. “Studying the Effect of AI Code Generators on Supporting Novice Learners in Introductory Programming”, The 2023 CHI Conference on Human Factors in Computing Systems. DOI:10.1145/3544548.3580919, 23 pages, 2023.
[5] Leinonen, J., Hellas, A., Sarsa, S., Reeves, B., Denny, P., Prather, J., & Becker, B. A. “Using Large Language Models to Enhance Programming Error Messages”, The 54th ACM Technical Symposium on Computer Science Education. DOI: 10.1145/3545945.3569770, pp. 563-569, 2023.
[6] Wakatani, A. and Maeda, T. “Prototype advisory system for learning C programming using generative AI”, Proceedings of the 15th International Conference on Education & Educational Psychology, 2024.

[7] Aho, A. V. M., Lam, Sethi, S. R., & Ullman, J. D. “Compilers: Principles, Techniques, and Tools (2nd ed.)”, Pearson Education, 2006.