Kashihara, A. et al. (Eds.) (2024). Proceedings of the 32nd International Conference on Computers in Education. Asia-Pacific Society for Computers in Education

Developing a LLMs-Driven System Based on Human-AI Progressive Code Generation Framework to Assist Mathematics Learning

Chun Yan Enoch SIT[a*], Yin YANG[b*], Wing Kei YEUNG[a*] & Siu Cheung KONG[ab*]
[a]*Artificial Intelligence and Digital Competency Education Centre, The Education University of Hong Kong, Hong Kong SAR, China*
[b]*Department of Mathematics and Information Technology, The Education University of Hong Kong, Hong Kong SAR, China*
*{ecysit, yyin, yeungwk, sckong}@eduhk.hk

Abstract: This paper proposed a system interface based on a novel progressive code generation framework to produce verified programming codes using natural language for mathematics learning. With the latest advancement of large language models such as GPT-4, people can ask GPTs to generate programming code using natural language. However, the codes generated may not be directly executable or aligned with the user's desired goal or purpose. Furthermore, different people may express their goals and purpose in different ways. Hence, there are many possible ways to map natural languages and code. Many of the code evaluation frameworks such as pass-ratio@n, BLEU and CodeBLEU evaluate large language models (LLMs) code generation ability based on their actual execution results on the first prompt without giving feedback of the code execution error. With this in mind, a framework that allows code generated from any large language model to progressively improve was proposed. The idea is to re-prompt the LLMs with the error feedback to generate codes until the code is executable and achieve the user's desire goal or purpose with a predefined number of iterations. If a certain number of iterations has been reached and the code is still not executable or cannot achieve users' intended goals or desire, the prompt would be useful to include in the training dataset for fine-tuning for that large language model. In this study, the application of the proposed framework to enhance the accuracy of mathematical problem-solving problems was tested and reported. This framework may be useful to improve any LLM code generating ability continuously. Discussions were made.

Keywords: generative artificial intelligence, natural language processing, large language models (LLMs), tutoring system, mathematics

1. Introduction

The Large Language Models (LLMs), such as OpenAI's GPT (Generative Pre-trained Transformer) series, have revolutionized the capabilities of Natural Language Processing (NLP) by enabling the processing of vast amounts of text data (Yang et al., 2024). NLP is a branch of artificial intelligence (AI) that aims to enable computers to understand and process language similarly to humans. For instance, natural language descriptions or instructions can be used to direct the focus of a model during training and let it learn specific types of biases or patterns (Liu et al., 2023). The studies on using LLMs like GPT to generate human-like language in education have been widely explored. A number of studies have shown that LLMs can be used to generate step-by-step explanations for solving mathematical problems though lacking a skill of calculating and cannot effectively correct misconceptions (Wardat et al., 2023; Matzakos et al., 2023). Fundamentally speaking, the transformer model (Vaswani et al., 2017)

is not specifically designed to perform arithmetic operations (such as multiplications, square roots, modulus or division); rather, it is designed to perform associations.

On the other hand, LLMs have showed promise in comprehending and generating programming code (Dehaerne et al., 2022). Employing LLMs to generate source code implies utilizing advanced machine-learning models to assist in implementing code for specific tasks or functions (Kazemitabaar et al., 2024; Yeo et al., 2024). By providing a natural language description, LLMs generate the corresponding code (Kong et al., in press). However, the generation of executable programming code from natural language descriptions is a challenging yet crucial area in AI. The produced codes may either fail to execute or not meet the users' goals due to misinterpretations or incomplete translations of the intent. Many existing code evaluation frameworks, such as pass-ratio@n (Yeo et al., 2024), BLEU (Papineni et al., 2020), and CodeBLEU (Ren et al., 2020), assess the code generation ability of LLMs based on the actual execution results of the first prompt, without providing feedback on code execution errors.

In this study, we proposed a LLMs-driven system based on human-AI a novel progressive code generation framework to assist mathematics learning. In many mathematics learning contexts, the goal is not just to find the correct answer, but also to have students demonstrate their understanding by evaluating the correctness of the solution derivation. Conversely, by having the LLM generate code, the newly proposed framework can use human feedback to iteratively improve the model's code generation capabilities and enhance accuracy. All in all, it is hoped that the system will help instructional designers or teachers develop learning materials.

2. Literature Review

The prevalent LLMs, such as GPT and BERT, are primarily built on the transformer architecture (Vaswani et al., 2017), which are able to generate text based on pre-trained associations rather than logical reasoning or computational accuracy. This can lead to what are termed "hallucinations" in generated content, particularly noticeable in tasks requiring precise calculations, such as mathematics (Sun et al., 2024). When using natural language as guidance, these models can adeptly handle text generation but often falter with tasks requiring deep domain knowledge or exact computational outputs. The performance limitations of GPT in mathematical contexts can largely be attributed to this inherent design focus on associative rather than computational capabilities.

LLM generated code evaluation framework (Papineni et al., 2020; Ren et al., 2020; Yeo et al., 2024) has been discussed recently. Usually, the evaluation framework starts with extracting coding problems from various coding platforms to gather relevant coding problems that can be used for further processing by the LLM. Then, using transformer and self-attention mechanisms, the LLM generates code based on the extracted problems. Finally, the LLMs generate code and the evaluation framework analyze and evaluate the quality of the generated code. This is a linear evaluation process starts from preparing the coding problem, through LLM inference, to the final evaluation of the generated code. However, this framework does not let humans in the loop. Using natural language to instruct LLMs is possible and can be integrated into the traditional evaluation framework to help educators to adjust the generated content.

Figure 1 shows the process of integrating prompts in a traditional LLM code generator. First, the user can write prompts at different levels —from simple problem descriptions to more complex instructions with constraints and execution examples (Yeo et al., 2024). These prompts are fed into the LLM code generator, which produces the initial code. This generated code is then subjected to a two-step evaluation process: first, it is tested against predefined test cases, and second, the results are analyzed using specific metrics (Yeo et al., 2024).

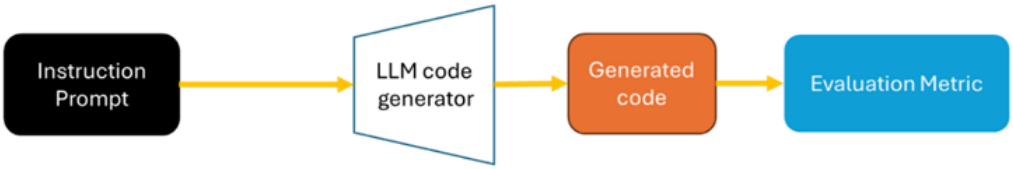

Figure 1. General framework for code evaluation procedure

To sum up, existing evaluation metrics, including pass-ratio@n (Yeo et al., 2024), BLEU (Papineni et al., 2020), and CodeBLEU (Ren et al., 2020), predominantly assess code based on the initial correctness of output. These metrics do not consider the potential for iterative improvement through feedback or the model's ability to correct errors post-deployment (reference needed here). This gap highlights the need for more dynamic and adaptive evaluation frameworks that can capture the ongoing learning and adjustment capabilities of LLMs in code generation tasks.

3. The Proposed Human-AI Progressive Code Generation Framework

The new framework proposed in this study aims to integrate Human-AI progressive code generation framework into the existing code evaluation framework to increase the accuracy of calculating mathematical problems by dynamically improving the code generation through re-prompting the LLM with specific feedback using natural language on execution errors or misalignments with user goals. Figure 2 shows that prompts are dynamically generated according to errors feedback from both code execution error and human evaluator iteratively. Initially, LLM(s) are presented with the Math problem and a prompt. If the generated code is executed with error(s) in the intended code execution environment, the error message will be fed back to the LLM(s) along with all the chat history until the code can be successfully executed. If the code is successfully executed in R iterations, the code output answer will then be evaluated against the correct answer. If the code generated a wrong answer, the evaluation results will be presented back to the LLM in H iterations. If the code generated from the LLM(s) cannot be executed in the intended execution environment under R iterations or (R+H) iterations, we will consider the LLM to have failed this test case in the similar manner to the other framework shown in Figure 2.

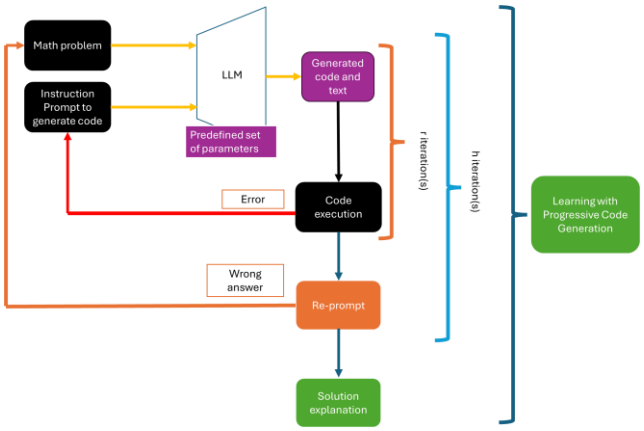

Figure 2. Process in generation llms-generated gode evaluation framework

In this paper, we propose the Progressive Code Generation Framework. The Progressive Code Generation (ProG) Metric derived from several parameters. At the very essence of the framework, the Progressive Code Generation Score must depend on the parameter r and h as follows:

$$ProG = f(r,h) = \frac{1}{e^{(r+h-o)}}$$

Where o refers to the offset. For instance, if the evaluation progress includes further evaluation (i.e. h ≥ 1 and r ≥ 1), o should be set to 2. On the other hand, if the evaluation progress only includes the code execution evaluation process (i.e. h = 0 and r ≥ 1), o should be set to 2. We focused on the case where there is no further evaluation process. Figure 3 shows the Vanilla Progressive Code Generation Framework only considering executable code.

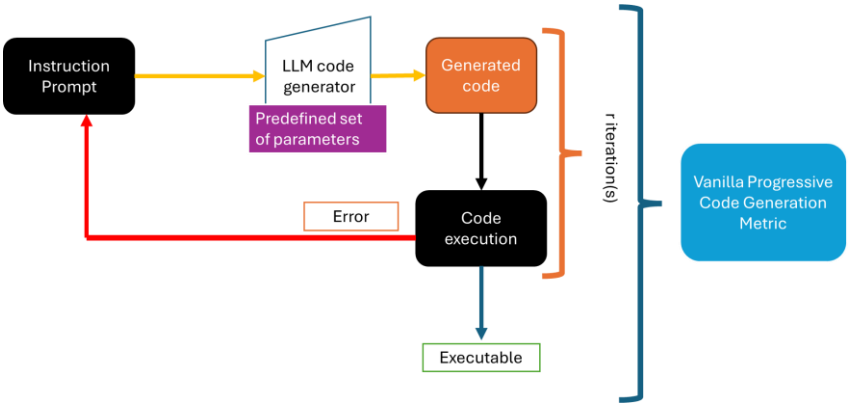

Figure 3. Vanilla progressive code generation framework only considering executable code

The equation can be simplified as follows:

$$vProG = \frac{1}{e^{s(r-1)}}$$

The equation above captures the idea that if the code is executable in the first iteration without any error, the accuracy is 1 (align with other framework shown before and the more feedback it needs to give the correct code, the lower the score it gets. The equation also shows that the parameters can be scaled to allow dynamic scaling if the average number of r iterations required is larger than 5. Where s is defined in the equation below.

4. The Introduction to the System

The following section shows the introduction to the system. Figure 4 shows the interface of the system. The user interface of our system comprised of three main parts. The left most panel list out the Math questions. The middle panel provides a space for taking notes during the interaction with the LLM. The right-hand side of the panel allows teaching experts to interact with the LLM, including calculation through JavaScript and explaining the questions and answers.

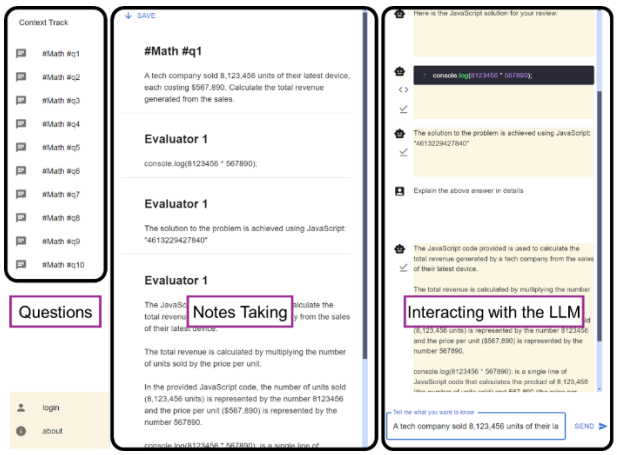

Figure 4. User interface of the system, including parts for selecting questions (left), taking notes (middle) and interacting/ chatting with LLM (right)

Figure 5 illustrates how users can interact with the LLM using our system. Users first choose one of the questions from the question panel or provide a natural language description of the mathematics problem they want to solve. After sending the prompt using the "JS Eval" function, LLM returns JavaScript code to calculate the answer to the question. Once users click "execute code", the JavaScript code is executed, then the calculation's answer is displayed. Users can also ask LLM for explanations on the questions, logic of formulas, or calculations.

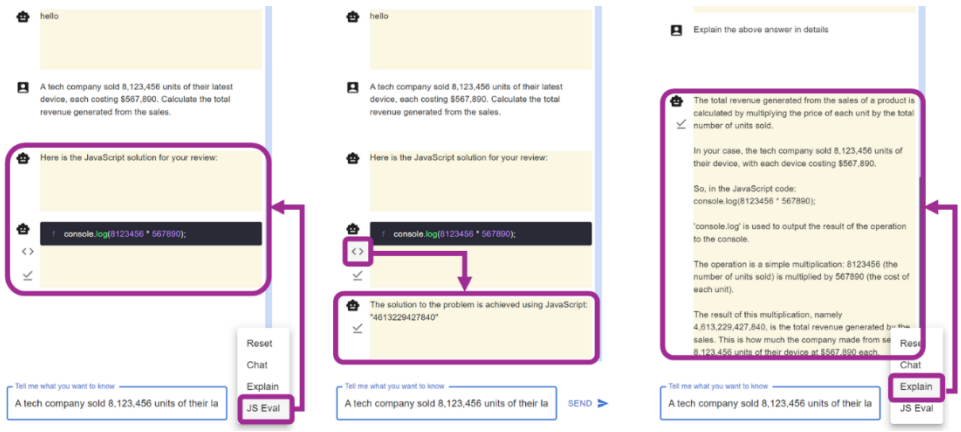

Figure 5. Function of listing formula with JavaScript (left), solving the formula with JavaScript (middle) and generating explanations (right)

After the system generates codes, answers and explanations, users can evaluate the results against the expected solution, and record the useful information with the function of recording and saving the corresponding text. Figure 5 shows the function of note taking of this system.

5. Methodology

In this study, a usability test was conducted to evaluate the effectiveness of the proposed Human-AI progressive code generation framework in solving mathematical problems. The test involved preparing situational questions designed to assess the arithmetic aspects of mathematical problem-solving capabilities. These questions were used to compare the performance of GPT-4 with and without using the progressive generation framework in this study.

Ten situational questions were prepared for testing on different chatbots. Each of them involves multiplication or divisions of multi-digit numbers (answers with more than thirteen digits, or seven decimal places). An example of the multiplication questions is as follows: “A tech company sold 8,123,456 units of their latest device; each costing $567,890. Calculate the total revenue generated from the sales.”

Four researchers and developers were involved in this usability test. Each tester was asked to interact with both GPT-4 with and without using our framework to solve the given questions. Users provided a natural language description of the problem to each chatbot and observed the generated code and solutions. The performance of both chatbots was evaluated based on the feedback of LLMs and the accuracy of the calculation.

6. Results

Our testers test each of the ten questions once on chatbot with GPT-4. It was found that 0 out of 40 trials has returned a correct answer, resulting in a 0% accuracy rate. It was found that the first few digits of the answers given by GPT-4 are often close to the correct answer. The

model's answers also vary across the trials, while some trials appear to be significantly larger in comparison to the other answers, suggesting a lack of consistency in its responses.

7. Discussions and Conclusions

This study shows the potential of a human-AI progressive code generation framework in enhancing the accuracy of answering mathematical multiplication questions in LLMs. The results indicate that the frameworks' ability to adapt to complex arithmetic mathematical tasks more effectively.

Future studies will aim to validate and expand the application of this proposed system in real classroom settings. Thus, both teachers' and students' pattern of prompting and opinions will be collected. In addition, the framework can be revised and used for other disciplines, such as coding education.

Last but not least, although our system allows users to provide error feedback through the "Chat" function, integrating error feedback or code execution feedback error have not been tested extensively. This study only uses a limited set of arithmetic mathematic problems to demonstrate the framework without showing cases with r larger than 1. In the future, we should demonstrate cases where r and h are both larger than 1.

References

Anderson, R. E. (1992). Social Impacts of Computing: Codes of Professional Ethics. Social Science Computer Review, 10(4), 453-469. https://doi.org/10.1177/089443939201000402

Dehaerne, E., Dey, B., Halder, S., De Gendt, S., & Meert, W. (2022). Code Generation Using Machine Learning: A Systematic Review. IEEE Access, 10, 82434–82455. https://doi.org/10.1109/ACCESS.2022.3196347

Kazemitabaar, M., Ye, R., Wang, X., Henley, A. Z., Denny, P., Craig, M., & Grossman, T. (2024). CodeAid: Evaluating a Classroom Deployment of an LLM-based Programming Assistant that Balances Student and Educator Needs. arXiv.Org. https://doi.org/10.48550/arxiv.2401.11314

Kong, S.C., Sit C. Y., Yang, Y., & Yeung, W.K. (2024, in press). One step forward towards the use of human language to instruct computers to work: A reflection on an example of applying prompts in text-based generative AIfor programming. In the Proceedings of the 8th International Conference on Computational Thinking and STEM Education, Beijing, China.

Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., & Neubig, G. (2023). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. ACM Computing Surveys, 55(9), 1-35.

Matzakos, N., Doukakis, S., & Moundridou, M. (2023). Learning mathematics with large language models: A comparative study with computer algebra systems and other tools. International Journal of Emerging Technologies in Learning (iJET), 18(20), 51-71.

Papineni, K., Roukos, S., Ward, T., & Zhu, W. J. (2002, July). Bleu: A method for automatic evaluation of machine translation. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics. (pp. 311-318).

Ren, S., Guo, D., Lu, S., Zhou, L., Liu, S., Tang, D., ... & Ma, S. (2020). Codebleu: A method for automatic evaluation of code synthesis. arXiv preprint arXiv:2009.10297.

Sun, Y., Yin, Z., Guo, Q., Wu, J., Qiu, X., & Zhao, H. (2024). Benchmarking hallucination in large language models based on unanswerable math word problem. arXiv preprint arXiv:2403.03558.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is all you need. arXiv.Org. https://doi.org/10.48550/arxiv.1706.03762

Wardat, Y., Tashtoush, M. A., AlAli, R., & Jarrah, A. M. (2023). ChatGPT: A revolutionary tool for teaching and learning mathematics. Eurasia Journal of Mathematics, Science and Technology Education, 19(7), em2286. https://doi.org/10.29333/ejmste/13272

Yang, J., Jin, H., Tang, R., Han, X., Feng, Q., Jiang, H., … Hu, X. (2024). Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond. ACM Transactions on Knowledge Discovery from Data, 18(6), 1–32. https://doi.org/10.1145/3649506

Yeo, S., Ma, Y., Kim, S. C., Jun, H., & Kim, T. (2024). Framework for evaluating code generation ability of large language models. ETRI Journal, 46(1), 106–117. https://doi.org/10.4218/etrij.2023-0357