Natural Language Processing Term Project: Project Midterm Report

Jinming Zhang jzhang00@g.ucla.edu

Chieh-Hung Cheng james110012@g.ucla.edu

Mingshen Sun sunlq0411@g.ucla.edu

Yachao Mi michael521@g.ucla.edu

1 Teaming Information

This research group consists of four students:

- Chieh-Hung Cheng, MENG-AI, 806129432
- Jimming Zhang, MENG-AI, 50618259
- Mingshen Sum, MENG-AI, 606184617
- Yachao Mi, MENG-AI, 006183324

2 Project Introduction

The focus of this project will be on code generation for mathematical problems. As computational technology continues to thrive, mathematical problems, which used to be calculated and solved solely by human hand, can leverage the power of programming to solve problems both more efficient and arcane. As the application of Natural Language Processing (NLP) becomes more prevalent nowadays, auto-completion of programming codes, as well as chatbots to aid students in homeworks are often observed, but we want to further investigate their capabilities and performance in terms of difficult mathematical problems that are designed to be solved by programming languages.

In this survey, we will investigate the performance of several models, GPT-4, Llama2, and CodeGen, in oreder to test their performances on solving mathematical problems on Project Euler, a series of challenging mathematical programming problems that are designed to solve by writing computer programs. We will collect reference codes and answers to the problems, and then conduct prepossessing to prepare the data for the models to input. The generated codes will be evaluated by the metric we designed to capture how well the model performed on code generation. Last but not least, we'll report the results and give detailed explanation to get the insights in both quantitative and qualitative manners.

3 Work Distribution

Jinming Zhang

- Define a downstream application of LLMs
- Design methods for automated metrics (part 1)
- Prepare the final presentation

Chieh-Hung Cheng

- Identify 1-2 LLMs to evaluate
- Design a set of test data (part 1)
- Prepare the final presentation

Yachao Mi

- Design methods for automated metrics (part 2)
- Interact with LLMs to obtain results
- Evaluate the LLM's generations

Mingsheng Sun

- Design a set of test data (part 2)
- Interact with LLMs to obtain results
- Draft a report with quantitative and qualitative results

4 Literature Review and Baseline Model Selection

4.1 ChatGPT

According to OpenAI[3], GPT-4's versatility in generating code across various programming languages, such as Python, C++, and JavaScript, allows it to perform computational tasks effectively for complex mathematical problems. Its proficiency in interpreting math problems from natural language and offering tailored solutions indicates a high level of understanding specific user inputs.

OpenAI's research titled "Solving (some) formal math olympiad problems"[7] explores a unique approach to tackling formal math problem-solving using a combination of a modified reinforcement learning algorithm and a language model. The reinforcement learning algorithm generates and evaluates symbolic expressions, showing success in solving a subset of math olympiad problems. While acknowledging limitations, the paper emphasizes AI's potential to solve complex mathematical problems and suggests directions for future research and enhancement (OpenAI, 2022).

Another paper "Improving Mathematical Reasoning with Process Supervision"[8] investigates the role of process supervision, a reinforcement learning technique, in training LLMs to improve mathematical reasoning. This study demonstrates that process supervision outperforms traditional outcome supervision, leading to large language models that solve complex math problems with greater accuracy and reliability (OpenAI, 2023).

4.2 Llama2

Llama2 demonstrates a high level of proficiency in coding tasks, performing notably well on benchmarks like HumanEval. This indicates the model's ability to understand and generate code, showcasing significant advancements over its predecessors. In the realm of mathematical problem-solving, Llama2 exhibits capabilities that surpass traditional models. It performs competently on complex benchmarks such as GSM8K, which tests the model's ability to understand and solve grade-school level math problems. The success of Llama 2 in this area can be attributed to its large-scale training that includes diverse mathematical contexts, enabling the model to apply abstract mathematical concepts to specific problems. However, Llama 2 outputs may sometimes include errors, especially in edge cases or highly complex scenarios.

4.3 CodeGen

Proposed by Salesforce Research AI Research in 2023, CodeGen models are autoregressive, transformer-based next-token predicting models that specializes in program synthesis. These models are trained on programming related datasets THEPILE, BIGQUERY, and BIGPYTHON, and further finetuning processes leads to CodeGen models with different sizes and specialize programming languages.

The researchers evaluated the performances both in single-turn and multi-turn fashion. In single turn, the model was asked to complete a single function given the descriptions and evaluate whether it can pass all test cases; while in multi-turn, the model is given prompt each step to generate code segments that fulfill the requirements of the description, then concatenate all output codes together and evaluate the pass rate. The experiments show that the larger the data and model size tends to lead to better performances, as well as multi-step procedures achieves more accurate program synthesis compared with single-step ones.

4.4 Metrics

The automatic evaluation metric BLEU, proposed by Papineni et al. in 2002[9], measures the extent to which a candidate translation matches a set of reference translations. It calculates the overlap percentage between n-grams and introduces a brevity penalty to penalize overly short translations. The introduction of BLEU significantly accelerated research in the field of machine translation. In 2004, Lin proposed ROUGE[6], a new evaluation metric for machine translation based on recall. This standard compares the answers generated by the model with reference answers using recall calculations.

However, recent researchs has identified limitations in these matching-based evaluation criteria. In 2020, Renet et al. pointed out that BLEU fails to capture the unique semantic features of code and introduced a new evaluation standard, CodeBLEU[10]. This method combines traditional n-grams with a matching of grammatical structures through Abstract Syntax Trees (AST) and evaluates semantic similarity through data flow structures. They compared CodeBLEU, BLEU, and accuracy across three tasks: text-to-code, code translation, and code refinement. CodeBLEU was found to perform better than BLEU in these tasks.

In the evaluation of code generation models, merely considering the degree of code matching may overlook programs that implement the same function but do not align well textually. Therefore, the correctness of code often hinges on the execution results. Recognizing this, researchers have shifted towards an evaluation standard centered on functional correctness. In 2019, Kulal et al[5]. introduced a metric such as pass@k, which generates k different outputs for each programming question. A problem is considered solved if any of these outputs successfully solves it. The model's performance is then evaluated by the proportion of all solved problems.

Subsequently, evaluation sets like HumanEVAL, MBPP, and APPS were proposed by Chen et al[2], Austin et al[1] and Hendrycks et al[4]. These sets comprise a wide range of programming problems, each accompanied by detailed documentation and unit tests. As a result, these datasets are now widely used as evaluation standards in the field of code generation and help to unify assessment criteria.

G-EVAL, another novel framework that integrates large language models with a chain-of-thought (CoT) approach in a form-filling evaluation paradigm, specifically designed to enhance the alignment with human judgment. This method structures the evaluation process by generating detailed evaluation steps that guide the assessment, leading to a higher correlation with human evaluations, particularly in open-ended and creative tasks such as text summarization and dialogue generation.

5 Approach and Methodology

Preprocessing We retrieved the mathematical programming problem set from the Project Euler website, as well as reference python codes and answers from Project Nayuki. For preprocessing, we simply remove the HTML related symbols to make it more concise and readable. For the answers, we transform the string into suitable types (integers, ratio, float) to allow better metric calculation in the future.

Metric Design When evaluating the performance of the models, we believe the metrics covered in the literature review section not suitable enough for the mathematical code generating task that we are interested in. However, we indeed are inspired by the ideas of several metric designs. We believe there are some crucial constructs in deciding the quality of the generated codes, and some examples may be,

- Execution: Whether the generated code can be executed without causing error
- Correctness: Whether the generated code can function and result as intended
- Complexity: The time/space complexity when solving the problem
- Readability: How well a human can understand the content of the code
- Proximity: If the answer is incorrect, how far off it is to the correct one
- Others

Those constructs may be correlated with one another, and we want to design a metric that can better capture the quality of the generated codes. Inspired by *pass@k* and *G-EVAL*, we believe combining the two may be beneficial to create a better measure for the designated task. We will first execute the generated codes and find whether the desired answer can be obtained; if not, the model will try again for at most k times, as done in *pass@k* fashion. In the process, the execution, correctness and complexity of the code will be evaluated to form one score. Afterwards, we'll input the code as well as some prompts to let LLMs decide the readability, proximity and other aspects in determining the quaility of the output, gaining the other score. Last but not least, we'll get the final score by taking the weighting average of the model's scores from previous steps, while leveraging the difficulty of each of the questions to get a comprehensive evaluation for the models capabilities in the interested task.

6 Weekly Plan

April 18, 2024:

- Done data pre-processing and preparation
- Complete reading paper of ChatGPT

- Complete the metric selection of our model
- Complete baseline model selection

April 25, 2024:

- Completed literature review
- Discussed the metric design
- Tested out CodeGen model and generated samples
- Tested out GPT-4 model and generated samples
- Tested out Llama2 model and generated samples

Subsequent Work Distribution

- Chieh-Hung Cheng: Deploy CodeGen, set up evaluation pipeline
- Jimming Zhang: Deploy Llama2 to GCP and use pipeline to test the result
- Mingshen Sum: Deploy GPT-4 to GCP and complete test code
- Yachao Mi: Design an evaluation component for both GPT and Llama2

References

[1] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[2] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[3] Arthur Ecoffet, Jonathan Achiam, Stephen Adler, Shubhojit Agarwal, Luqman Ahmad, Ilge Akkaya, et al. GPT-4 Technical Report. March 2024.

[4] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*, 2021.

[5] Sumith Kulal, Panupong Pasupat, Kartik Chandra, Mina Lee, Oded Padon, Alex Aiken, and Percy S Liang. Spoc: Search-based pseudocode to code. *Advances in Neural Information Processing Systems*, 32, 2019.

[6] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

[7] OpenAI. Solving (some) formal math olympiad problems, 2022.

[8] OpenAI. Improving mathematical reasoning with process supervision, 2023.

[9] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[10] Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*, 2020.