Large Language Models (LLMs) are AI-based tools that can understand and reproduce human language. LLMs can generate more accurate responses as they are trained on additional data, making them more popular. These models often work in a chat-based prompt-response system where the LLM produces an output in response to the user's input. Our project is focused on investigating the LLMs' ability to produce high-level language code given a prompt in natural language that describes the code intent. We evaluated the LLMs' ability to generate correct code as many people are currently using these tools for code synthesis by using the following cycle:

1. Prompt
2. LLM
3. Output
4. Evaluation
5. Feedback
6. Iteration

We found that the LLMs are not powerful in understanding numerical data, after consulting with mentors, we decided to change the direction of the project and find a more effective way to use LLMs as code generation tools. This taught me that work is not strict, it is flexible and fluid which I need to be a better researcher and student. We settled on attempting the same project but using prompts instead of input/output examples, through this we found a paper titled, "Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT" in which they used prompts made by HumanEval at OpenAI to test against the LLMs. We decided to use their data because LLMs are constantly updated and using the same information could be used as a verification of their result as we try and reproduce them.

Objective & Impact of Professor’s Research

Professor Mukund Raghothaman’s research is mainly concerned with the inner workings of computer programming languages. Using machine learning and formal verification techniques, Professor Raghothaman works to solve various problems in the field, like improving program synthesis methods and fault localization.

- Machine Learning is an extremely popular field of computer science, concerning teaching a computer how to do a task well by feeding it large amounts of data.
- Formal Verification is determining whether a program completes a job by verifying if it passes a set of mathematical benchmarks.

Method & Results

Procedures

1. Gather prompts from human-evaled, a dataset containing original handmade programming questions publicly available LLMs are unlikely to be trained on.
2. Ask prompts to LLM and gather generated code.
   - Phase 1: Using test cases provided in the human-evaled dataset, evaluate the results of code.
   - Results: 122 Passed, 42 Failed
   - Phase 2: Go through failed test cases and give LLM well selected input/output examples that the code does not satisfy to allow LLM to correct its mistake.
   - Results: 10 Passed, 14 Failed
   - Phase 3: If it fails again analyze the code and correct it to guide it towards its intended purpose using natural language.
   - Results: 9 Passed, 15 Failed

Acknowledgments

We have to give thanks to Professor Mukund Raghothaman and Matin Amini and Sara Baradaran, our PhD student mentors. We would also like to thank the SHINE team, Jayron, Marcus and Monica for helping us along the way outside of the project. I’d also like to give a special thanks to my family for driving me and my friend Ephrem who made this project more enjoyable.

Citations


https://arxiv.org/abs/2304.10778