multiAgent

CS-201 final project. Create a multi-agent system for writing program

Shiding LIn

Abstract:

This project investigates the application of a Multi-Agent System (MAS) to improve the reliability of code generation by Large Language Models (LLMs). The main objective was to establish a system where different agent instances could cross-verify and debug each other's output, enhancing the overall quality of the generated code. I employed a task delegation strategy that differentiated roles among programming, debugging, and quality control agents. The methodology centered on prompt engineering tailored to each agent's role, ensuring precise task execution without overlap.

Introduction:

In recent years, Large Language Models like GPT-3 have revolutionized various domains, including automated code generation. However, these systems often lack self-verification mechanisms, leading to sometimes obvious errors in the output. In addition, There is a limited amount of content a chatbot can keep track of at a time, and as the size of a codebase increase, it gets increasinly more difficult to keep track of all code and To address this limitation, this project explores the use of a Multi-Agent System to enhance the accuracy and reliability of code produced by LLMs. MAS consists of multiple intelligent agents interacting to solve complex problems that are challenging for individual agents. The motivation behind this study is to improve the quality of automated code generation by enabling cross-verification among different agents within the system. The scope involves designing a system where agents are specialized in distinct tasks like programming, debugging, and quality control, allowing them to collaboratively refine the code output.

Literature Review:

Looked at both Gemini and ChatGPT APIs to explore options. ChatGPT's API turned out to be much easier to use, so that's what I went with.

Data Collection and Preprocessing:

Not applicable, no data was used.

Methodology:

The methodology involved designing a three-tier MAS where each agent was responsible for a specific aspect of code generation—programming, debugging, or quality control. I spent the majority of the time on prompt engineering to produce the desired output. The programming agents were tasked with generating initial code based on provided specifications. Debugging agents reviewed this code for errors and inconsistencies, while quality control agents assessed the overall quality and suggested improvements if necessary. If the debugging or quality control agent was not satisfied with the code, it would pass the code back to the programmer agent, along with instructions on how to fix/improve it. The system was built using Python and utilized the OpenAI API for integrating LLMs.

The mechanism by which task is delegated to the agents are by telling the agents to append a string to the output: “program”, “debug”, “quality” and “final”. The program parse the output to see which agent produced the output, and which agent to send it to next.

Experimental Setup:

Tested with various prompts to generate code snippets.

Results:

The MAS system was able to somewhat reliably debug and improve the quality of the code. Sometimes, it will fail to follow instructions, not returning the code snippet, or mistakenly identifying my syntax seperators as part of the code. Through improving the prompts and using gpt-4 instead of 3.5, these errors were minimized.

Discussion (350 words):

The enhanced performance of the MAS in code generation can be attributed to two factors: the the specialized focus of each agent and the ability to double check work by cyclically passing results between agents. By allowing separate agents to focus solely on programming, debugging, or quality control, the MAS framework reduces cognitive load and increases the accuracy of each process. Passing code between agents address one fundamental limitation of LLM, where they cannot directly backtrack and double check their own output halfway through generating a response.

Occasional failures to follow instructions are still unavoidable. Due to the nature of LLMs, there is always a chance for agents to fail to register an instruction.

On a side note, when I asked ChatGPT to improve the prompts, it turns out to be quite bad at prompt engineering. It was not able to produce prompts with better performance than my own.

Conclusion and Future Scope (175 words):

There is still limitations, where currently this can only handle one code snippet at a time. A more complex system would be able to store the generated code in memory, swapping code snippets around, effectively allowing a MAS to keep track of an entire repository. ## References: https://platform.openai.com/docs/overview

https://console.cloud.google.com/

https://www.deeplearning.ai/short-courses/chatgpt-prompt-engineering-for-developers/