LLM Agent Development for Python Coding Assistance

Suvrat Pal

May 5, 2024

1 Introduction

The aim of this project was to develop an LLM (Large Language Model) based agent to assist in coding tasks specifically for Python programming language. The agent was designed to provide code suggestions, debugging tips, and documentation assistance based on user queries. This report summarizes the approach, implementation details, testing, and potential improvements for the developed agent.

2 Agent Development

For this project, Python was chosen as the target programming language due to its popularity and widespread use. The agent was developed using the Gemini-Pro model, a pre-trained generative model accessible through the Google Generative AI API. This model was selected for its capabilities in generating human-like text responses.

3 Requirements

- Python 3.x
- Tkinter (Python GUI library)
- Google Generative AI API
- IPython (for displaying Markdown)
- ctypes (for adjusting DPI)
- threading (for smooth user experience during API calls)

4 Features Implementation

The agent implemented three main features:

1. **Code Completion**: Users could input partial code snippets, and the agent would generate completions based on the provided context.
2. **Debugging Assistance**: The agent was capable of suggesting fixes for common syntax or logical errors in the user's code.
3. **Documentation Retrieval**: Users could request information about specific Python functions or libraries, and the agent would provide relevant documentation.

5 Integration and Testing

The agent was integrated into a simple user interface (UI) built using Tkinter, a standard Python GUI toolkit. Users could interact with the agent by selecting options for code completion, debugging assistance, or documentation retrieval. Threading was utilized to prevent the UI from freezing during API calls, ensuring a smooth user experience.

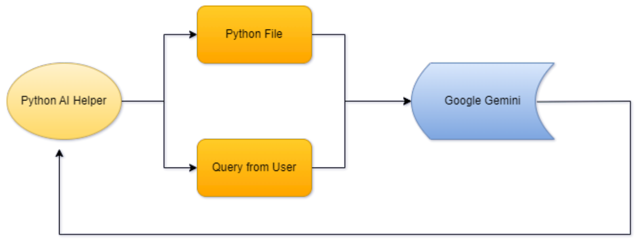

Figure 1: Flow Diagram

Basic testing was conducted to evaluate the agent's performance in understanding user queries and providing helpful responses. This included unit tests for individual features and user testing to assess overall usability.

6 Evaluation and Optimization

The agent's performance was evaluated based on accuracy, response time, and relevance of the output. While the agent demonstrated promising results in providing accurate responses, response time could be improved, especially for longer queries. Additionally, the relevance of code completions and debugging suggestions could be enhanced through further fine-tuning of the model.

Potential improvements include:

- Optimizing API calls to reduce response time.

- Fine-tuning the model on a larger corpus of Python code to improve accuracy.
- Implementing user feedback mechanisms to iteratively improve the agent's performance over time.

7 Conclusion

In conclusion, the developed LLM-based agent shows potential in assisting Python programmers with code completion, debugging, and documentation retrieval tasks. While the agent provides valuable assistance, there is room for optimization and refinement to further enhance its usability and effectiveness. With continued development and iteration, the agent has the potential to become a valuable tool for Python developers seeking coding assistance.