CS224G Personal Insights

Xinyun(Tess) Tao

This quarter, our team works on Py-Assistant, a LLM-driven platform that supports software developers in three essential areas: code translation, code generation, and code summarization.

Our app leverages open source LLM including gpt-3.5-turbo, and claude-1.3 as the backend models, and generates answers based on manually crafted prompt and user input. Since we want our LLM to excel in the three specific tasks, we added a preliminary step in our prompting to first analyze user intent, and determine whether it belongs to one of the three coding tasks. This step helps to ensure that our task-specialized chatbots will avoid answering questions in unrelated areas, and could incorporate the determined intent into the final prompt for enhanced accuracy. When experimenting with different prompts, we discovered that it could be hard to clearly define the categories. For instance, if we are too ambiguous about what “code generation” means, the model would classify arbitrary questions as code generation; on the other hand, instructing the model to search for keywords such as “summarize” would make the LLM fail to identify user input that doesn’t include these keywords. We ended up using detailed, carefully explained definitions to handle more edge cases, but the performance could still be unstable across multiple re-runs.

Additionally, we enhanced our application's functionality by integrating a RAG system for code summarization, which significantly improved the preciseness and detail of responses by utilizing data extracted from provided documents. Our exploration included two prompting strategies: a restricted mode, where RAG was confined to using only the fetched information as context, and an unrestricted mode, allowing RAG to draw upon its original training knowledge base. We discovered that RAG will consistently prioritize the retrieved context, but incorporating external knowledge proved beneficial for tasks that required generating comprehensive, executable code snippets. For instance, if we query a general-purpose LLM about “how to build a prompt”, the model tends to provide merely a basic grammatical structure. However, submitting specific tutorials from Langchain and consulting RAG could offer detailed guidance on creating a Prompt template in Langchain. We learned from class that techniques like chain-of-thought and RAG significantly enhance LLM performance, and by implementing an actual RAG system, we've firsthand witnessed its potential to elevate.