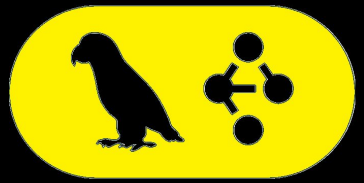
Code Generation-Explanation Agent using LangGraph



Abdelrahman Elaraby

The Flow of the App:

1. Get user query.
2. Classify the intent of the user whether: explain code, generate code, or other.
3. If the user needed to generate code, then we use the retriever to get similar code snippets from our vector database, in which there are 164 code examples from the human eval dataset. Then we use the top 3 relevant code snippets as a context in the prompt to generate new code.
4. If the user needed to explain code, we check whether the user typed a python function or class to explain, otherwise we explain the last generated code.
5. If the user query was neither a code generation or explanation we handle it with a different node in the graph.

The openai/openai_humaneval dataset:

The HumanEval dataset released by OpenAI includes 164 programming problems with a function signature, docstring, body, and several unit tests. They were handwritten to ensure not to be included in the training set of code generation models.

Link: https://huggingface.co/datasets/openai/openai_humaneval

The Vector Database, and the Embeddings:

I used FAISS (Facebook AI Similarity Search) Database in this project.

And for the embedding model, I used: BAAI/bge-code-v1, which is specialized in creating vector embeddings for code samples and retrieving them.

The LLM Backbone:

The main LLM that I used in comprehending the retrieved codes and creating new ones, and explaining, and the general replies is the same LLM which was: mistralai/mistral-7b-instruct:free from OpenRouter.

The reason for choosing this model is that it worked well for my task (Compared to Llama) and it is free and available most of the time.

The AgentState:

The Langgraph requires a state to flow between the nodes of the graph. I chose to implement the state to include the following data:

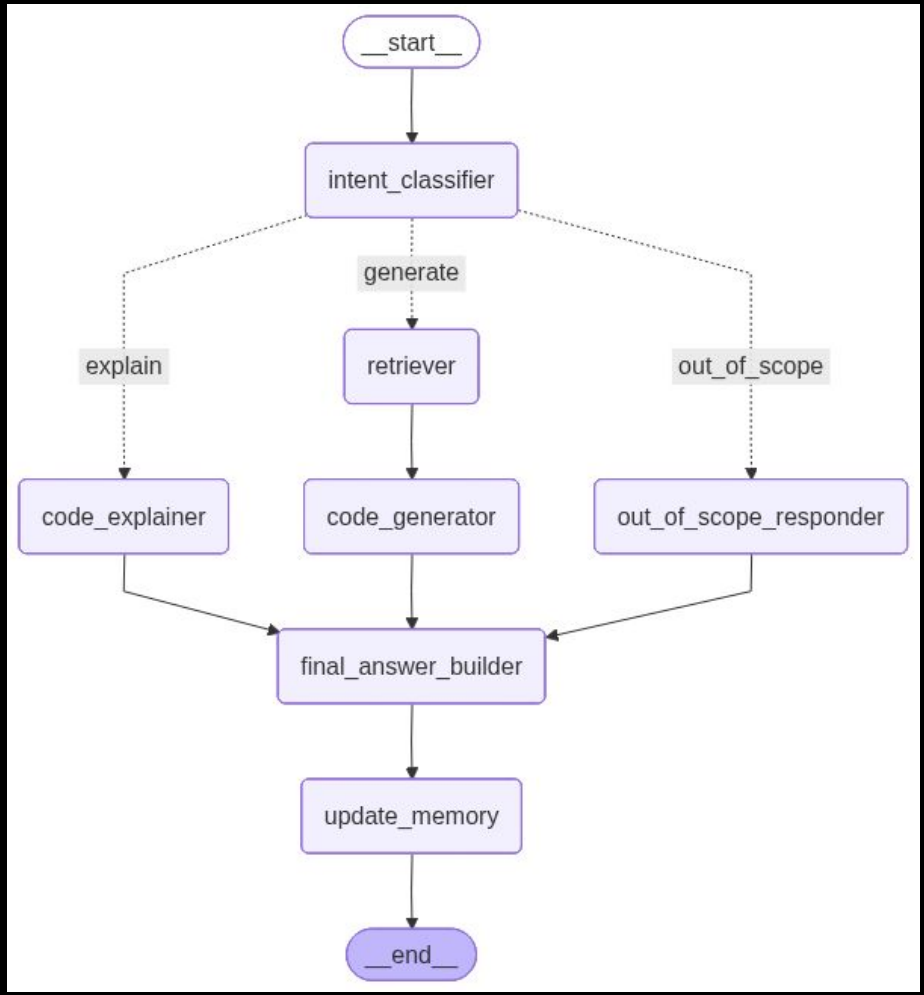
```
class AgentState(TypedDict):

    user_input: str
    intent: Optional[str]
    retrieved_docs: Optional[List[Document]]
    generated_code: Optional[str]
    explanation: Optional[str]
    final_answer: Optional[str]
    memory: List[Dict[str, str]]
```

The Graph:

I created 7 nodes in my graph and made each one specialized in one task.

The intent_classifier has two versions, one with simple hard coded rules and another one that uses LLM to classify.



The Interface:

I made the interface using the Streamlit Library.

THANK YOU.