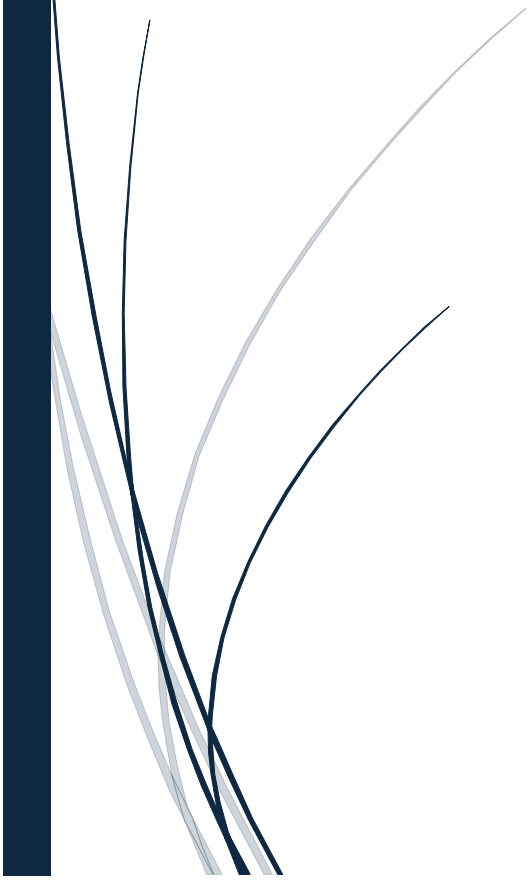
Code Assistant



Problem Statement

Developers often struggle to understand unfamiliar code, leading to inefficiencies in debugging, modifying, and maintaining software, which can be alleviated by an AI-powered code assistant that generates descriptions and answers questions about the code.

Why this is a problem

Understanding code—especially when it's complex, poorly documented, or written by someone else—can be time-consuming and frustrating for developers. Whether working on legacy systems, reviewing open-source projects, or onboarding new team members, developers frequently spend hours deciphering code logic, function purposes, and dependencies. Traditional documentation, if available, is often outdated or incomplete, forcing developers to rely on trial and error or lengthy code reviews.

An AI-powered code assistant solves this by automatically generating clear descriptions of code functionality and allowing developers to ask questions through an interactive chatbot. This not only accelerates comprehension but also reduces cognitive load, improving productivity and enabling faster troubleshooting and iteration. Such a tool is particularly valuable in large-scale projects, cross-functional teams, and fast-paced development environments where rapid understanding of code is critical.

Proposed Solution

The proposed solution is an AI-powered code assistant that allows users to upload code files, which are then automatically processed to generate structured descriptions. These descriptions, including explanations of functions, classes, and key logic, are compiled into a downloadable document for easy reference. Additionally, users can interact with an AI chatbot to ask specific questions about the uploaded code, receiving clear and contextual responses. The interface will be intuitive, offering a seamless file upload experience, real-time progress tracking, and an interactive chat window for inquiries. This approach ensures that developers can quickly understand unfamiliar code, reducing time spent on manual documentation and improving overall efficiency in software development and maintenance.

Tools Used

LLM → gpt-4o-mini

I selected GPT-4o-mini as the primary LLM because it sufficiently meets the current feature requirements while remaining cost-effective.

Embedding model → text-embedding-3-small

In this phase, the data primarily consisted of simple English with a limited dataset, making a smaller model sufficient. However, I probably will use a more advanced model capable of understanding code for improved performance later.

Orchestration

I am using LangChain, LangGraph as the orchestration frameworks to streamline interactions between the LLMs, embedding model, and vector databases. LangChain was chosen for its modular design, extensive integrations, and ability to manage complex retrieval-augmented generation (RAG) workflows efficiently.

Monitoring

I will use **LangSmith** to monitor the LLM's performance, track API usage, and log responses in real time.

Evaluation

I am using Ragas to evaluate retrieval quality and response accuracy, ensuring the system provides relevant and reliable outputs.

User Interface

For the user interface, I chose **Chainlit** due to its ability to rapidly prototype chatbot systems while providing robust features for interaction and debugging. Its lightweight framework enables quick iterations, making it an ideal choice for efficiently testing and refining the chatbot's functionality.

Agents

In my solution, an agent will be used to fetch information about libraries and coding updated information and provide code descriptions.

Another agent will be used to write the documentation.

External API

search_pypi

A custom tool to call PyPI external api to return description of a certain Package.

This tool is used to give more reliability and updates information to the model when analyzing the imported packages in the code.

Data Sources

1. ## User Code

   User code will be used as the main data source to build the knowledge base for RAG

2. ## Generated code -> Description data (to be used in the future)

   This dataset will be used to enrich the model with relative examples on how to generate a description of a code block (function, class, etc).

   This dataset will be used for evaluation as well

3. ## Generated code -> documented code (to be used in the future)

   This dataset will be used to enrich the model with relative examples on how to generate ‘docstring’ of a code block (function, class, etc).
   This dataset will be used for evaluation as well

Chunking Strategy

For code-based data, the most effective strategy is to chunk the code into logical blocks, such as functions and classes, while keeping related code grouped together. This structured approach enhances retrieval accuracy and improves overall performance by preserving the context of each code segment, making it easier to understand and analyze.

RAGAS Evaluation

The flow of the solution is as follows

1- User provides code file

2- Code is analyzed
3- Using an agent, description for this code is generated and inserted in a vector store
4- This vector store along with another model are used for RAG
5- Using this RAG user can ask questions about code and get the answer back

RAGAS evaluation was done on the description RAG part

Results:

```
Evaluating: 100%|██████████| 50/50 [00:33<00:00,  1.48it/s]
{'faithfulness': 0.7352, 'factual_correctness': 0.6570, 'context_recall': 0.9500, 'llm_context_precision_with_reference': 0.8000, 'answer_relevancy': 0.3706}
```

Conclusion:

- Metrics related to context (recall and precision) have high scores since the chunks in vector store are very few at the moment. Evaluation will be more informative as more chunks are generated in the future
- Faithfulness and factual correctness are average. This indicates more enhancements are needed to be done on the prompt.

IMPORTANT NOTE

Why was tuning an embedder not included in this version/prototype?

I made a great effort to incorporate all requirements into this midterm; however, time was not on my side. Given that my data is code-based, I am still in the process of identifying and fine-tuning a suitable embedding model that can work well with both human language and code.

Future Enhancements

1- Making my solution able to process a whole file with all kinds of code blocks
2- Making my solution able to process more than one file
3- Parsing code in a more effective way
4- Using a suitable embedder for code
5- Building a solid knowledge base that's able to link related code with each other.
6- Giving the user another version of the code that's documented along with the code description document.