🧠 CodePilot

Authors

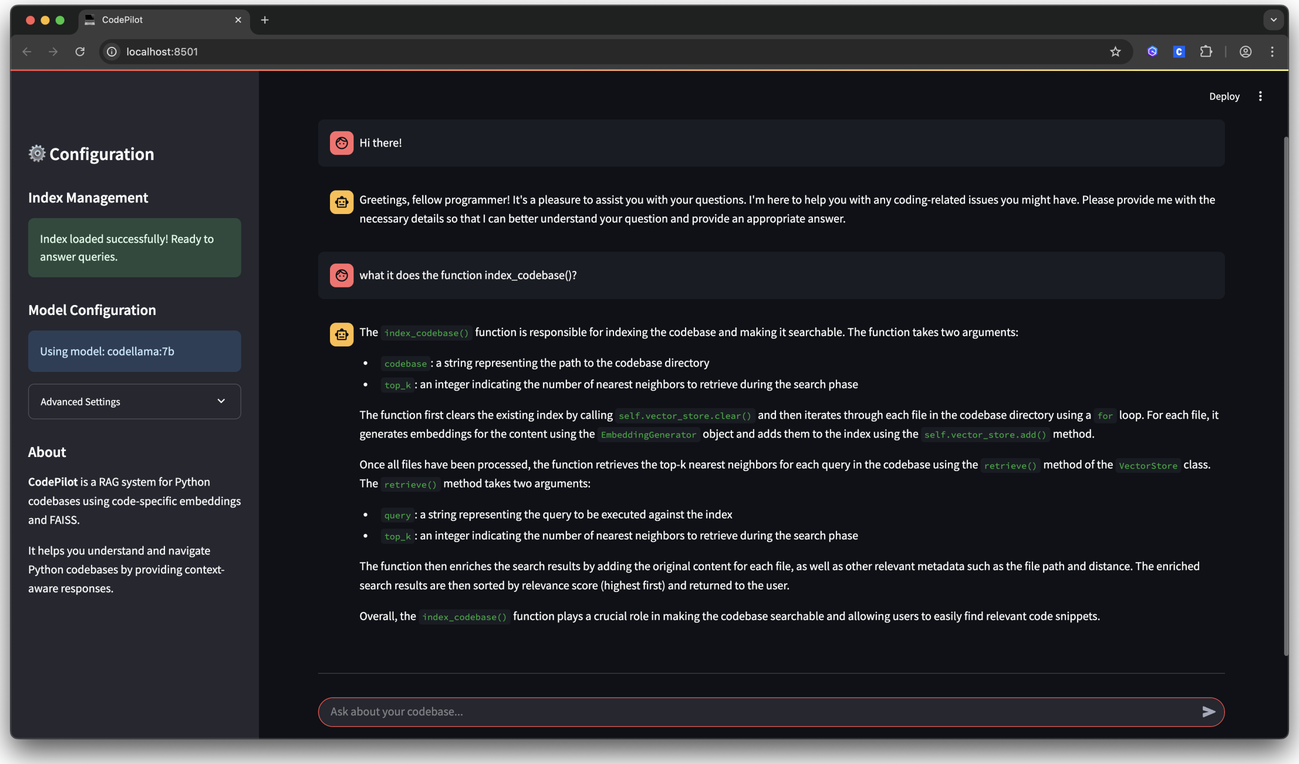
- Anass Anhari
- Pol Vidal
- Eric Muthomi

CodePilot is a simple POC (*Proof Of Concept*) local agent for natural language understanding and querying of Python codebases. It uses a RAG (*Retrieval Augmented Generation*) architecture that parses, embeds, and retrieves code snippets to help developers explore and understand Python projects efficiently, and most importantly, with no internet connection 😁 !

The system is designed to be modular and extensible, allowing for easy integration of different models and components, hence, it can be adapted to various use cases and programming languages (AST Parsing, ...). The architecture consists of several core modules, including a code processor, vector database, engine, and user interface.

Examples

Here are some examples of CodePilot in action:



⚙️ Configuration

Index Management

Index loaded successfully! Ready to answer queries.

Model Configuration

Using model: codellama:7b

About

CodePilot is a RAG system for Python codebases using code-specific embeddings and FAISS.

It helps you understand and navigate Python codebases by providing context-aware responses.

Hi there!

Greetings, fellow programmer! It's a pleasure to assist you with your questions. I'm here to help you with any coding-related issues you might have. Please provide me with the necessary details so that I can better understand your question and provide an appropriate answer.

what it does the function index_codebase()?

The `index_codebase()` function is responsible for indexing the codebase and making it searchable. The function takes two arguments:

- `codebase` : a string representing the path to the codebase directory
- `top_k` : an integer indicating the number of nearest neighbors to retrieve during the search phase

The function first clears the existing index by calling `self.vector_store.clear()` and then iterates through each file in the codebase directory using a `for` loop. For each file, it generates embeddings for the content using the `EmbeddingGenerator` object and adds them to the index using the `self.vector_store.add()` method.

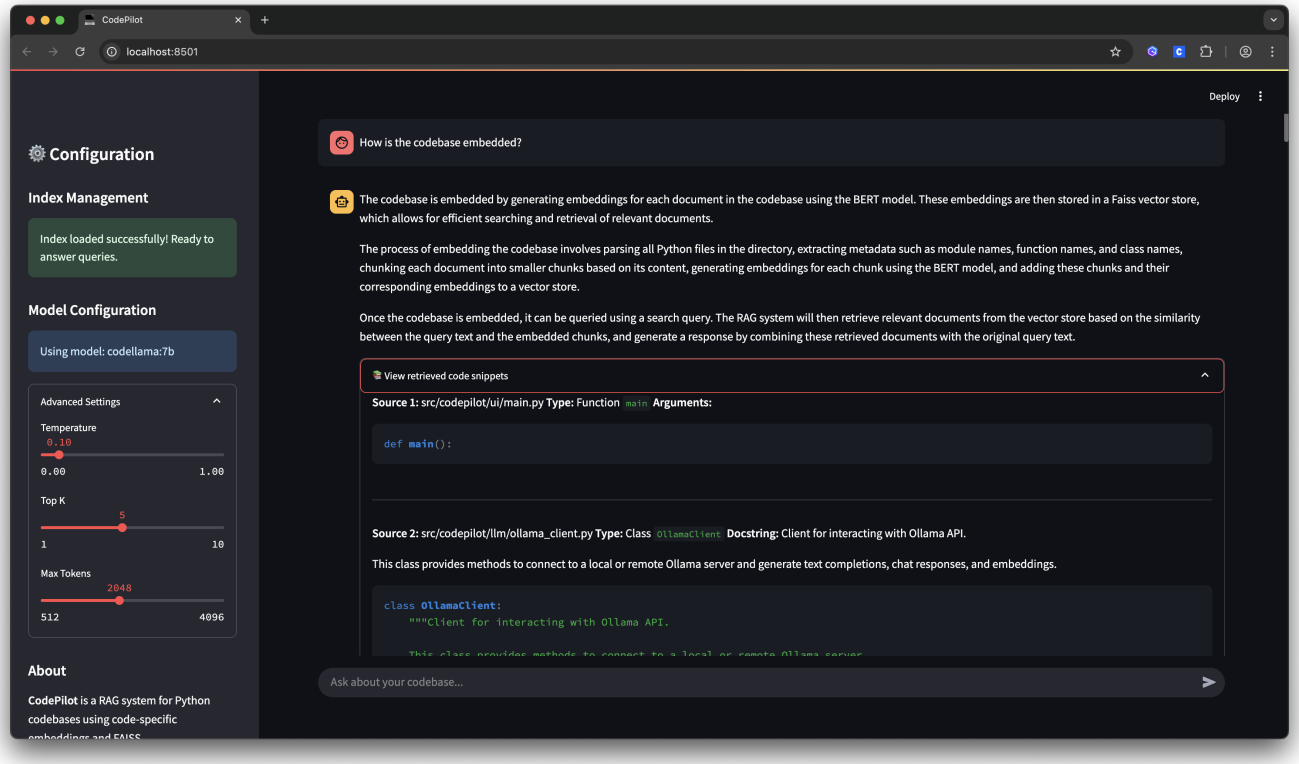
Once all files have been processed, the function retrieves the top-k nearest neighbors for each query in the codebase using the `retrieve()` method of the `VectorStore` class. The `retrieve()` method takes two arguments:

- `query` : a string representing the query to be executed against the index
- `top_k` : an integer indicating the number of nearest neighbors to retrieve during the search phase

The function then enriches the search results by adding the original content for each file, as well as other relevant metadata such as the file path and distance. The enriched search results are then sorted by relevance score (highest first) and returned to the user.

Overall, the `index_codebase()` function plays a crucial role in making the codebase searchable and allowing users to easily find relevant code snippets.

Ask about your codebase...



⚙️ Configuration

Index Management

Index loaded successfully! Ready to answer queries.

Model Configuration

Using model: codellama:7b

Advanced Settings

Temperature 0.10 (0.00 – 1.00)

Top K 5 (1 – 10)

Max Tokens 2048 (512 – 4096)

About

CodePilot is a RAG system for Python codebases using code-specific embeddings and FAISS

How is the codebase embedded?

The codebase is embedded by generating embeddings for each document in the codebase using the BERT model. These embeddings are then stored in a Faiss vector store, which allows for efficient searching and retrieval of relevant documents.

The process of embedding the codebase involves parsing all Python files in the directory, extracting metadata such as module names, function names, and class names, chunking each document into smaller chunks based on its content, generating embeddings for each chunk using the BERT model, and adding these chunks and their corresponding embeddings to a vector store.

Once the codebase is embedded, it can be queried using a search query. The RAG system will then retrieve relevant documents from the vector store based on the similarity between the query text and the embedded chunks, and generate a response by combining these retrieved documents with the original query text.

📚 View retrieved code snippets

Source 1: src/codepilot/ui/main.py **Type:** Function `main` **Arguments:**

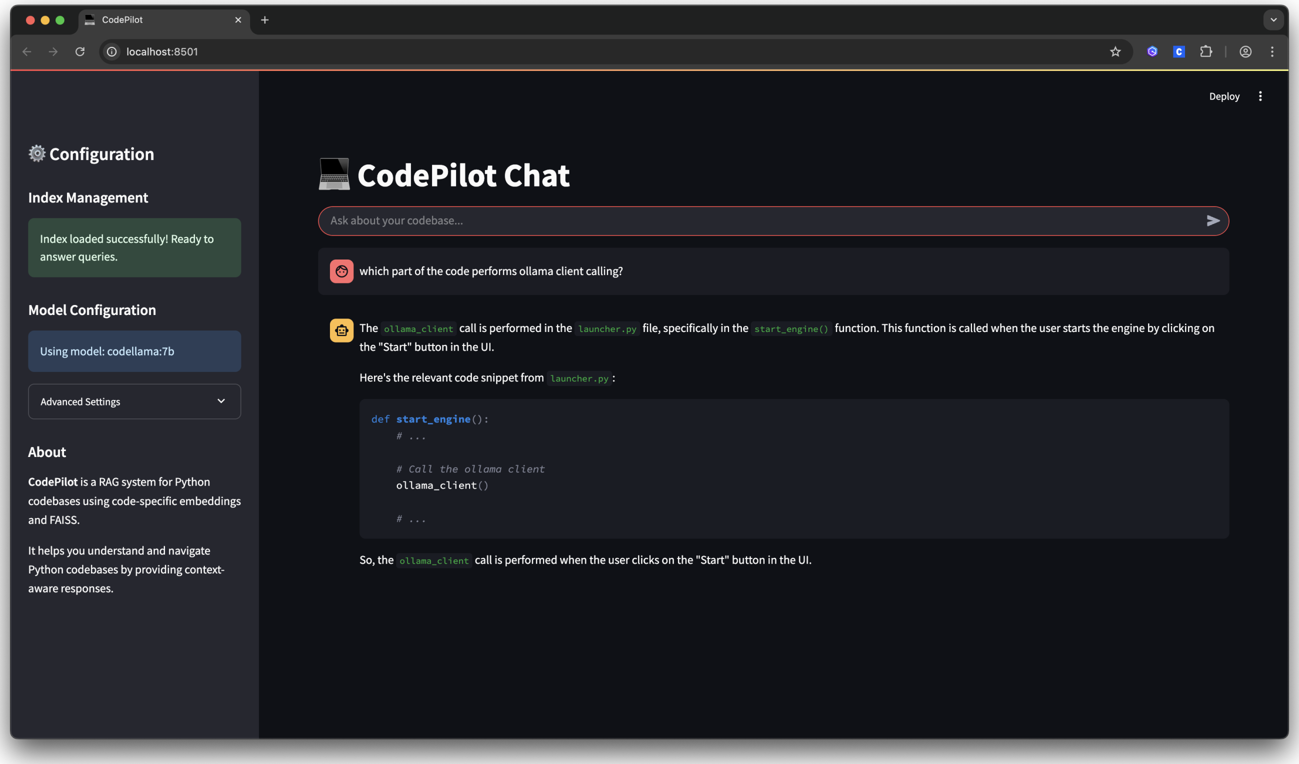
```
def main():
```

Source 2: src/codepilot/llm/ollama_client.py **Type:** Class `OllamaClient` **Docstring:** Client for interacting with Ollama API.

This class provides methods to connect to a local or remote Ollama server and generate text completions, chat responses, and embeddings.

```
class OllamaClient:
    """Client for interacting with Ollama API.

    This class provides methods to connect to a local or remote Ollama server
```



⚙️ Configuration

Index Management

Index loaded successfully! Ready to answer queries.

Model Configuration

Using model: codellama:7b

Advanced Settings

About

CodePilot is a RAG system for Python codebases using code-specific embeddings and FAISS.

It helps you understand and navigate Python codebases by providing context-aware responses.

💻 CodePilot Chat

which part of the code performs ollama client calling?

The `ollama_client` call is performed in the `launcher.py` file, specifically in the `start_engine()` function. This function is called when the user starts the engine by clicking on the "Start" button in the UI.

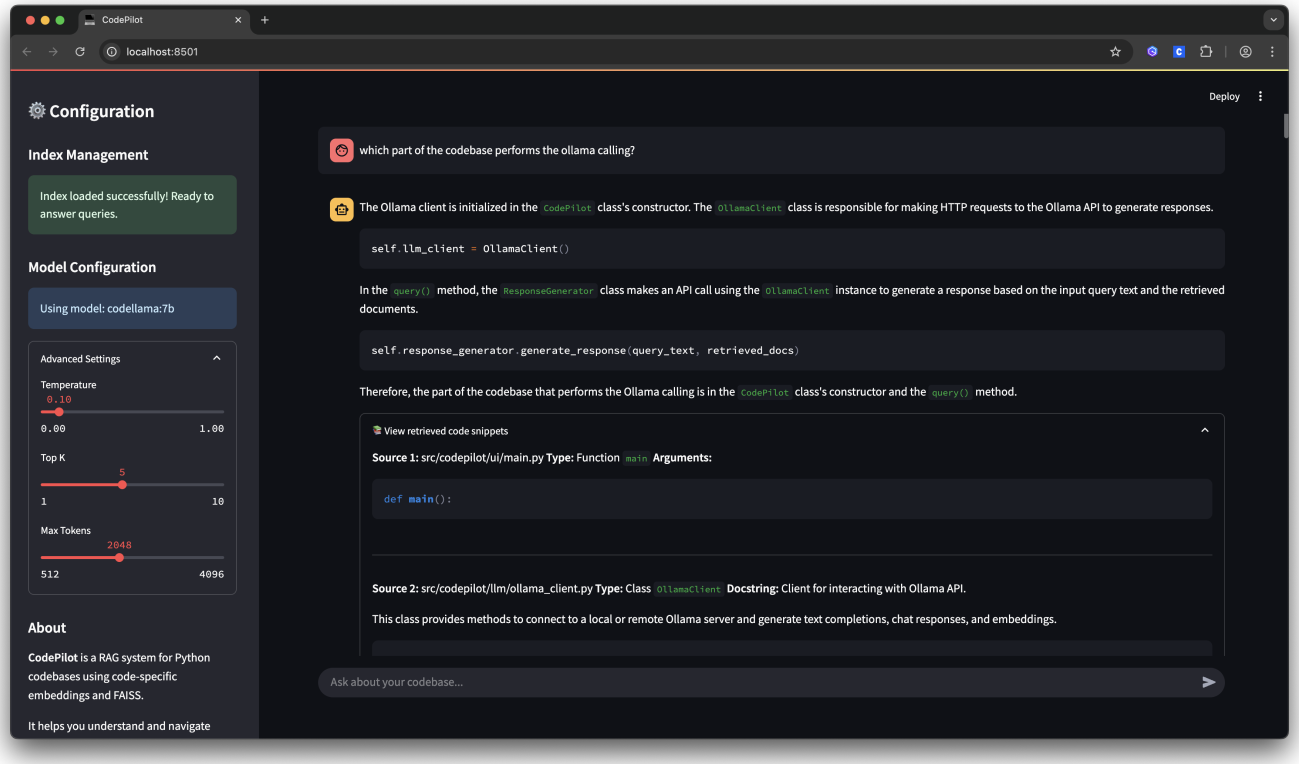
Here's the relevant code snippet from `launcher.py`:

```
def start_engine():
    # ...

    # Call the ollama client
    ollama_client()

    # ...
```

So, the `ollama_client` call is performed when the user clicks on the "Start" button in the UI.



⚙️ Configuration

Index Management

Index loaded successfully! Ready to answer queries.

Model Configuration

Using model: codellama:7b

Advanced Settings

Temperature: 0.10 (0.00 – 1.00)

Top K: 5 (1 – 10)

Max Tokens: 2048 (512 – 4096)

About

CodePilot is a RAG system for Python codebases using code-specific embeddings and FAISS.

It helps you understand and navigate

which part of the codebase performs the ollama calling?

The Ollama client is initialized in the `CodePilot` class's constructor. The `OllamaClient` class is responsible for making HTTP requests to the Ollama API to generate responses.

```
self.llm_client = OllamaClient()
```

In the `query()` method, the `ResponseGenerator` class makes an API call using the `OllamaClient` instance to generate a response based on the input query text and the retrieved documents.

```
self.response_generator.generate_response(query_text, retrieved_docs)
```

Therefore, the part of the codebase that performs the Ollama calling is in the `CodePilot` class's constructor and the `query()` method.

📚 View retrieved code snippets

Source 1: src/codepilot/ui/main.py **Type:** Function `main` **Arguments:**

```
def main():
```

Source 2: src/codepilot/llm/ollama_client.py **Type:** Class `OllamaClient` **Docstring:** Client for interacting with Ollama API.

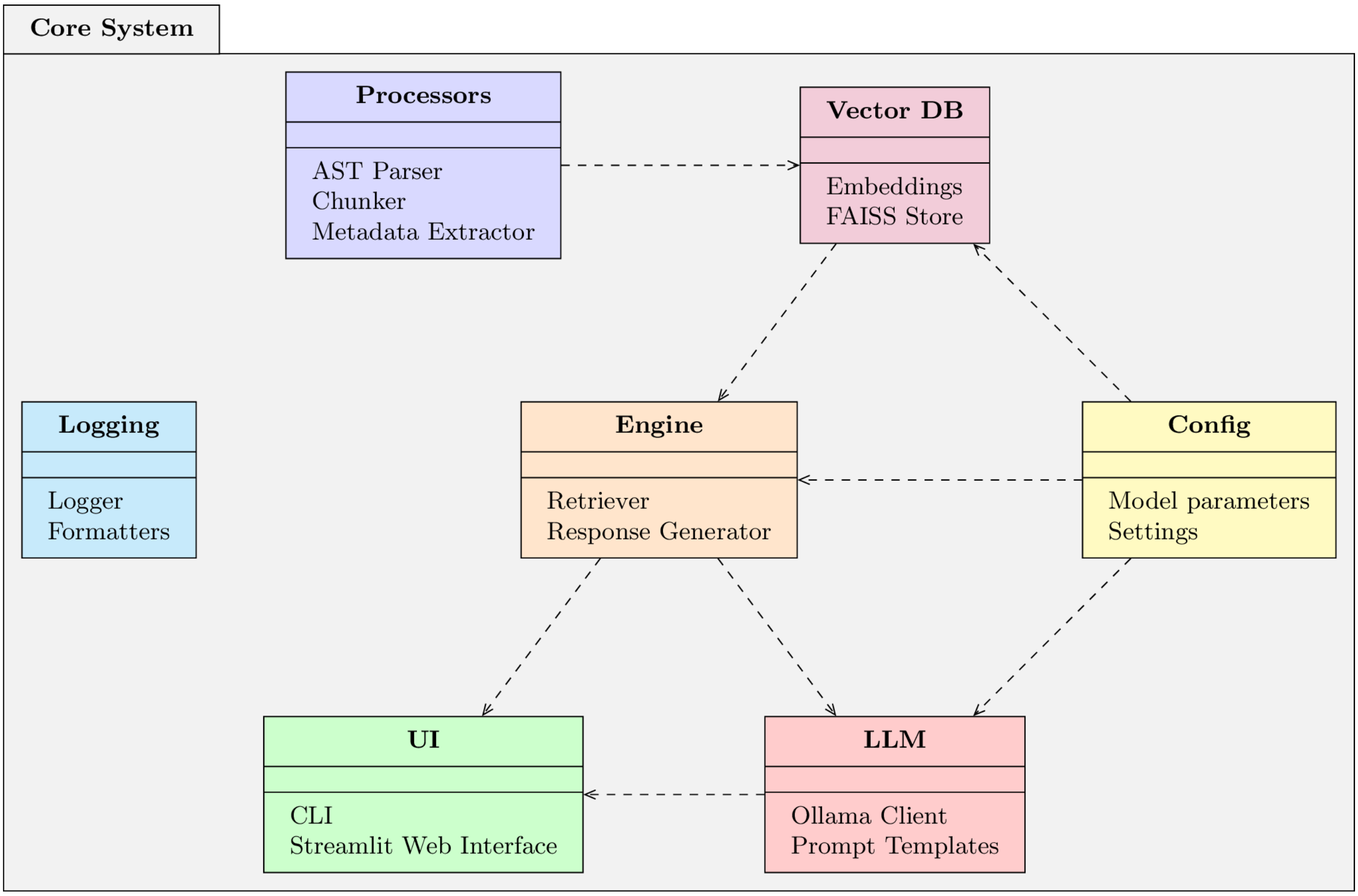
This class provides methods to connect to a local or remote Ollama server and generate text completions, chat responses, and embeddings.

Ask about your codebase...

Project Structure

CodePilot is organized into several modules:

- **Processors**: Parse and process Python code files
    - `ast_parser.py`: Uses Python's Abstract Syntax Tree to extract code structure
    - `chunker.py`: Splits code into manageable chunks for embedding
    - `metadata_extractor.py`: Extracts additional metadata from code
- **Vector DB**: Manages the vector database for code retrieval
    - `embeddings.py`: Generates embeddings using the CodeBERT model
    - `faiss_store.py`: Stores embeddings in a FAISS index for efficient similarity search
- **LLM**: Handles interactions with language models
    - `ollama_client.py`: Client for interacting with Ollama API
    - `prompt_templates.py`: Templates for structuring prompts to the LLM
- **Engine**: Core retrieval and response generation
    - `retriever.py`: Finds relevant code snippets
    - `response_generator.py`: Generates natural language responses
- **UI**: User interfaces
    - `main.py`: Streamlit web interface
    - `launcher.py`: Script to launch the Streamlit UI



Installation

Prerequisites

- Python 3.8 or higher (>=3.11 recommended)
- Ollama installed and running locally with the CodeLlama model

Step 1: Clone the repository

```
git clone https://github.com/Anass-23/CodePilot.git
cd CodePilot
```

Step 2: Install the package

> Recomended to use a virtual environment

```
python3.11 -m venv env
source env/bin/ctivate
pip install -U pip
```

This will install CodePilot and all its dependencies

```
pip install -e .
```

How to Run the Agent

CodePilot offers two interfaces: a command-line interface and a web-based UI.

Command-Line Interface

1. **Index a codebase** - Before querying, you need to index a Python codebase:

```
codepilot --index /path/to/your/python/project
```

2. **Query the codebase** - Once indexed, you can query the codebase:

```
codepilot --query "How does the retriever module work?"
```

3. **Interactive mode** - Run without arguments to enter interactive mode:

```
codepilot
```

Web-Based UI (Streamlit)

Start the web interface with:

```
codepilot-ui
```

> Note: Before quering, you also need to index the codebase

This launches a Streamlit app where you can:

- Ask questions via a chat interface
- Configure model parameters

💬 Usage

Example Queries

Here are some examples of questions you can ask:

- "What does the `generate_response` method in the `ResponseGenerator` class do?"
- "Show me how chunks are processed in the codebase"
- "Explain the vector store implementation"
- "How does the system handle errors?"

Configuration

CodePilot's behavior can be customized by modifying the `config.py` file:

- **EMBEDDING_MODEL**: The model used for creating embeddings (default: `microsoft/codebert-base`)
- **MODEL_NAME**: The LLM model to use via Ollama (default: `codellama:7b`)
- **TEMPERATURE**: Controls randomness in responses (default: `0.1`)
- **TOP_K**: Number of code snippets to retrieve (default: `5`)

How It Works

1. **Indexing**: CodePilot parses Python files using the AST module, chunks the code, generates embeddings using CodeBERT, and stores them in a FAISS index.

2. **Querying**: When you ask a question, CodePilot:

   - Embeds your query
   - Finds similar code snippets in the vector database
   - Sends the relevant context along with your question to a language model
   - Returns the generated response

3. **Local Processing**: All processing happens on your machine - no code is sent to external servers 😁

License

This project is licensed under the MIT License - see the LICENSE file for details.