CodePilot Lite
An NLP-Powered Codebase Question Answering Agent

Group Members: [Anass, Eric, Pol]

April 2025

1 Project Title

CodePilot Lite: An NLP-Powered Codebase Question Answering Agent

2 Objective

Problem Statement

Understanding unfamiliar codebases is a common challenge in software development. Developers often spend significant time navigating through multiple files to comprehend the purpose and structure of the code. Our goal is to develop a lightweight assistant that can answer natural language questions about a Python codebase.

3 Historical Context

The complexity of modern software systems has steadily increased, necessitating better tools for navigating and understanding large codebases. Early tools such as grep and ctags helped with code search and navigation. With the evolution of integrated development environments (IDEs), static analysis and autocompletion tools became common, offering partial support for code comprehension. More recently, machine learning and NLP advancements—particularly large language models (LLMs)—have enabled intelligent code understanding, code summarization, and even generation capabilities. Tools like GitHub Copilot, Amazon CodeWhisperer, and Tabnine now integrate code-aware AI directly into developer workflows, laying the groundwork for conversational and assistant-style interfaces.

4 State of the Art

The current frontier in AI-assisted programming includes tools like OpenAI Codex (used by GitHub Copilot), DeepMind's AlphaCode, and Salesforce CodeGen. These models are capable of not only generating code but also answering natural language questions about existing codebases, performing code summarization, and completing functions based on intent. Our approach draws from this progress by implementing a mini-RAG pipeline tailored for static Python codebases. Unlike fully integrated code-generation tools, our system focuses on explainability and transparency, leveraging prompt engineering and retrieval techniques to ground LLM responses in actual code.

Relevance and Novelty

This project combines natural language processing techniques with static code analysis and retrieval-augmented generation (RAG) to provide insightful responses. It enables faster understanding of codebases and helps developers quickly find relevant functions, classes, or modules.

5 Data or Knowledge Sources

Input Format

Plain-text `.py` files from a Python project directory.

Data Sources

We will use:

- A synthetic multi-file Python project for development.
- A small open-source GitHub project for testing.

Preprocessing Steps

1. Parse code using Python's `ast` module to extract:
   - Function names and parameters
   - Docstrings and imports
   - Class definitions
2. Chunk the parsed code (by function/class level).
3. Generate semantic embeddings for each chunk.

6 Methodology

System Overview

The system consists of three major modules:

1. **AST Parsing**: Static code analysis using Python's `ast` module to extract structured metadata.

2. **Embedding & Retrieval**: Embed each code chunk using OpenAI or HuggingFace models and store them in a vector database (e.g., FAISS or Chroma).

3. **Query & Answering**: Retrieve the most relevant code snippets and generate a response using an LLM with a custom prompt via LangChain.

Response Generation

We use Retrieval-Augmented Generation (RAG):

- The user query is embedded and matched against stored vectors.
- Top-k relevant code snippets are used as context.
- The query and context are formatted into a prompt template.
- A language model (e.g., GPT-3.5) generates a natural language response.

Tools and Libraries

- Python `ast` module
- LangChain
- OpenAI API / HuggingFace Transformers
- FAISS or ChromaDB
- Streamlit (optional interface)

7 Expected Outcome

A working command-line or notebook-based system that:

- Accepts natural language questions
- Retrieves and analyzes relevant code segments
- Responds with a clear, informative answer

8 Evaluation

Functional Testing

We will create test questions such as:

- "What does `funcs.sum()` do?"
- "Where is the `User` class defined?"

Answers will be evaluated manually for correctness.

Optional Metrics

If labeled examples are used, we may compute:

- Precision@k
- Similarity scores

9 Timeline and Task Breakdown

Task	**A**	**B**	**C**	**Time Estimate**
AST Parser and Metadata Extraction	X			4 hours
Embeddings and Chunking		X		4 hours
Vector Database Setup		X	X	4 hours
LangChain Integration			X	3 hours
Interface and Testing	X		X	4 hours
Documentation and Report Writing	X	X	X	4 hours

Table 1: Timeline and task distribution among group members.

10 Creativity and Innovation

The system is modular, interpretable, and easy to extend. It's a practical example of applying state-of-the-art NLP (RAG, embeddings, LLMs) to the software engineering domain. The fallback "I don't know" response encourages trustworthy AI behavior.

11 Final Deliverables

- **Codebase**: Well-organized and documented
- **README**: Instructions to run and test the assistant
- **Report**: System overview, design, and evaluation
- **Demo**: Live walkthrough of question-answering on a Python project