Git Repo Codebase RAG System

Sunny Zhou

Motivation

Although there isn't much on my GitHub, I would still like the projects on the the files that are there are not the most readable. Not only that, but I believe that an interactive element to exploring my code would make it more memorable to people who visit. Originally, I had the following goals with the project:

> "I would be happy if this assistant would be able to generate and return accurate work chunks with explanations relevant to the users' queries or questions. Any other goals spelled out below, such as the tone and some hallucination concerns, will be cherries on top. The evaluation for this assistant would come with a ground-truth (i.e. did it retrieve the intended document), which accuracy would work fine for, and a more open-ended metric (e.g. how well did it answer a clarifying question); the latter being more difficult to answer and requiring human evaluation."

I was not able to tune for tone of the chat assistant, but I made significant progress to the retrieval functionality and the assistant is able to give bits of explanation. Although exact code chunks are still not able to be presented, there is a certain level of reasoning that the LLM is able to do on its own.

Data Processing

Each repo was collected and removed of git files, detaching from the remote branches to prevent the continuous history tracking. This was all done through the shell script clone_all.sh. After, the documents were semantically chunked using LangChain's language specific textsplitter, which allowed python documents, for example, to be split up by their methods. I started out with chunks of size 600. I also allowed overlapping of the chunks. Later, I decreased the chunking sizes to condense the information, attempting to gather better embeddings. All files that didn't fit into the provided file types were ignored. Additionally, data files were also ignored since it provides little content to the work I done. I also cut out a project because it was introducing too much data balancing issues.

The data was stored in a json files (as seen in the data directory) in the following format:

```
{
        "document_name": document,
        "chunk_id": "{num:0{width}}".format(num=chunk_num, width=5),
        "content": content,
        "project_name": project,
        "address": path,
        "type": file_type,
        "doc_chunks": [],
        "project_chunks": []
    }
```

The metadata (i.e. project_name, address, type, doc_chunks, and project_chunks) were meant to provide cross document and chunking information, preserving a certain level of context for after retrieval. I got a total of 15963 chunks.

Methodology

Pipeline

There were a few different parts a query has to go through for a generation:

`Input -> Summarize -> Context/Code Retrieval -> Prompt Formatting -> Chat LLM -> Output`

After each user query, the query is fed into a summarizer, which condenses the query into what exactly about the codebase is being questioned. This allows for better retrieval. These chunks are then formatted and

added onto the beginning of the original prompt:

```
Relevant documents:
Project Name: {name}
        Document: {doc}
            Path: {path}
            Type: {type}
            Content: \'{text}\'
Please answer the following question about the documents: {message}.
Please provide explanation and be willing to provide more information.
If the input is not a question, please respond accordingly.
```

Only after everything, then is it fed into a final chat model, which provides the final response.

Chat

I started the project with using transformer pipeline to generate text, simulating chat. Llama 3.2 1B did not perform very well at summarizing the queries. I found that larger models through the transformers framework took too long to get generations, so I end up transitioning all generation to ollama. This allowed me to use Mistral 7B Instruct as the central chat model. All messages are appended to a history to provide a chat history. However, this history is not persistent between sessions.

The chat model system has been tweaked to prevent off topic conversations and clarity: >«SYS» >You are a helpful project expert. You are to pose as the author of all of the presented documents. You know everything there is about the documents provided to you and how to provide clarifying explanations about them. You must answer as straightforwardly as possible and highlight important parts of the codebase and the reasoning behind it. >For each response, please provide the excerpts that are the most relevant in your opinion, where they are in the code base, and what other documents the user should look at. >Before responding, you must determine what documents provided are relevant. You must phrase your response using only the relevant documents. Do not reveal information about the documents that are not relevant. >If you don't know how to answer the question, please say I don't know. If the question doesn't require the provided documents, please choose not to talk about them. >«SYS»

RAG

The design of this project went through a few iterations. First, all of the documents chunks were added into a single FAISS index. The search was done using IndexFlatL2. However, due to the inaccuracies of the embeddings, L2 exacting matching characteristic is not completely often gathered irrelevant files. Instead I moved to using IndexHNSWFlat to using a nearest neighbor approach. Moreover, moved all README.md and .tex files to a separate Index. From initial testing, I found that the retrieval would often miss the project referenced in the query entirely, which made it impossible for the model to make any inference about what the project could do. These files in particular were moved because they offered the most explanation to the design decisions and explanations because they often were used as the write up or paper portion of assignments and projects.

Evaluation

The evaluation shows how the performance of the RAG system is far from perfect. I collected test data by prompting ChatGPT to generate RAG test questions from a variety of documents (these questions can be found in the test_questions.tsv). Some of these questions were extremely vague and didn't reference an exact file in the question itself. This made it extremely difficult for the retriever to identify the files needed for the query. Nevertheless, these questions were used to test the general abilities of the system.

In general, I found that, as expected, the system had a hard time at finding relevant documents, often directly mentioning that the provided files were not enough to answer the question. Despite this, sometimes it would give a best guess given the context clues as to what the file could be talking about. Additional, testing of the system prompt could help this performance.

The table below shows the macro precision, recall, and F1 of ROUGE-1, ROUGE-2, and ROUGE-L. It is interesting though that the higher scores were in recall for ROUGE-1 and ROUGE-L, and not ROUGE-2.

Metric	Precision	Recall	F1 Score
ROUGE-1	0.0430	0.5110	0.0778
ROUGE-2	0.0077	0.0913	0.0140
ROUGE-L	0.0328	0.3990	0.0594

This suggests the response is providing a pretty good context and explanations, but the phrasing is not in the same exact way as the answer. Additionally, as expected, the low precision suggests that the response is WAAAY longer than the expected answer, which also forces down the F1 scores. Although, this is not necessarily a problem given we want a more human response.

Delivery

There a two ways to test the RAG system out for yourself. For a both methods to run smoothly, you will also have to install the following packages using your preferred package manager: - ollama - tqdm - pandas - faiss - sentence_transformers

If you are using pip to install the packages, you can simply run:

```
pip install ollama tqdm pandas faiss sentence_transformers
```

The first way will run the system in the terminal. To run enter the following command in your terminal:

```
python main.py
```

the chat model will launch in the browser. It will take a minute to launch, but afterwards you will be met with a loading screen, after which you can start interacting with the assistant.

Second, I have made a very bare bones site with Streamlit. You can run this app by running the following command in the terminal:

```
streamlit run app.py
```

This site will also take a second to load, after which there will be a sidebar and a chat UI in the middle. The sidebar will help you navigate through the projects and what files there are to ask about. Keep in mind that this interface doesn't actually let you see any the files yet.

Known Bugs

During each call of the model, Streamlit will print this error in the terminal, which I don't know how to fix:

```
2025-05-07 00:50:13.135 Examining the path of torch.classes raised:
Traceback (most recent call last):
  File "/Users/local/miniconda3/envs/rag/lib/python3.12/site-packages/streamlit/web/bootstrap.py", line
    if asyncio.get_running_loop().is_running():
       ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
RuntimeError: no running event loop

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/Users/local/miniconda3/envs/rag/lib/python3.12/site-packages/streamlit/watcher/local_sources_wa
                                                                                    line 217, in get_module_paths
    potential_paths = extract_paths(module)
                      ^^^^^^^^^^^^^^^^^^^^^
```

```
  File "/Users/local/miniconda3/envs/rag/lib/python3.12/site-packages/streamlit/watcher/local_sources_wa
                                                                                    line 210, in <lambda>
    lambda m: list(m.__path__._path),
                   ^^^^^^^^^^^^^^^^
  File "/Users/local/miniconda3/envs/rag/lib/python3.12/site-packages/torch/_classes.py", line 13, in __
    proxy = torch._C._get_custom_class_python_wrapper(self.name, attr)
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
RuntimeError: Tried to instantiate class '__path__._path', but it does not exist! Ensure that it is regi
                                                                                                  torch
```

For now please ignore this error.

Future Directions

There are a lot of future implementations I would like to do.

1. Git History Credits Currently this codebase has a lot of files that are not completely my work. I want to provide credit to my teammates, but that will require for me to parse and add information from the git tree/history, which would have been too much for this project.
2. File Explorer The side bar in Streamlit should do more than provide a list of files you can ask about. I would like to add more interaction with this app where clicking the button of a particular file will show you the content complete with syntax highlighting. This should dictate what documents/project the Index can search. Additionally, queries should be able to change the file that you are looking at. Depending on the prompt, important portions should be highlighted as well.
3. Better Indexing and Context Similar to the previous direction, more work as to be done with the overall retrieval mechanism. Additional Indexes can be added and finetuning of the embeddings can help. The context retrieval can also include docstrings,which could be used to implement reranking. All in all, this requires more documentation work than anything.