Git Inspector! Inspecting Github Repositories with Open-Source LLMs

Razvan Florian Vasile [1]

[1]Computer Science, University of Bologna

Introduction

Suppose a dedicated engineer exploring unknown Github repositories struggles with the steep learning curve inherent to new codebases, where traditional LLMs often struggle with due to the sheer amount of information available. Could an assistant facilitate this process, helping them explore and understand the code? I explore the task of using **Open-Source LLMs** to generate responses to **questions about Github repositories**.

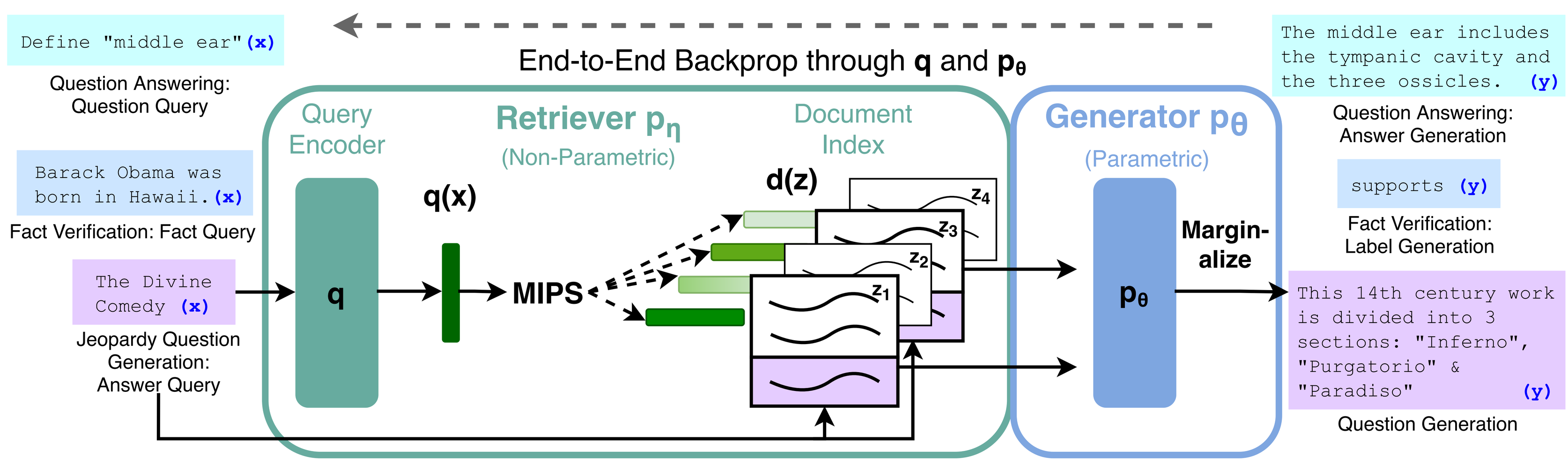


Figure 1. The RAG Architecture. Taken from [3].

Problem Statement

- **Identify the Problem:** Traditional LLMs struggle with knowledge-intensive tasks, such as specialized domain questions, fact verification, or specific code related queries.
- **Put it into context:** Answering questions about codebases may exhibit these limitations.
- **Find the root cause:** The model often fails to locate precise codebase passages, leading to incomplete or inaccurate answers.
- **Ideal outcome:** Provide the model with relevant information to generate accurate responses.
- **Propose a solution:** Develop a method to expand prompts with essential information.

Results: Generated Responses with Citations

Experiments using Qwen-2.5 Coder 32B

OpenAI Scala Client Functionalities

What functionalities does the openai-scala-client offer?

Source1. File name: /service/OpenAIService.scala

```
/**
* Central service to access all public OpenAI WS endpoints [...].
* The following services are supported:
* - Models: listModels, and retrieveModel
* - Completions: createCompletion
* - ChatCompletions: createChatCompletion, [...] createChatToolCompletion
* - Edits: createEdit (deprecated)
* - Images: createImage, createImageEdit, createImageVariation
* - Embeddings: createEmbeddings
* - Batches: createBatch, retrieveBatch, cancelBatch, and listBatches
* - Audio: createAudioTranscription, createAudioTranslation
* - Files: listFiles, uploadFile, deleteFile, retrieveFile, and retrieveFileContent
* - Fine-tunes: createFineTune, listFineTunes, [...], and deleteFineTuneModel
* - Moderations: createModeration
* - Threads: createThread, retrieveThread, modifyThread, and deleteThread
* - ThreadMessages: createThreadMessage, retrieveThreadMessage, [...]
* - Runs: createRun, etc.
* - RunSteps: listRunSteps, etc.
```

Answer. The `openai-scala-client` offers a comprehensive set of functionalities [...]. Here's a breakdown:

1. **Model Operations**: List models [...], Retrieve details [...].
2. **Completions**: Generate text completions [...].
3. **Chat Completions**: Create chat completions [...]. With optional tool calls.
4. **Edits**: Create edits to text (deprecated).
5. **Images**: Generate images, Edit or vary existing images [...].
6. **Embeddings**: Create embeddings for texts [...].
8. **Audio**: Transcribe audio, Translate languages, Generate speech [...].
10. **Fine-tuning**: Create fine-tuned models, Manage the lifecycle of fine-tunes.

The openai-scala-client provides a comprehensive Scala interface to OpenAI's API, supporting models, completions, chat, images, embeddings, audio, fine-tuning, moderation, threads, and run operations.

The Scala 3 Language

What are some key features of Scala 3?

Source1. File name: /docs/_docs/pattern-matching.md

The implementation of pattern matching in Scala 3 was greatly simplified [...]. From a user perspective, this means that Scala 3 generated patterns are a lot easier to debug.

Source2. File name: /scala2-library-cc/src/scala/language.scala

```
/**
* The language features are:
* - Dynamics [...] enables defining calls rewriting using the Dynamic trait
* - Existentials [...] enables writing existential types
* - ImplicitConversions [...] enables defining implicit methods and members
* - ReflectiveCalls [...] enables using structural types
```

Source3. File name: /docs/_docs/reference/overview.md

To enable porting most uses of macros, we are experimenting with the advanced language constructs listed below. [...]

- **MatchTypes** [...] allow computation on types.
- **Inline** [...] provides a straightforward implementation of some simple macros [...].
- **QuotesandSplices** [...] provide a natural way to express macros and staging [...].
- **Typeclassderivation** [...] creates an in-language implementation of the `Gen` macro.
- **By-nameparameters** [...] a more in-language implementation of the `Lazy` macro.

Answer. Scala 3 introduces several key features:

1. **Option-less Pattern Matching**: Scala 3 simplifies pattern matching, making it easier to debug by [...] ensuring all variables are visible in debug modes.
2. **Extractor Support**: Scala 3 supports a superset of Scala 2's extractor patterns.
3. **Language Feature Imports**: The scala.language object allows for selective import of language features [...]

Scala 3 introduces significant language improvements including simplified pattern matching with better debugging, enhanced extractor support, selective language feature imports, and advanced metaprogramming constructs.

Methods: How is the Vector Store generated?

- **Github Repositories:** We have a number of repositories, each with a number of files, labeled as $r_1, r_2, \ldots, r_n$.
- **File Segmentation:** For each repository r_i we extract the code files. Let's call the k-th file of repository i as $f_{i,k}$.
- **Semantic Chunking:** For each file $f_{i,k}$, we extract the semantic chunks. Let's call the these chunks $c_{i,k,1}, c_{i,k,2}, \ldots, c_{i,k,m}$.
  We then use the embedding function f (as described in [4]) to convert each chunk into a 768-dimensional vector: $e_{i,k,l} = f(c_{i,k,l})$. Here, $e_{i,k,l}$ is the embedding of the l-th chunk from the k-th file of the i-th repository.
- **Vector Database:** The numerical features are stored in the Qdrant vector database D.
- **User Query:** When a user submits a query q, we:
  1. Convert the query to an embedding using the same function: $e_q = f(q)$.
  2. For each retriever, use Cosine similarity to find top 50 closest matches in the vector database D.
  3. As described in [2], use a reranker to find the top 3 most relevant chunks from the top 100 matches.

Why use two embedding models?

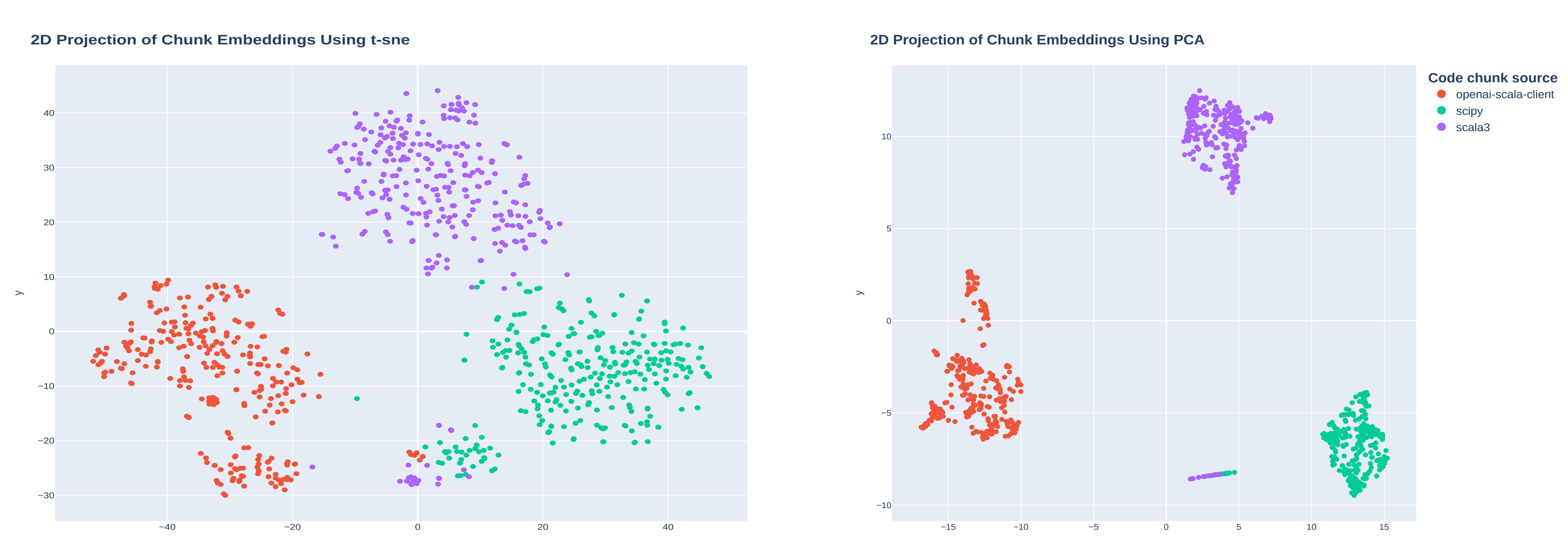
- **Query Embedding [1].** Small (161M parameters) yet powerful model optimized for code retrieval tasks. Trained for both text and code modalities.
- **Semantic Chunking [2].** Comparatively, larger model (278M parameters) designed for reranking a large number of documents. It seeks to improve search accuracy by analyzing the semantic meaning of the search query and the corpus to search over.

Model Name	Use case	Dim.	Size	Score Function
jina-embeddings-v2-base-code [1]	querying	768	161M	Cosine Similarity
jina-reranker-v2-base-multilingual [2]	reranker	N/A	278M	Relevance Score

Table 2. Properties of the models used to generate the vector store, embed queries and rerank documents.

Analysis: 2-Dimensional Cluster Visualization

- **PCA.** Is a linear dimension reduction technique that seeks to maximize variance and preserves large pairwise distances. In other words, items that are different end up far apart.
- **t-SNE.** It differs from PCA by preserving only small pairwise distances or local similarities. Gives an intuition on how data is arranged in higher dimensions.



(a) t-SNE 2D Projection

(b) PCA 2D Projection

Figure 2. Three clusters: Open-AI-Scala-Client (red), Scipy (green), Scala 3 (purple).

Pairwise Heatmap Embeddings

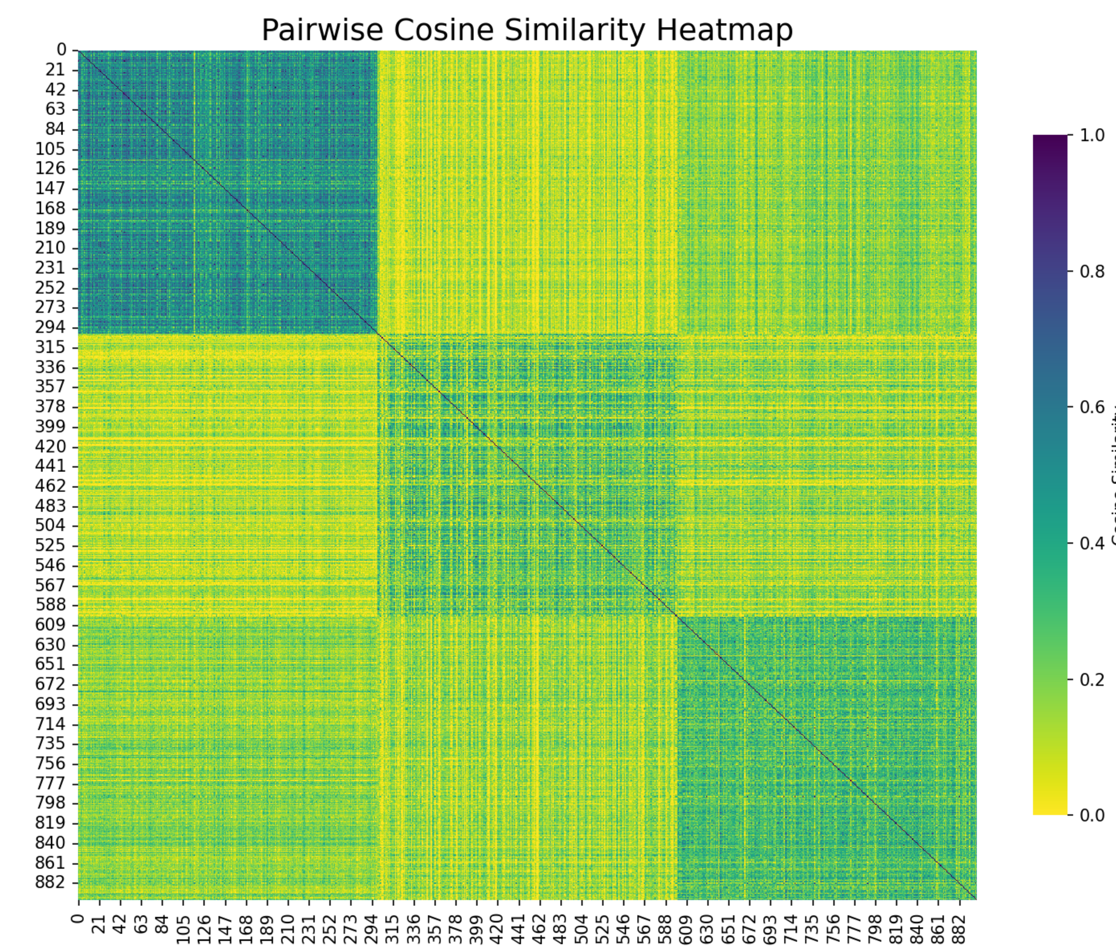


Figure 3. Upper left corner: Open-AI-Scala-Client, Middle: Scipy, Bottom Right: Scala 3.

- **Dark colors** represent lower pairwise distances, indicating **more similar pairings**.
- **Lighter colors** represent higher pairwise distances, indicating **less similar pairings**.

Conclusion

- **Local inference.** The model generally requires strong LLMs such as *Qwen 2.5 Coder 32B* in order to adequately use code retrieval workflows. Many other models are available through Ollama.
- **Future models.** Due to the fact that the Git Inspector library is usable locally as well as via remote inference, the project is likely to scale well as stronger models become available.

Future Work

- **Integrate with other services.** Use other resources made available through the Langchain4j API, including the ability to search Wikipedia or to query the arXiv knowledge database.
- **Enable Multi-agent workflows.** Allow users to interact with multiple agents by asking and answering questions and engaging in interactive conversations [5].

References

[1] jinaai/jina-embeddings-v2-base-code · Hugging Face. `https://huggingface.co/jinaai/jina-embeddings-v2-base-code`, January 2024.

[2] jinaai/jina-reranker-v2-base-multilingual · Hugging Face. `https://huggingface.co/jinaai/jina-reranker-v2-base-multilingual`, September 2024.

[3] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, and Mike Lewis. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2020. `arXiv:2005.11401`.

[4] sentence-transformers/multi-qa-mpnet-base-dot-v1 · Hugging Face — huggingface.co. `https://huggingface.co/sentence-transformers/multi-qa-mpnet-base-dot-v1`.

[5] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework, 2023.

razvanflorian.vasile@studio.unibo.it