The Future of AI-Powered Conversations - Swisscom Customer Support Chatbot using RAG

Team "underfitted"

October 27, 2024

1 Introduction

This hackathon project aims to build a question-answering chatbot using Swisscom's website data to enhance customer interactions by overcoming the limitations of large language models (LLMs) through Retrieval-Augmented Generation (RAG).

2 System Design

- **Architecture Overview**: The RAG system combines a retrieval module with a language model to provide accurate, context-driven responses. Document chunks relevant to the user query are retrieved from a vector store, re-ranked for relevance, and then passed to the language model as context. This architecture enables the chatbot to deliver fact-based, traceable responses grounded in the retrieved information.
- **Key Components**:
  - **LangChain**: Provides integration for language models and vector stores, facilitating the development of RAG pipelines.
  - **Chroma**: A vector store used to store and retrieve document embeddings for similarity search.
  - **VoyageAI (Model and Embeddings)**: `voyage-3` model is used to generate document embeddings, with `rerank-2` for re-ranking retrieval results.
  - **OpenAI GPT-4o**: Powers the LLM, delivering responses based on prompt inputs and retrieved context.
  - **Python Libraries**: Includes libraries like `transformers` for tokenization, `getpass` for secure key entry, and standard libraries for data handling.

3 Data Preparation

- **Dataset**: The dataset consists of Swisscom's public website data in multiple languages (English, German, French, Italian) provided in JSON files with fields for content, source, title, and language.
- **Preprocessing Steps**: HTML cleaning, deduplication, and chunking were applied to prepare the data for embedding.

4 Vector Store Generation

To support efficient retrieval, we construct a vector store using Chroma, where document embeddings are stored. The vector store generation process involves:

- **Document Loading and Chunking**: Using a custom loader (`CustomVoyageAILoader`), documents are loaded, preprocessed, and chunked into manageable parts.
- **Embedding Creation**: Each document chunk is embedded using the `VoyageAIEmbeddings` model (`voyage-3`) with a batch size of 32.
- **Vector Store Construction**: The Chroma vector store is built from these embeddings, storing them in a persistent directory for efficient similarity-based retrieval.

5 Retrieval Process

The retrieval process in our RAG system combines an initial similarity search with a re-ranking step to improve relevance.

5.1 Similarity Search

We begin by querying the Chroma vector store, where each document chunk is embedded in vector space. Using an embedding function, the system retrieves the top 50 document chunks similar to the query. This initial search creates a pool of potentially relevant documents.

5.2 Re-Ranking

The top 50 documents are then re-ranked using the `voyageai/rerank-2` model to process query-document pairs together and refine relevance. Document contents are passed as strings to the model, which scores each chunk based on query relevance. We return only the top `k`(=10) re-ranked results for further processing.

6 Generation Module

The generation module is responsible for producing responses by leveraging both the query and relevant context retrieved from the vector store. When a user query is entered, the RAG system workflow is as follows:

- **Retrieval and Contextualization**: The query is first processed through the `Retriever` module, which defines relevant document chunks retrieved from the vector store as context of the query.
- **Prompt Construction and LLM Invocation**: Using the retrieved chunks and conversation history, the `LLM_request` class constructs a prompt for the GPT-4o model. This prompt integrates the user's query, retrieved context, and previous exchanges to ensure continuity and factual relevance in responses.
- **History Management**: The system maintains a conversation history to provide consistent responses across multiple user queries, adding each interaction to memory for context continuity.

This generation approach ensures that the chatbot can provide precise, contextually relevant responses that are grounded in retrieved information, enhancing both relevance and user trust.

7 Conclusion

- **Impact**: The RAG chatbot prototype provides a foundation for AI-powered customer support, with potential for further integration within Swisscom's support channels. Additionally, it could offer an efficient way to gather representative data on customer challenges, providing insights that can drive targeted improvements in services and support.
- **Future Work**: Future work includes enhancing multilingual capabilities, optimizing the retrieval model, incorporating advanced query handling, and adding speech support. Further improvements could focus on better memory management, gathering user feedback to guide enhancements, and refining the system based on user interactions.

Main References

- `https://python.langchain.com/docs/introduction/`
- `https://docs.voyageai.com/docs/introduction`
- `https://blog.voyageai.com/2024/09/30/rerank-2/`
- `https://docs.trychroma.com`
- `https://openai.com/index/hello-gpt-4o/`