Q&A AI Chatbot Design

Goal

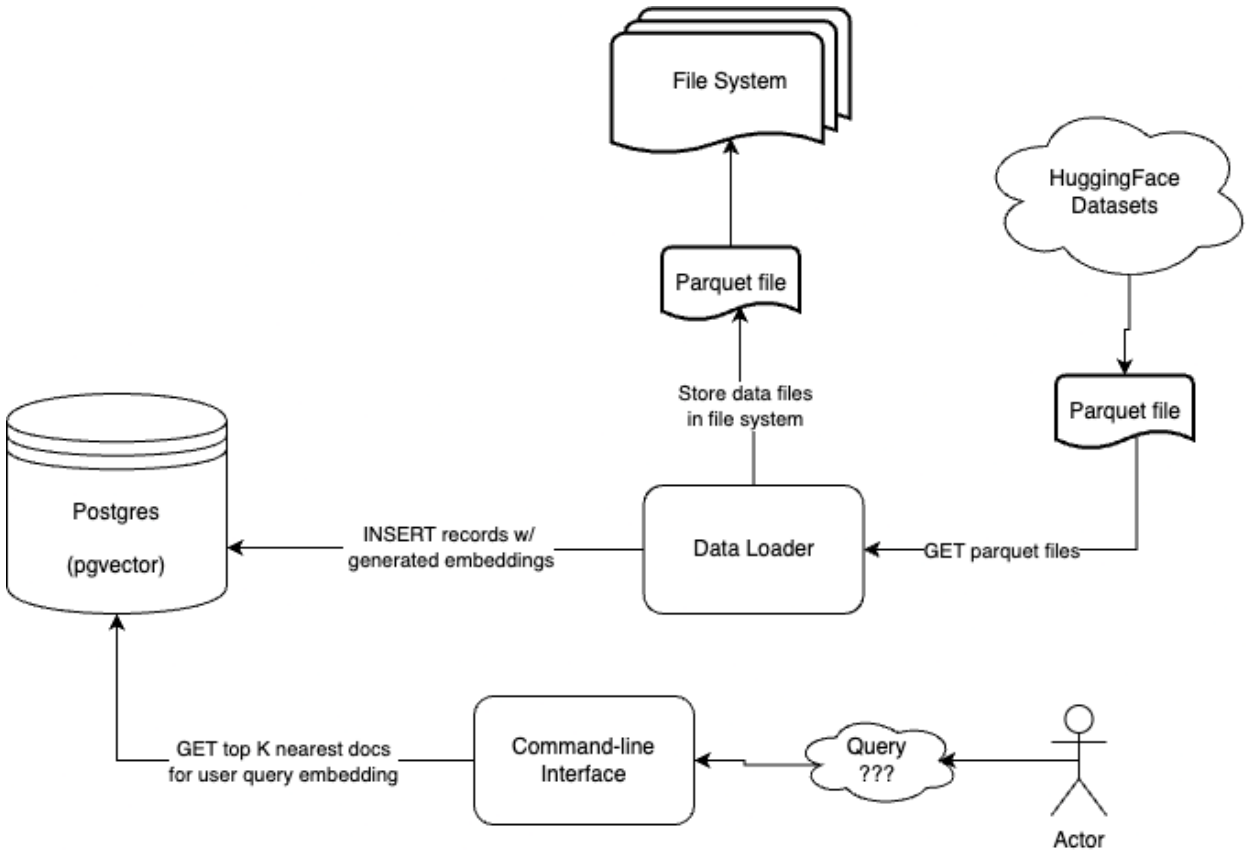
Build an easy-to-use chatbot to answer questions about a dataset.

Design

To achieve the goal without having to train a model, retrieval augmented generation (RAG) is leveraged here. With RAG, the system can accept an input question, retrieve the most relevant documents for a data store, and add that as context to the original input for a pretrained model to use in generating a more reliable response.

The system implementation consists of three main components: a database, a data loader, and the actual CLI program.

Architecture



Database

This system uses a Postgres database with the pgvector extension which provides a new "vector" data type to support embeddings store and search. Some of the advantages for this choice of data server include the following:

- Postgres has been around for a long time, is highly robust, and is extensively supported as an open source project.
- It still comes with all of the original storage capabilities. Metadata and other items can be easily stored alongside the vector data.
- Teams can stick with one data server (rather than maintaining a separate data server just for vector data).
- Postgres provides a consistent query language that is familiar.

An HNSW index is created on the embedding column in the table to make vector queries much faster for a large dataset.

Data Loader

The Data Loader performs all necessary steps to retrieve the dataset, process and prepare (chunk) the data for storage, generate vector embeddings for the data chunks, and store the embeddings in the database.

The initial challenge here was to find a dataset that can be easily read and manipulated. So HuggingFace was chosen as the source in order to make use of their Wikipedia dataset, which is structured in parquet files. The Data Loader downloads these parquet files and loads them one-by-one into a Pandas dataframe for convenient grid-level access.

Before embeddings are generated on the text data, the text needs to be chunked by a certain number of tokens (in this case, individual words are used as tokens). Chunks with length of 512 tokens is a common standard and limit for several transformer models (e.g. BERT). The text is chunked with an overlap to ensure sentences split in one chunk are still fully preserved in another.

After processing the full text data of a dataframe row, embeddings are generated with the HuggingFace SentenceTransformer library for each chunk (document), and the batch is inserted into the database.

Feature Extraction Model

The pretrained model "sentence-transformers/all-MiniLM-L6-v2" is used to generate the vector embeddings. Why use this instead of OpenAI's "text-embedding-ada-002" model? It produces embeddings with 384 dimensions as compared to the OpenAI model 1536 dimensions. This makes for much faster queries and less memory usage. And it is also worth noting that, number of dimensions is not correlated to performance. The sentence-transformers model has a very small footprint and still scores quite well according to the MTEB Leaderboard.

Command-line Interface

The command-line interface itself accepts an input question from the user and generates the embeddings for that question with the same “sentence-transformers/all-MiniLM-L6-v2” model. To come up with an answer to the question, the program makes a SELECT query to the database, leveraging the pgvector extension K nearest neighbor operator (<=>) to retrieve the top related documents (chunks of text). This serves as context for the question. Finally, the HuggingFace “transformers” library is used to perform the actual question answering based on the provided input question and retrieved context.

Question Answering Model

The pretrained model “distilbert-base-cased-distilled-squad” is used for question answering. This model was trained on the SQUAD dataset, which is a large corpus of questions and their corresponding answers. It is a smaller and (up to 60%) faster version of the BERT model that still maintains strong performance (95% of BERT’s).