Technical Documentation: QueryQuill

Table of Contents

1. Overview

The system is designed to provide efficient and accurate passage retrieval based on user queries. It utilizes state-of-the-art embedding techniques to represent questions and passages in vector space, and Elasticsearch to perform efficient vector space search.

2. Flask API

Design Choices:

- Flask was chosen because of its simplicity and effectiveness in building web APIs.
- The API contains endpoints for asking questions, uploading data, and resetting the Elasticsearch index.

3. SentenceTransformers & Embeddings

Design Choices:

- SentenceTransformers was chosen for its ability to produce high-quality embeddings for sentences and paragraphs.
- It uses a variant of the BERT model fine-tuned for sentence embeddings, which offers state-of-the-art performance on various NLP tasks.
- These embeddings allow for semantic search, making it possible to retrieve passages that are semantically similar to the user's query.

4. Elasticsearch

Design Choices:

- Elasticsearch, a distributed search engine, was employed to efficiently store and retrieve passages.
- It provides support for dense vector fields, allowing us to store embeddings and perform cosine similarity searches on them.
- Its distributed nature ensures scalability and high availability.

5. Data Parsing and Indexing

Design Choices:

- Given the potential for diverse data formats, a modular approach was taken to parse documents and extract meaningful passages.
- The system expects data in pairs of **.txt** (for passages) and **.json** (for metadata) files. This choice simplifies data ingestion and allows users to provide rich metadata for each passage.
- Indexing is handled in batches using Elasticsearch's bulk API for efficiency.

6. Generative AI

Design Choices:

1. **Choice of Generative Model - Llama2**:
    - The Llama2 model was selected for its proficiency in generating concise and semantically rich responses. This model has proven to be effective in understanding context and generating direct answers based on input passages, making it a suitable choice for our use case.
    - Direct answers are crucial in situations where users might not have the time or patience to go through passages to extract information. By providing direct, concise answers, we enhance the user experience and increase the utility of our system.
2. **Choice of Embedding Model - Paraphrase-Distilroberta-Base-V1**:
    - The **paraphrase-distilroberta-base-v1** model was specifically selected from the SentenceTransformers library because of its expertise in generating embeddings that capture the semantic meaning of the text. Trained primarily for paraphrasing tasks, this model ensures that embeddings retain the context and nuance of the original text, which is paramount for accurate information retrieval.
    - This model offers an optimal balance between computational speed, accuracy, and size. Its moderate size ensures that embeddings are computed efficiently without

compromising on performance. This is especially critical in a real-time search environment where response time is crucial.

- Given the vast array of embedding models available, it's essential to make a choice that aligns with the application's requirements. The **paraphrase-distilroberta-base-v1** model emerged as a top contender due to its specialized training and proven track record in paraphrase tasks.

7. Docker Integration

Design Choices:

- Docker was chosen for its ability to provide a consistent and reproducible environment.
- It ensures that the application runs the same regardless of where Docker is run, eliminating the "it works on my machine" problem.
- The provided Dockerfile ensures all necessary dependencies are installed and sets up the environment for the application.