Question-Answering-System - Technical PDF Documentation

Intriduction

The main objective of this project was to build question-answering (QA) machine learning system that can take a queries or answers from users, search for relevant passages in a given corpus, and return them as answers.

Task 1: Parsing

Design Choice: Implemented the necessary libraries for text processing to extract and chunk passages.

Here, I extracted the passages within child paragraph markers under parent section markers. Combined all the extracted passages into one, then split them into chunks of 5-sentences to form a passage each. The focus is on efficiency and accuracy in text extraction to ensure high-quality data for indexing.

Task 2: Passage Embeddings Generation

Design Choice: Selected SentenceTransformers and specifically the ‘paraphrase-MiniLM-L6-v2’ model because of its quality and efficiency in generating embeddings. Aimed for embeddings that capture semantic meanings to improve the relevance of retrieved passages.

Task 3: ElasticSearch Integration

Design Choice: I chose elasticSearch due to its scalability, full-text search capability, and vector search features.

Loaded the passages and metadata from the passage_metadata csv file and utilised the SentenceTransformers library to generate embeddings for each passage. The focus was on creating a custom index structure to optimize the storage and retrieval of vector data.

Task 4: Document Retrieval

Design Choice: Implemented a custom retrieval mechanism leveraging ElasticSearch’s capabilities.

Set up an ElasticSearch index to store the preprocessed and embedded data. Indexed each row of the passage, metadata, and embedding data from the "passage_metadata_emb.csv" file into ElasticSearch.

The goal was to ensured the retrieval is efficient and provides highly relevant results with scored relevance. I also calculated the cosine similarity between the question's embedding and the indexed passage embeddings to rank the results.

Task 5: Containerization

Design Choice: Docker specifically because it ensures that the application is isolated and has all dependencies to run.

Which is the reason why I containerize the application using Docker. I created a Dockerfile to include the application and all its dependencies. Secondly, I also use docker-compose to manage the application and Elasticsearch service's containers, ensuring they can interact smoothly. This techniques of containerization aids in deploying the application seamlessly in various environments.

Task 6: API Deployment with Flask
Design Choice: Flask because of its simplicity and flexibility, and it's used to expose endpoints for querying and uploading documents.

Implemented an endpoint to receive user questions and return relevant passages from the Elasticsearch index. Essentially, to upload documents, process them, and index them in Elasticsearch. We will also implement logging, request/response validation, and handle concurrent requests. The idea of using features like logging and request validation is to enhance security and usability.

Task 7: Evaluation
Design Choice: Implemented manual rating for relevance and calculated top-1 and top-3 accuracies.

Here, I utilised the Flask API I developed earlier to get the top 3 relevant passages for each query in the user_queries.txt file. This approach is essential because it provides insights into the system's effectiveness and areas for improvement.

Task 8: Direct Answers with Generative AI
Design Choice: Integrated a generative LLM to provide concise, direct answers, enhancing user experience. Here, I used generative language model (gpt2) to provide direct answers to the questions based on the retrieved passages. The model is also well implemented and tuned to generate near-deterministic responses for reproducibility.

Task 9: FrontEnd with Streamlit
Design Choice: I went ahead with streamlit because it offers rapid development of interactive web apps with minimal coding.

Created a basic frontend using Streamlit to allow users to interact with question-the system. It will provide options for uploading documents for indexing and entering questions to retrieve relevant answers. This design choice focuses extensively on user experience, making the QA system accessible and easy to use.

Conclusion
Each task's design and implementation considered factors like scalability, efficiency, and user experience. The evaluation mechanism ensures continuous improvement, and the generative AI integration aims at enhancing the user experience by providing direct and concise answers. Every technique and approach chosen aligns with the goal of delivering a robust, efficient, and user-friendly QA system.