Generative AI - Assignment 4
Technical Report

Muhammad Huzaifa
20i-2473
CS-A

May 11, 2024

1 Introduction

In recent years, the intersection of natural language processing (NLP) and information retrieval (IR) has witnessed remarkable advancements, leading to the development of innovative systems capable of understanding and generating human-like responses to queries. One such advancement is the Retrieval-Augmented Generation (RAG) paradigm, which combines the strengths of both retrieval-based and generation-based approaches to enhance the quality and relevance of generated responses.

The primary goal of RAG systems is to generate coherent and contextually relevant responses to user queries by leveraging both pre-existing knowledge (retrieved from a database) and generative models (capable of producing human-like text). This paradigm has found wide applications in various domains, including question answering, dialogue systems, and content creation.

In this technical report, we present the design and implementation of an RAG system tailored for assisting users in accessing and understanding academic literature, specifically focusing on Master of Science (MS) theses. Our system utilizes a dataset containing titles and abstracts of MS theses from FAST-NUCES University, Pakistan, and leverages advanced NLP techniques to provide users with informative and relevant responses.

2 Methodology

Data Collection and Preprocessing

The initial step in the methodology involved the collection and preprocessing of the dataset containing titles and abstracts of MS theses. The dataset was obtained from FAST-NUCES University, and it consisted of structured data stored in an Excel file format. The `pandas` library in Python was utilized to read the Excel file and extract relevant information from each sheet. Additionally,

data cleaning and formatting were performed to ensure consistency and usability of the extracted data. Specifically, each sheet in the Excel file was iterated through, and the title and abstract of each thesis were extracted and stored in a list of dictionaries.

Vectorization and Database Management

Once the data was collected and preprocessed, the next step involved vectorization and database management tasks. The `sentence-transformers` library in Python was used to generate vector representations of the textual data. Specifically, the `all-MiniLM-L6-v2` model, a pre-trained sentence embedding model, was employed to convert text into fixed-length vectors. These vectors encoded semantic information about the input text and served as representations in the vector space.

Furthermore, QDrant, an efficient vector database designed for similarity search, was utilized to manage and store the vector representations of the thesis abstracts. An in-memory QDrant instance was instantiated using the provided client library (`qdrant-client`), and a collection named `thesis-info` was created. The collection was configured to store vectors of the same dimensionality as the output of the sentence embedding model, with cosine distance as the similarity metric.

Upload Records

Following vectorization and database setup, the vector representations of the thesis abstracts were uploaded to the QDrant collection. This process involved iterating through the preprocessed data and uploading each abstract vector along with its corresponding metadata (title and abstract text) as a record in the QDrant collection. The `qdrant-client` library was utilized to interface with the QDrant instance and perform record uploads efficiently.

Query Processing and Retrieval

The next step in the methodology was query processing and retrieval, where mechanisms were designed to process user queries and retrieve relevant documents from the QDrant collection. Upon receiving a user query, the sentence embedding model was utilized to convert the query text into a vector representation. Subsequently, a similarity search was performed in the QDrant collection using the query vector to identify documents with abstracts similar to the user query. Cosine similarity was employed as the similarity metric to measure the similarity between query and document vectors.

Response Generation

Once relevant documents were retrieved, the methodology proceeded with response generation using a pre-trained language model. The Hugging Face

`pipeline` for text generation was employed, specifically utilizing the `text-generation` pipeline with the `gpt2` model architecture. Given the user query and the retrieved documents, a prompt containing relevant information was constructed and passed to the text generation pipeline. The pipeline generated informative and contextually relevant responses based on the input prompt, incorporating information from both the user query and the retrieved documents.

Web Interface

To facilitate user interaction and real-time response generation, the RAG system was deployed as a web application using the Flask framework in Python. Flask provided a lightweight and scalable platform for building the web interface, allowing users to input their queries through a user-friendly interface and receive generated responses in real-time. A simple HTML template was designed to serve as the user interface, with a form for inputting queries and an area for displaying generated responses. The Flask application handled incoming requests, processed user queries using the implemented methodology, and returned generated responses to the user interface for display.

Testing and Evaluation

Throughout the development process, rigorous testing and evaluation were conducted to ensure the functionality and performance of the RAG system. Both manual testing and automated testing techniques were employed to validate the correctness and robustness of each component. Additionally, the system's performance was evaluated in terms of response quality, response time, and scalability under various scenarios and workloads. The evaluation results provided valuable insights into the effectiveness and efficiency of the RAG system, guiding further optimizations and improvements.

3 Results

The implementation of the Retrieval-Augmented Generation (RAG) system yielded promising results in terms of functionality and performance. In this section, we present an overview of the key findings and outcomes obtained during the testing and evaluation phase of the project.

Functionality Evaluation

The functionality of the RAG system was extensively tested to ensure that all components performed as intended and that the system operated smoothly under various scenarios. The following aspects of functionality were evaluated:

- The data collection and preprocessing pipeline successfully extracted titles and abstracts from the provided dataset, ensuring that the input data was clean and formatted appropriately for further processing.

- The vectorization process effectively transformed textual data into high-dimensional vectors, which were then stored and managed in the QDrant database. The vector database facilitated efficient similarity search operations during query processing.
- The query processing mechanism accurately converted user queries into vector representations and retrieved relevant documents from the QDrant collection based on cosine similarity. The system demonstrated the ability to handle diverse user queries and return meaningful results.
- The response generation component generated informative and contextually relevant responses based on the user query and the retrieved documents. The generated responses exhibited coherence and relevance, providing users with valuable insights.
- The web interface provided a user-friendly platform for interacting with the RAG system. Users could input queries through the interface and receive generated responses in real-time. The interface design was intuitive and responsive, enhancing the overall user experience.

4 Key Findings

Based on the testing and evaluation results, the RAG system demonstrated robust functionality and satisfactory performance across all tested metrics. Key findings from the evaluation process include:

- The system effectively retrieved relevant documents and generated informative responses, showcasing its potential for assisting users in accessing and understanding academic literature.
- Response quality remained consistently high across different user queries and dataset variations, indicating the reliability of the system in providing accurate and relevant information.

5 Conclusion

In conclusion, the development and evaluation of the Retrieval-Augmented Generation (RAG) system have demonstrated its effectiveness in assisting users in accessing and understanding academic literature, particularly Master of Science (MS) theses from FAST-NUCES University, Pakistan. The implementation of the RAG system successfully integrated advanced natural language processing (NLP) techniques with efficient information retrieval (IR) mechanisms to provide users with contextually relevant and informative responses to their queries.

This may include fine-tuning the response generation model, enhancing the web interface for better usability, and expanding the dataset coverage to encompass a wider range of academic literature sources.