End-to-End Retrieval-Augmented Generation (RAG) System

DSAN 5800 Project Report

Yuting Fan Peng Li Yiwei Qi

2024-12-10

Summary

The project developed an end-to-end retrieval Enhanced generation (RAG) system that combines FAISS-based document retrieval and OpenAI for accurate context-aware question-and-answer. The system supports dynamic knowledge base extension and allows users to enter OpenAI API keys and select models through the Streamlit interface for increased flexibility and personalization. The evaluation results show that the retrieval efficiency is high, the precision is high, and the response is accurate and smooth, which highlights the potential of the system in education, research and other fields.

1. Introduction

RAG (Retrieval-Augmented Generation) makes up for the deficiency of the traditional generation model in specific domain knowledge by introducing knowledge retrieval module. Retrieval modules (such as FAISS) quickly locate knowledge documents relevant to user queries, while generation modules (such as the GPT family of language models) generate context-relevant and coherent responses based on these documents. This architecture effectively avoids the problem of generating model illusions due to insufficient data or incomplete training, making the system more reliable and controllable.

In practice, the main technical challenge facing retrieval modules is how to quickly and efficiently find the most relevant documents from the massive data. For example, FAISS uses vector index to implement approximate nearest neighbor (ANN) search, which is more efficient than traditional keyword matching, but requires periodic update of index structure when dealing with dynamic knowledge base, which may bring additional computational overhead. The challenge of generating modules is how to get the language model to generate accurate and logically coherent answers based on context, especially when the retrieved document contains redundant or contradictory information.

The core goal of this project is to build a scalable end-to-end RAG system that automates the whole process from user input to answer generation, including dynamic management of knowledge base and parameter optimization of generation module. By implementing the function of document upload, the system allows users to dynamically expand the knowledge base and make it adapt to the needs of different fields, so as to solve the problems of real-time embedding generation, index update and retrieval performance optimization of large-scale data. On the basis of integrating the search results and user queries, the system also optimizes the generation module (GPT model) to improve the context adaptability of the generated content.

To enhance user trust in the content generated by the system, the source information is embedded in each response. This is done by the retrieval module recording the source of the relevant document and presenting it in the generated answers. In addition, to further enhance the user experience, we optimized the front-end interface and developed it using the Streamlit framework. Streamlit's lightweight and fast development features enable users to interact efficiently with the system.

It is worth mentioning that the user interface has two important new features: users can enter their own OpenAI API keys to ensure seamless connection between the system and their personal accounts; Users can also choose their own generation models (such as GPT-3.5 or other models) according to their needs to meet the performance and generation requirements of different scenarios. These improvements further enhance the flexibility and personalized service capabilities of the system, so that it has a wider application potential in education, science and technology.

2. System Architecture

2.1 System composition

The core of the project is an end-to-end retrieval enhanced generation (RAG) system, which consists of three main components: document retrieval module, language generation module and user interface. Through modular design, we realize the complete data flow from user query to generated response, ensuring the efficiency and user experience of the system.

First, the document retrieval module is implemented based on FAISS (Facebook AI Similarity Search), which is used to quickly retrieve documents related to user queries from the knowledge base. Documents are preprocessed and vectorized before entering the knowledge base. We generate high-dimensional vector embeddings of documents using Hugging Face's SentenceTransformers model, and build dense indexes through FAISS to support efficient retrieval. User queries also generate query embeddings through the same vectorization process, and then use FAISS 'IndexFlatL2 method to match the most relevant documents based on Euclidean distances. This method not only guarantees the retrieval speed, but also improves the

accuracy of the returned documents, and provides a reliable input for the subsequent language generation.

On the basis of document retrieval, the system integrates language generation module and uses OpenAI GPT model to generate natural language answers. The core logic of language generation module is to concatenate the user's query and the retrieved document fragments into contextual input, so that the generation model can generate high-quality answers based on facts. By controlling the generation parameters, we further optimize the consistency and accuracy of the generated content. In addition, the module supports the return of answers containing reference documents, which enhances the interpretability and credibility of the system.

Finally, the user interface is implemented through the Streamlit framework, providing users with a simple and intuitive operating experience. The interface design includes real-time chat box, document upload and response display. Users can enter questions in real time and receive model-generated answers, while the upload document feature allows users to dynamically update the knowledge base, and the uploaded content is generated in real time and embedded and added to the retrieval index. In addition, users can enter their own OpenAI API key to ensure that the system can be seamlessly connected to their personal accounts; At the same time, users can also choose the generation model (such as GPT-3.5 or other versions) to meet the generation requirements of different scenarios. These improvements greatly enhance the system's flexibility and personalized service capabilities. In terms of response display, the system is accompanied by the source information of the referenced document, so that users can clearly understand the basis of the generated content, and further enhance the transparency and trust of the system.

2.2 Data Flow

The overall data flow of the system starts from the user query, returns the relevant document through the retriver, generates the response through the language model, and finally displays it in the user interface. This smooth design enables a seamless connection between the document and the generated model, while ensuring efficient retrieval and accurate answers. The modular architecture of the system is not only easy to expand, but also provides flexible adjustment space for future optimization. With this structured design, we have achieved an efficient, transparent and user-friendly RAG system.

3. Implementation Details

3.1 Retriever

In this project, we chose FAISS as the core tool for document retrieval, while using Hugging Face's SentenceTransformers model to generate vector embedding of documents and queries.

Specifically, we use a pre-trained model based on Transformer architecture, all-MiniLM-L6-v2, to transform text into a fixed-length vector representation, thereby effectively preserving semantic information to improve retrieval accuracy. In terms of Index construction, we mainly used a Flat Index structure, while considering the feasibility of using hierarchical K-Means indexes (such as HNSW or IVF) in large-scale knowledge base scenarios to optimize retrieval efficiency. In the similarity calculation, we further improve the consistency of the calculation by normalization.

In the actual retrieval process, the query entered by the user is first transformed into a vector embedding, and then k-Nearest Neighbors is searched through the FAISS index. The initial search results are sorted by cosine similarity to ensure that the returned documents have a high degree of relevance. In addition, we introduced the rerank step in the retrieval process, and sorted the preliminary search results by the CrossEncoder model (cross-encoder/ms-marco-MiniLM-L-12-v2). In the reordering stage, semantic matching scores of query and candidate document pairs are used to further optimize the search results, which significantly improves the quality of the final sorting results.

3.2 Generator

The generation module allows users to choose different language models based on their preferences. In the implementation process, we support users to choose to use the gpt-3.5-turbo or gpt-4-o model provided by OpenAI to generate high-quality query responses.

The input splicing strategy is adopted in the generation process to integrate the document fragments queried and retrieved by users into the model input, and attach multiple related documents to provide sufficient context information. This design helps the model fully understand the user's needs and generate accurate and coherent responses in conjunction with relevant content. To further improve the quality of generation, we have optimized the model parameters, such as setting the temperature to 0.7 to balance the randomness and stability of generation, while adjusting the max_tokens parameter to limit the output length and ensure the thematic relevance and simplicity of the generated content. In addition, through the Streamlit user interface, users can dynamically select the language model to meet the personalized needs of different scenarios. This flexible generation strategy significantly improves the system's ability to respond to complex queries, while ensuring contextual consistency and semantic accuracy of the generated content, providing users with a higher quality interactive experience.

3.3 Web Application

The Web application uses the Streamlit framework and provides real-time interaction, including a chat interface, document upload, and traceability information display. The chat interface supports the user to enter natural language queries, the query content will be passed to the

retrieval module, the accurate answer will be generated through the retrieved relevant documents, and the response results will be displayed to the user in real time through st.write(). The user interface allows users to enter the OpenAI API key by themselves, ensuring that the system can seamlessly connect with the user's OpenAI account. In addition, the system also provides model selection function, users can choose to use different generation models according to the needs (such as GPT-3.5 or other supported models) to meet the diversified generation needs. The Document Upload feature allows users to upload custom documents via drag and drop or file selection, which, after pre-processing (such as text cleaning and embed generation), are dynamically updated to the knowledge base and immediately indexed to the FAISS database, ensuring that new content can be used in real time for subsequent queries.

In addition, while generating answers, the system also provides the retrieved document source information, including the document title and related fragments, which users can click to view the original content, thus enhancing the credibility and transparency of the answer. Through the optimization of these functions, the system further improves the flexibility and interactivity of user experience, while ensuring the accuracy and interpretability of the results.

4. Evaluation

To verify the stability and compatibility of the end-to-end Search Enhanced Generation (RAG) system we built, we conducted comprehensive tests on file support, file size, file format, and query validity. These tests are designed to verify the compatibility of the system with different file types to ensure the wide applicability of the document upload function. At the same time, test and analyze the performance and efficiency of the system when processing files of different sizes, especially the ability to support large files, and evaluate the robustness of the system when processing invalid file contents and invalid queries, so as to ensure that the system can run stably and provide clear feedback when the user enters errors or irregularities.

4.1 File Type Test

We tested the mainstream file types to check whether they are supported by the project (e.g. PDF, PNG, HTML, TSV, JPG, etc.) to verify file upload capabilities and system compatibility. Specifically, we simulate the user, using the front-end page as the entry point, using the 'upload_document' method in the project code 'rag_app_new.py' to handle the file upload, and calling the FAISS retrieval through the backend to generate the embed. The SentenceTransformers model is used to parse the text content. Among the supported file types, PDF (such as 'lecture-07a-notes.pdf') can successfully extract text and generate embeddings, and the preprocessing module effectively removes redundant characters from complex typesetting. However, for binary files (such as '.dmg' and '.pkg'), TSV files (such as 'metadata.tsv'), Jupyter Notebook files (such as 'word2vec-demo.ipynb'), the system cannot process, This is because such files do not contain parsable natural language text, and there is no dedicated

processing logic for binaries implemented in the project code. This shows that the system has good support for text files, but there are limitations in binary file processing.

4.2 File Size Test

In the file size test, we performed performance analysis for files of different sizes. For small files (<1KB), the system can quickly generate the embed and complete the retrieval task with almost negligible response delay. However, for large files (>100MB), we use the batch logic in *rag_app_new.py* to process the document in segments, with each segment generating an independent embed. Although this method is effective, tests show that the query speed of the FAISS index decreases significantly when the file size approaches the memory limit. For very large files (>1GB, such as *rating.dta*), the system ran out of memory during the file preprocessing phase and failed to complete the task. This indicates the need to introduce stream processing mechanism to optimize the parsing ability of large files, so as to improve the scalability and stability of the system.

4.3 Invalid File Content Test

In the invalid file content test, we mainly verified the system's ability to handle special cases, including empty files and garbled files. For the ** empty file (Doc.docx) **,** the system can accurately recognize that its content is empty, and through the code logic 'if not content:' returns a clear message: "The file content is empty, please upload a file containing valid text." This processing method shows the reliability of the system in input verification. For thegarbled file (well.docx) **, although the system tries to parse the text content through the preprocessing module, it cannot generate meaningful embeddings due to garbled text, but the generated results illustrate this point to help the user understand. This shows that the system has the ability of content verification and exception handling when dealing with non-standard text.

4.4 Query Validity Test

In the query validity test, we simulate a variety of situations to evaluate the robustness of the system, including invalid queries (such as random characters' skjfnwejfdkj ') and empty queries. In the invalid query scenario, the system can successfully generate query embedding, but due to the lack of relevant document matching, the system returns the prompt "No matching content found", showing a certain stability. In empty query scenarios, the system automatically blocks query execution through logical verification and prompts users to enter valid questions in time. These mechanisms guarantee the basic stability of the system under abnormal input conditions.

4.5 Generating Quality Test

In order to comprehensively evaluate the performance of the system generation module, we conducted a manual evaluation of the generated content of different language models from the two dimensions of accuracy and relevance, mainly comparing the performance of GPT-3.5-Turbo and GPT-4.0. The specific generated content is shown in Figure 1 and Figure 2.

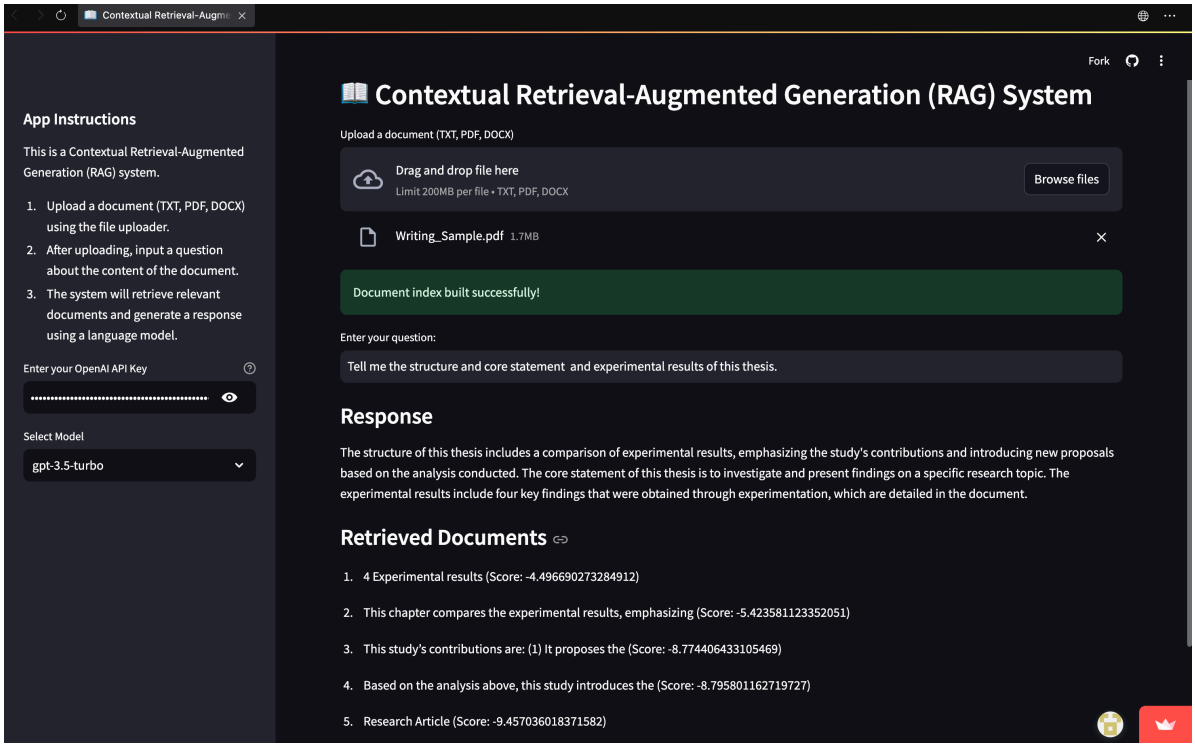

Figure 1: Figure 1: GPT-3.5-turbo

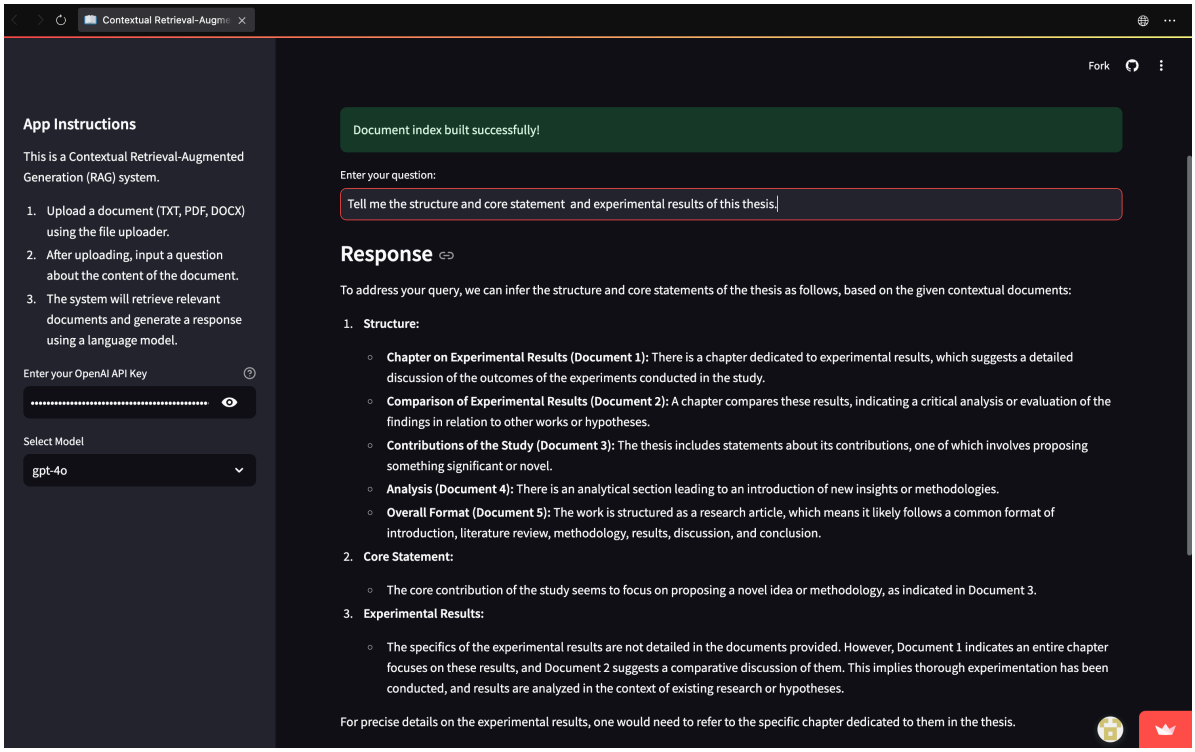

Figure 2: Figure 2: GPT-4o

Accuracy is a key indicator to measure whether the generated content is consistent with the retrieved document information and can accurately answer user questions. In the evaluation, GPT-4.0 showed higher accuracy. Taking the paper structure and the task of generating experimental results as an example, GPT-4.0 can clearly list the core chapters of the paper (such as the chapter of experimental results, the chapter of research contribution and the chapter of analysis), and accurately extract the key information in the retrieved documents for summary. The model has a more comprehensive understanding of the user's problem, and the generated content fully conforms to the original information of the document. In contrast, the GPT-3.5-Turbo, while also providing more accurate answers, is slightly less effective at integrating documentation information. For example, when analyzing the experimental results, GPT-3.5-Turbo's answers were more general and lacked in-depth summaries of the details of the comparative experiments.

In terms of semantic relevance, the generated content of GPT-4.0 is more logical and targeted. For complex questions asked by users, such as the analysis of "paper structure and experimental results", GPT-4.0 can fully integrate the retrieval of document content to generate a closely related and coherent answer, showing its deep understanding of the context of the question. In contrast, the GPT-3.5-Turbo generated content is slightly less relevant, although it can cover the core points of user questions, but the organization and detail of the answers are slightly

scattered. For example, the generated content sometimes ignores important information points mentioned in the retrieved documents and fails to provide specific and comprehensive answers in full context.

5. Conclusion

In this project, we have successfully implemented an end-to-end Retrieval Augmented Generation (RAG) system. The system combines efficient dense retrieval technology (based on FAISS implementation) with powerful natural language generation model (GPT) to provide accurate answers based on knowledge base while responding quickly to user queries. By implementing the functions of document uploading and source tracing, the system not only has good scalability, but also ensures the interpretability of the generated content.

Users can enter their own OpenAI API keys through the interface in order to connect to their personal accounts, ensuring the convenience and security of generating services. In addition, users can also choose the language model (such as GPT-3.5 or other models) to meet the performance requirements and generation effects in different scenarios. The introduction of these functions makes the system better adapt to the diversified user needs.

Although our RAG system has achieved the main functionality, there is still room for improvement to be explored in the future. First of all, the current system only supports the retrieval and generation of text knowledge base, the future can be extended to support multimodal data, such as images, audio or video. This extension will enable the system to handle more complex queries, such as extracting content with images from documents or generating summaries based on video and audio.

Secondly, the current system uses a pre-trained GPT model, although its generation ability is strong, but the depth of knowledge in a specific field is still insufficient. In the future, the GPT model can be fine-tuned with domain data to improve the accuracy and relevance of the generated results. In the medical field, for example, adding fine-tuning to PubMed datasets can make models more professional in answering medical-related questions.

With the above improvements, the RAG system can be further expanded to a wider range of application scenarios, such as intelligent question answering, academic assistants, and cross-modal knowledge management platforms. This will not only further enhance the academic value of the system, but also provide more powerful technical support for practical applications.

6. Appendix

- Code Repository: https://github.com/pengleee/Project-5800-RAG-2024Fall
- Knowledge Base Sources: Custom Uploads.
- Evaluation Dataset: User Queries.