# Q/A Chatbot

21st August 2024


Ahmad Meda

Final Year Student at Heriot-Watt University

BSc Computer Science

# Table of Contents

# RAG Model

## Introduction

This report covers the development of a Retrieval-Augmented Generation (RAG) model aimed at enhancing document-based query answering. The system integrates several components, including embedding models, retrievers, and a query engine. This architecture allows for document retrieval through both semantic and keyword-based search while generating informative, accurate responses using a language model (LLM). This document provides an in-depth explanation of the model architecture, retrieval approach, and the mechanism for generating responses.

# Model Architecture

After retrieving the relevant documents, the **RetrieverQueryEngine** uses the **Groq LLM** to generate responses based on the content of the retrieved documents. The process works as follows:

## Document Retrieval

The hybrid retriever fetches a set of relevant documents (or nodes) based on the user's query.

## Response Synthesis

The LLM is provided with the retrieved documents, and it generates a coherent, human-like response. This generation is guided by the context provided in the documents, ensuring that the response is both accurate and contextually appropriate.

## Evaluation

The retrieval results and the LLM responses are evaluated using metrics like precision, recall, and hit rate. These metrics ensure that the model retrieves the most relevant documents and generates accurate answers.

# Retrieval Approach

The hybrid retriever uses both BM25 and semantic retrieval to fetch relevant results from the document corpus. The hybrid retriever performs a dual search: one using BM25, which is keyword-based, and the other using vector embeddings to capture the semantic meaning of the query. The two sets of results are merged, and a final list of nodes is created to ensure coverage of both keyword relevance and semantic similarity.

# Generative Responses

Once the relevant nodes have been retrieved, the language model synthesizes a response. The LLM uses the provided context (retrieved nodes) to generate an answer that is tailored to the specific question asked. This approach ensures that the response is not only relevant but also backed by evidence extracted from the document corpus.

# Code Breakdown

## Embeddings and Settings

```
Settings.embed_model = HuggingFaceEmbedding(model_name="BAAI/bge-small-en-v1.5")
Settings.llm = Groq(model="llama3-8b-8192", api_key="your_groq_api_key")
```

We initialize the embedding model and language model to create document vectors and generate responses.

## Document Parsing and Splitting

```
text_splitter = SentenceSplitter(chunk_size=1024)
documents = SimpleDirectoryReader("/path/to/documents").load_data()
nodes = text_splitter.get_nodes_from_documents(documents)
```

Documents are loaded and split into manageable chunks for indexing and retrieval.

## Vector Index and Hybrid Retriever

```
vector_index = VectorStoreIndex(nodes)
semantic_retriever = vector_index.as_retriever(similarity_top_k=5)
bm25_retriever = BM25Retriever(nodes=nodes, similarity_top_k=5)
retriever = HybridRetriever(semantic_retriever, bm25_retriever)
```

The vector index is built, and the hybrid retriever is created by combining both semantic and keyword retrieval methods.

## Evaluation

```
qa_dataset = generate_question_context_pairs(nodes, llm=Settings.llm,
num_questions_per_chunk=1)
metrics = ["hit_rate", "mrr", "precision", "recall", "ap", "ndcg"]
retriever_evaluator = RetrieverEvaluator.from_metric_names(metrics, retriever=retriever)
eval_results = retriever_evaluator.aevaluate_dataset(qa_dataset)
```

The retrieval mechanism is evaluated using standard metrics such as precision, recall, and hit rate.

## Query Execution and Response Generation

```
query_engine = RetrieverQueryEngine(retriever=retriever,
response_synthesizer=get_response_synthesizer())
response = query_engine.query("Tell me the balance for 2023")
print(str(response))
```

The query engine retrieves the relevant documents and generates a response using the LLM.

# Challenges and Solutions

## Efficient Retrieval of Relevant Information

In large datasets, ensuring both high recall and precision during retrieval can be challenging. Traditional methods (like BM25) are strong with keyword-based search but may miss semantically relevant documents.

### Solution

The hybrid retriever, combining both BM25 and vector-based search, was designed to address this. By merging the results from both retrieval methods, we ensured that the retrieval system could cover both semantic and keyword relevance.

## Latency in Large Query Processing

Performing both BM25 and vector-based search increases the time taken for queries to be processed.

### Solution

We optimized the retrieval pipeline by limiting the number of top-k results from each retriever. Reducing the number of documents considered per retrieval (top-k) while still maintaining relevance was key to addressing latency issues.

# Key Decisions and Justifications

## Using BM25 and Semantic Search Together

The decision to use both BM25 and semantic search was based on the need to provide a comprehensive retrieval solution. BM25 works well for exact keyword matches, while semantic search captures the meaning behind queries. This combination helped in covering different aspects of relevance.

## Model Selection

The decision to utilize **Groq's large language model (LLM)** and **NVIDIA embedding models** was based on their demonstrated performance, robustness, and efficiency. These models consistently deliver high-quality semantic understanding and responses. Additionally, their **cost-effectiveness and speed** further enhance their suitability for handling complex tasks such as semantic search and natural language processing.

# Part-2: Building the Chatbot

## Overview

The Q/A Chatbot application allows users to upload PDF documents, ask questions related to the content of those documents, and receive answers in real-time. Built using Streamlit and the Llama Index library, the app leverages advanced language models for effective information retrieval and response generation.

## User Instructions

### 1. Uploading Files

- **Step 1**: Navigate to the application interface where you will see a section labeled "Upload a PDF file."
- **Step 2**: Click on the "Browse" button or drag and drop your PDF file into the designated area.
- **Step 3**: Once the file is uploaded, you will see a confirmation message stating "PDF File Uploaded Successfully!"

### 2. Asking Questions

- **Step 4**: After uploading the PDF, you will find an input box labeled "Ask a Question."
- **Step 5**: Type your question regarding the content of the uploaded PDF.
- **Step 6**: Click the "Get Answer" button to submit your question.

### 3. Viewing Bot Responses

- **Step 7**: Upon submitting your question, the bot will process the request and generate an answer.
- **Step 8**: The response will be displayed under the "Answer:" heading.
- **Step 9**: Below the answer, you will see the "Chunks used:" section, which lists excerpts from the PDF that were referenced in formulating the answer.

## Technical Implementation

### Architecture

- **Streamlit**: The front-end framework used for building interactive web applications. It allows for rapid development and easy deployment.
- **Llama Index**: A library for document handling and natural language processing. It incorporates various models to embed and query text efficiently.
- **PDF Processing**: The application uses PyMuPDF to read and extract text from PDF files. This library supports complex PDF structures, ensuring accurate data extraction.

### Environment Setup

- **Dependencies**: The application requires several Python packages, including streamlit, llama-index, pymupdf, and python-dotenv. These dependencies are listed in a requirements.txt file, making setup straightforward.

### Security and Performance

- **API Key Management**: The application securely retrieves API keys from a .env file, ensuring sensitive information is not hardcoded into the source code.
- **Response Time**: The application is optimized for quick responses, with efficient text processing and retrieval mechanisms that minimize latency for the user.

## Example Interactions

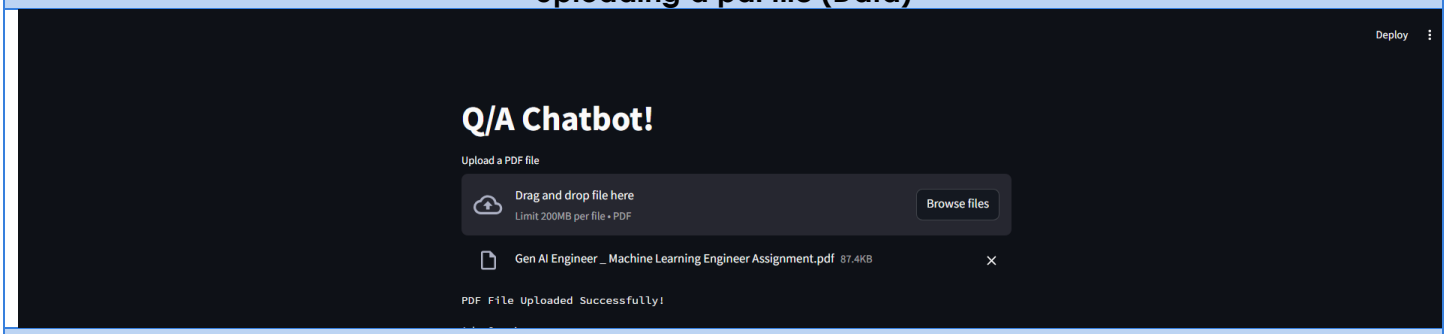
**Example Interaction 1**

**User Action**: Uploads a PDF titled "Jessup Cellars Wine Company"

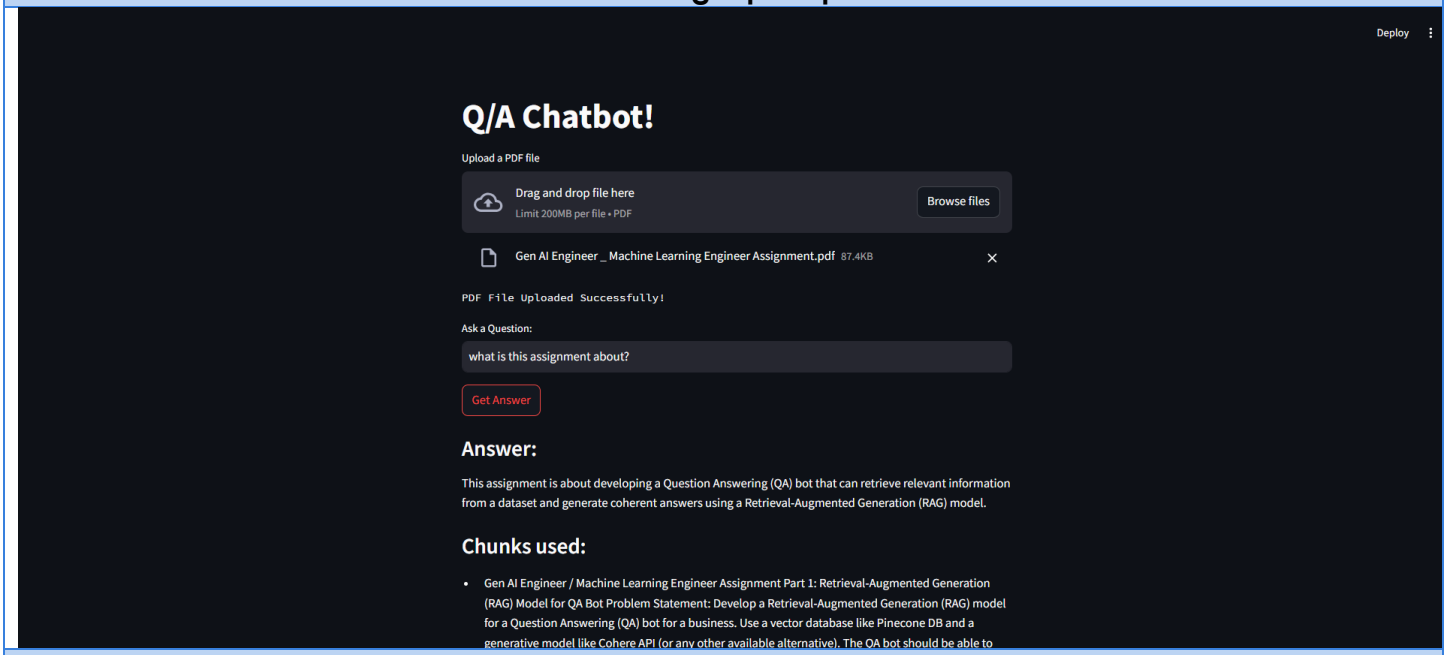
- **User Question**: "Are Dogs allowed at Jessup Cellars?"
- **Bot Response**: ""
- **Chunks Used**:
    - [7 Chunks of Data were used..(Listed after output)]

# Chatbot UI

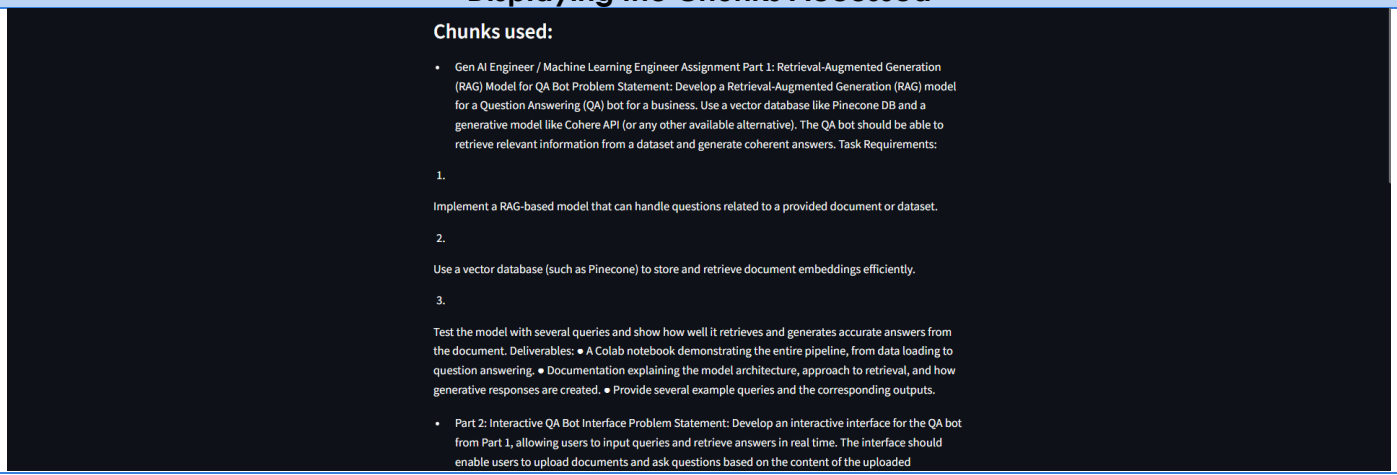
**Uploading a pdf file (Data)**




**Entering a prompt**




**Displaying the Chunks Accessed**



# Challenges Faced

During the development of the Q/A Chatbot application, several challenges were encountered:

## Model Integration:

- **Challenge**: Integrating multiple models for embeddings and language processing introduced complexity in configuration and usage.
- **Solution**: Careful planning and testing of the Llama Index settings allowed for smoother integration, ensuring optimal performance of the embedding and querying process.

## User Input Handling:

- **Challenge**: Ensuring the application can handle unexpected user inputs gracefully was crucial to maintain a good user experience.
- **Solution**: Implementing checks for valid questions and providing clear feedback helped enhance user interaction.

## Performance Optimization:

- **Challenge**: Balancing response time with the accuracy of answers was an ongoing concern, particularly with larger documents.
- **Solution**: Optimizing the retrieval algorithm and refining the model settings improved the efficiency of the query responses.

# Docker Containerization

To facilitate deployment and ensure the application runs consistently across different environments, the code docker is containerized.

This Q/A Chatbot application provides a user-friendly interface for interacting with PDF content, enabling users to gain insights and information efficiently. With its capability to extract relevant data from uploaded documents, it serves as a powerful tool for information retrieval in various contexts.