An LLM-Based RAG System for Question Answering over User-Provided Documents

by

S. M. Shufol Islam
ID: CSE2203027065

Md.Ashraful Islam
ID: CSE2203027036

Shahin Alam
ID: CSE2203027017

Meherun Nesa Juie
ID: CSE2203027040

Supervised by
Salma Tabashum

Submitted in partial fulfillment of the requirements for the degree of
Bachelor of Science in Computer Science and Engineering



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SONARGAON UNIVERSITY (SU)
September 2026

APPROVAL

The thesis titled "**An LLM-Based RAG System for Question Answering over UserProvided Documents**" submitted by **S. M. Shufol Islam** (CSE2203027065), **Md. Ashraful islam** (CSE2203027036), **Shahin Alam** (CSE2203027017) and **Meherun Nesa Juie** (CSE2203027040) to the Department of Computer Science and Engineering, Sonargaon University (SU), has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering and approved as to its style and contents.

Board of Examiners

Salma Tabashum
Lecturer & Coordinator,
Department of Computer Science and Engineering
Sonargaon University (SU)

Supervisor

(Examiner Name and Signature)
Department of Computer Science and Engineering
Sonargaon University (SU)

Examiner 1

(Examiner Name and Signature)
Department of Computer Science and Engineering
Sonargaon University (SU)

Examiner 2

(Examiner Name and Signature)
Department of Computer Science and Engineering
Sonargaon University (SU)

Examiner 3

DECLARATION

We, hereby, declare that the work presented in this report is the outcome of the investigation performed by us under the supervision of **Salma Tabashum**, Lecturer & Coordinator**,** Department of Computer Science and Engineering, Sonargaon University, Dhaka, Bangladesh. We reaffirm that no part of this thesis has been or is being submitted elsewhere for the award of any degree or diploma.

Countersigned

(Salma Tabashum)
Supervisor

Signature

S. M. Shufol Islam
ID: CSE2203027065

Md.Ashraful Islam
ID: CSE2203027036

Shahin Alam
ID: CSE2203027017

Meherun Nesa Juie
ID: CSE2203027040

ABSTRACT

The rapid growth of digital documents has created a need for efficient methods of accessing and understanding information contained within them. Traditional document-reading approaches require users to manually search through large amounts of text, which can be time-consuming and inefficient. This project presents an LLM-based Retrieval-Augmented Generation (RAG) system for question answering over user-provided documents, designed to provide users with relevant and context-aware answers from their uploaded documents.

The proposed system combines semantic information retrieval with a Large Language Model (LLM) to improve document-based question answering. Users can upload supported documents to the system, after which the document content is extracted and divided into smaller text chunks. These chunks are transformed into numerical vector representations using a sentence-transformer-based embedding model and stored in ChromaDB, a vector database. When a user submits a question, the system converts the query into an embedding and performs semantic similarity search against the stored document chunks. The most relevant chunks are then retrieved and provided as contextual information to the LLM. The LLM generates an answer based primarily on the retrieved context, allowing the system to respond to questions without requiring the entire document to be supplied to the model for every query.

The system consists of a React-based frontend, a FastAPI backend, a ChromaDB vector store, a sentence-transformer embedding model, and an LLM accessed through the Groq API. The application also provides source information associated with retrieved document content, allowing users to identify the document from which the answer was generated. The separation between document processing, semantic retrieval, and answer generation provides a modular architecture that can be extended with different language models or retrieval configurations.

The developed system demonstrates the practical application of Retrieval-Augmented Generation for document-based question answering while reducing dependence on manually searching through lengthy documents. Although the current implementation has limitations related to document formats, retrieval accuracy, context size, and dependence on external LLM services, the architecture provides a strong foundation for future improvements such as multi-document knowledge management, conversation memory, improved retrieval techniques, document management interfaces, authentication, advanced source citation, and more sophisticated question-answering capabilities.

ACKNOWLEDGEMENT

At the very beginning, we would like to express my deepest gratitude to the Almighty Allah for giving us the ability and the strength to finish the task successfully within the scheduled time.

We are auspicious that we had the kind association as well as supervision of **Salma Tabashum**, Lecturer & Coordinator, Department of Computer Science and Engineering, Sonargaon University, whose heartfelt and valuable support with the best concern and direction acted as a necessary recourse to carry out our project.

We would like to convey our special gratitude to **Prof. Bulbul Ahamed**, Head, Department of Computer Science and Engineering, for his kind concern and precious suggestions.

We are also thankful to all our teachers during our whole education for exposing us to the beauty of learning.

Finally, our deepest gratitude and love to my parents for their support, encouragement, and endless love.

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
CPU	Central Processing Unit
GPU	Graphics Processing Unit
LLM	Large Language Model
NLP	Natural Language Processing
PDF	Portable Document Format
QA	Question Answering
RAG	Retrieval-Augmented Generation
RAM	Random Access Memory
UI	User Interface

TABLE OF CONTENTS

Title		Page No.
DECLARATION …….……..		iii
ABSTRACT ……...……		iv
ACKNOWLEDGEMENT …….……		v
LIST OF ABBREVIATION …….……		vi
CHAPTER 1		1 – 7
INTRODUCTION TO AUTOMATIC SPEECH RECOGNITION		
1.1	Introduction ……..……...	1
1.2	Background of the Study ……..…..	2
1.3	Problem Statement …….……	3
1.4	Motivation ……..…….....	4
1.5	Objectives ……..…….....	5
1.5.1	General Objective ……..……	5
1.5.2	Specific Objectives ……..……...	5
1.6	Scope of the Thesis ……..……...	6
CHAPTER 2		8 – 18
LITERATURE REVIEW		
2.1	Introduction …….….....	8
2.2	Traditional Information Retrieval Systems …….....	8
2.3	Limitations of Standalone Large Language Models…….....	9
2.3.1	Limited Access to Private Information…….....	9
2.3.2	Hallucination…….....	9
2.3.3	Outdated Knowledge…….....	9
2.3.4	Lack of Source Grounding…….....	10
2.4	Retrieval-Augmented Generation…….....	10
2.5	RAG Architecture…….....	11
2.5.1	Document Ingestion.....	11
2.5.2	Text Extraction	11
2.5.3	Text Chunking	11
2.5.4	Embedding Generation	11

2.5.5 Vector Storage and Retrieval 12
2.5.6 Context Construction 12
2.5.7 Answer Generation 12
2.6 Semantic Search 12
2.7 Text Embeddings 13
2.8 Vector Databases 13
2.9 Document-Based Question Answering Systems 14
2.10 Research Gap 16
2.11 Relevance of the Proposed System 16
2.12 Conceptual Framework of the Proposed System 17
2.13 Summary 18

CHAPTER 3 19 – 28

DATASET AND KNOWLEDGE BASE ANALYSIS

3.1 Introduction 19
3.2 Dataset Sources 19
3.3 Raw Dataset Description 20
3.3.1 Sonargaon University Information Book -1 20
3.3.2 Sonargaon University Information Book -2 21
3.3.3 Sonargaon University Room Info 23
3.3.4 Sonargaon University Teachers Info 24
3.4 Dataset Statistics 26
3.5 Raw Dataset Analysis 27
3.6 Text Extraction 27
3.7 Chapter Summary 28

CHAPTER 4 29 – 42

SYSTEM ARCHITECTURE AND METHODOLOGY

4.1 Introduction 29
4.2 Overview of the Proposed System 29
4.3 Overall System Architecture 31
4.4 Document Upload and Processing 32
4.5 Text Extraction 32

4.6 Document Chunking 33
4.7 Text Embedding Generation 34
4.8 Vector Storage Using ChromaDB 34
4.9 Query Processing and Semantic Retrieval 35
4.10 Context Construction 36
4.11 Answer Generation Using a Large Language Model 37
4.12 Source Identification 38
4.13 Frontend and Backend Communication 38
4.14 Complete System Workflow 39
4.14.1 Document Knowledge Base Creation 39
4.14.2 Question Answering Process 40
4.15 Chapter Summary 42

CHAPTER 5 43 – 55

SYSTEM IMPLEMENTATION

5.1 Introduction 43
5.2 Development Environment 43
5.3 Technology Stack 44
5.4 Frontend Implementation 45
5.5 Backend Architecture 45
5.6 Document Processing Implementation 46
5.7 Embedding and Vector Database Implementation 47
5.8 Semantic Retrieval Implementation 48
5.9 LLM Integration 59
5.10 Prompt and Context Construction 59
5.11 Source Attribution 50
5.12 API Integration 51
5.13 Error Handling 52
5.14 User Interface Implementation 52
5.15 Complete Implementation Workflow 53
5.16 Chapter Summary 55

CHAPTER 6 56 – 63

RESULTS AND DISCUSSION

6.1 Introduction 56

6.2 Experimental Environment 56
6.3 Document Upload Results 57
6.4 Question Answering Results 59
6.5 Semantic Retrieval Results 61
6.6 Discussion 63
CHAPTER 7 64 – 71
CONCLUSION AND FUTURE WORKS
7.1 Conclusion 64
7.2 Contributions of the Project 64
7.2.1 End-to-End RAG Implementation 64
7.2.2 User-Provided Knowledge Integration 65
7.2.3 Semantic Document Retrieval 65
7.2.4 LLM-Based Natural Language Generation 65
7.2.5 Source-Aware Responses 65
7.2.6 Modular Software Architecture 65
7.3 Limitations 65
7.4 Future Work 66
7.4.1 Improved Document Processing 66
7.4.2 Improved Retrieval Techniques 66
7.4.3 Persistent Database Architecture 67
7.4.4 Authentication and Authorization 67
7.4.5 Advanced Source Citation 67
7.4.6 Conversation Memory 68
7.4.7 Examination Question Analysis 68
7.4.8 Multiple Knowledge Bases 69
7.4.9 Local and Privacy-Focused Deployment 69
7.4.10 Quantitative Evaluation 70
7.5 Final Remarks 70
REFERENCES 72

LIST OF TABLES

Table No.	Title	Page No.
Table 3.1	Documents Used in the Knowledge Base	19
Table 3.2	Dataset Statistics	27
Table 4.1	Document Chunk Storage Structure in ChromaDB	35
Table 5.1	Technology Stack of the Proposed System	44
Table 6.1	Experimental Environment	57

LIST OF FIGURES

Figure No.	Title	Page No.
Fig.3.1	Raw CSE Guide Book Used as a Knowledge Source	21
Fig.3.2	Raw CSE Service Charter Used as a Knowledge Source	23
Fig.3.3	Sonargaon University Room Info	24
Fig.3.4	Raw Faculty Information Dataset	26
Fig.3.5	Example of Extracted Text from a Source Document	28
Fig. 4.1	Architecture of User's Journey	31
Fig. 4.2	Question Answering Process	41
Fig.6.1	Document uploading process from the admin side	58
Fig.6.2	Document uploading process from the admin side	58
Fig.6.3	Backend Server log of document upload	59
Fig.6.4	Question Answering Result	60
Fig.6.5	Question Answering Result	60
Fig.6.6	Source Data	62
Fig.6.7	Semantic Retrieval and Source Result	62
Fig.6.8	Semantic Retrieval and Source Result	63

CHAPTER 1
INTRODUCTION

1.1 Introduction

The rapid growth of digital information has significantly increased the amount of documents generated, stored, and shared by individuals, educational institutions, organizations, and businesses. Documents such as PDF files, Word documents, reports, academic materials, manuals, policies, guidelines, and other textual resources often contain valuable information. However, finding specific information from a large collection of documents can be time-consuming and inefficient when users need to manually read or search through the documents.

Traditional keyword-based search systems can help users locate documents containing particular words or phrases, but they often fail to understand the actual meaning or context of a user's question. For example, a user may ask a question using words that are different from the exact words used in a document. In such cases, a traditional search system may fail to provide relevant information even when the required answer exists within the document.

Recent advancements in Artificial Intelligence, particularly Large Language Models (LLMs), have created new opportunities for developing intelligent question-answering systems. LLMs are capable of understanding natural language and generating human-like responses. However, a general-purpose LLM does not automatically have access to the private, personal, or newly uploaded documents of a user. Asking an LLM questions directly without providing relevant document information may result in inaccurate answers or responses based on the model's general knowledge.

Retrieval-Augmented Generation (RAG) addresses this limitation by combining information retrieval techniques with the generative capabilities of Large Language Models. In a RAG-based system, documents are first processed and converted into a searchable knowledge base. When a user asks a question, the system retrieves the most relevant information from the provided documents and supplies that information to the language model as context. The language model then generates an answer based on the retrieved context.

This thesis presents the design and implementation of an **LLM-Based Retrieval-Augmented Generation System for Question Answering over User-Provided Documents**. The developed system allows users to provide documents and ask questions related to their content. The documents are processed, divided into smaller text chunks, converted into vector embeddings, and stored in a vector database. When a user submits a question, the system performs semantic similarity search to identify relevant document chunks. The retrieved information is then provided to a Large Language Model to generate a context-aware response.

The proposed system integrates several modern technologies. The backend is developed using **FastAPI**, while the frontend is developed using **React and Vite**. Document embeddings are generated using the **SentenceTransformers all-MiniLM-L6-v2** model,

and **ChromaDB** is used as the vector database for storing and retrieving document embeddings. The language generation component is integrated through the **Groq API**, which provides access to a Large Language Model for generating responses.

An important feature of the proposed system is that the generated answers are intended to be based on information retrieved from the documents provided to the system. In addition, the system identifies the source document associated with the retrieved information, allowing users to understand where the answer originated.

The proposed system can be applied in various domains, including education, organizations, businesses, customer support, research, and personal knowledge management. For example, students may upload course materials and ask questions related to their academic content. Organizations may use internal documents to create a document-based information assistant. Similarly, businesses may use manuals, policies, product documentation, or knowledge base documents to assist users in finding relevant information.

Therefore, the proposed system demonstrates how Retrieval-Augmented Generation can be used to create a practical and intelligent question-answering application that combines semantic document retrieval with the natural language generation capabilities of Large Language Models.

1.2 Background of the Study

The availability of digital documents has transformed the way information is stored and accessed. Educational institutions, businesses, government organizations, and individuals increasingly depend on digital files to preserve and distribute information. A single organization may maintain hundreds or thousands of documents containing policies, reports, guidelines, manuals, academic materials, and other important information.

Although storing information digitally has become easier, efficiently retrieving specific information remains a significant challenge. Users often need to manually browse through multiple documents or use conventional keyword-based search systems to locate the required information. This process becomes increasingly difficult when the number and size of documents increase.

Traditional search systems primarily depend on keyword matching. These systems generally search for documents containing the same or similar words used in a query. However, natural language allows the same concept to be expressed using different words and sentence structures. As a result, keyword-based search may not always identify the most relevant information.

For example, a document may contain information about the process for obtaining an academic transcript, while a user asks, "What should I do to collect my university result document?" Although the meaning of the question may be related to the information in the document, the exact keywords may differ. A semantic understanding of the query and document content is therefore necessary for more effective information retrieval.

The development of machine learning and Natural Language Processing has introduced more advanced approaches to information retrieval. Text embedding models can convert textual information into numerical vector representations that capture semantic meaning.

These vector representations make it possible to compare the meaning of a user query with the meaning of document content.

Vector databases are designed to efficiently store and search these embeddings. Instead of relying only on exact keyword matches, vector similarity search can identify text chunks that are semantically related to a user's question.

At the same time, Large Language Models have demonstrated significant capabilities in understanding and generating natural language. Models can produce conversational and human-readable responses to a wide range of questions. However, a general-purpose LLM may not contain information about a user's private documents or organization-specific knowledge. Furthermore, allowing a language model to answer without providing relevant evidence can lead to inaccurate or unsupported responses.

Retrieval-Augmented Generation provides a solution by connecting document retrieval with language generation. Rather than expecting the language model to answer solely from its internal knowledge, relevant information is retrieved from an external knowledge source and provided to the model during answer generation.

The basic process can be represented as follows:

User Documents → Text Extraction → Chunking → Embedding Generation → Vector Database

When a user asks a question:

User Question → Query Embedding → Similarity Search → Relevant Document Chunks → LLM → Generated Answer

This approach allows the system to dynamically use information from user-provided documents without requiring the language model to be retrained whenever new documents are added.

The development of accessible APIs, open-source embedding models, vector databases, and modern web frameworks has made it possible to develop practical RAG applications without requiring extensive computational infrastructure for training large language models from scratch.

Based on these developments, this research focuses on developing an LLM-based RAG system capable of answering questions over user-provided documents. The system aims to demonstrate a complete workflow starting from document processing and semantic retrieval to context-aware answer generation through an interactive web-based interface.

1.3 Problem Statement

Organizations and individuals frequently store important information in documents such as PDF, DOCX, and text files. Although these documents contain valuable information, users often face difficulties when trying to quickly locate specific answers.

The traditional approach requires users to manually read documents or search for specific keywords. This process can become inefficient when documents are lengthy or when

multiple documents are involved. Furthermore, conventional keyword-based search systems may fail to understand the semantic meaning of a user's question.

Large Language Models offer a potential solution by providing natural language interaction. However, a general-purpose LLM does not automatically have access to private or user-provided documents. If a user asks questions about information that is not included in the model's general knowledge, the model may provide incorrect information, incomplete responses, or generated content that is not supported by the actual documents.

Another challenge is maintaining a connection between the generated answer and its original source. Users may want to know which document contains the information used to generate an answer. Without source identification, it becomes more difficult to verify the reliability of the generated response.

Therefore, the main problem addressed in this research is the development of a system that can:

- Accept user-provided documents.
- Process and organize the textual information contained within those documents.
- Retrieve information based on the semantic meaning of a user's question.
- Provide relevant retrieved information to a Large Language Model.
- Generate answers based on the available document context.
- Identify the source document associated with the retrieved information.
- Provide an accessible web-based interface for document-based question answering.

The proposed RAG-based approach is designed to address these challenges by combining semantic retrieval, vector databases, and Large Language Models into a single integrated application.

1.4 Motivation

The primary motivation behind this research is the increasing difficulty of efficiently accessing information from large collections of digital documents. Although a significant amount of knowledge is available in documents, accessing the exact information required by a user can still require considerable time and effort.

During the development and exploration of modern Artificial Intelligence applications, it became evident that Large Language Models provide an effective way for users to interact with information through natural language. However, an LLM alone cannot automatically access user-specific or private documents. This limitation creates a need for systems that can connect external document knowledge with the language generation capabilities of modern AI models.

Retrieval-Augmented Generation provides an effective architecture for addressing this problem. Instead of training a separate language model using every new document, relevant information can be retrieved dynamically whenever a user asks a question. This approach makes the system more flexible and practical for real-world applications.

Another motivation for the project is the potential usefulness of such a system in multiple domains. For example:

- Students can upload academic materials and ask questions about the content.
- Teachers can provide course documents as a knowledge source.
- Organizations can use internal policies and guidelines for employee assistance.
- Businesses can use product documentation and knowledge bases for customer support.
- Researchers can interact with research materials more efficiently.
- Individuals can create personalized document-based knowledge systems.

The project also provides an opportunity to integrate several modern technologies into a complete end-to-end application. Rather than focusing only on a theoretical RAG pipeline, the objective is to develop a functional web application that includes document processing, semantic search, vector storage, LLM-based answer generation, API communication, and a user-friendly interface.

Therefore, the motivation of this research is both academic and practical: to explore the capabilities of Retrieval-Augmented Generation while developing a functional system that can make information stored in user-provided documents easier to access through natural language interaction.

1.5 Objectives

1.5.1 General Objective

The main objective of this research is to design and develop an **LLM-Based Retrieval-Augmented Generation system for answering natural language questions using information retrieved from user-provided documents**.

1.5.2 Specific Objectives

The specific objectives of this research are:

1. To develop a web-based application that allows users to interact with a document-based question-answering system.
2. To implement a document processing mechanism for extracting textual information from supported user-provided documents.
3. To divide extracted document text into smaller chunks suitable for semantic retrieval.
4. To generate vector embeddings from document chunks using a sentence embedding model.
5. To store document embeddings and associated metadata in a vector database.
6. To implement semantic similarity search for retrieving relevant document chunks based on user queries.
7. To integrate a Large Language Model through an API for generating natural language answers.

8. To provide the retrieved document context to the LLM during answer generation.
9. To restrict the answer generation process to the information available in the retrieved document context as much as possible.
10. To display source information associated with retrieved documents to improve answer transparency.
11. To integrate the frontend and backend into a complete document-based conversational application.
12. To evaluate the practical functionality, strengths, and limitations of the developed system.

1.6 Scope of the Thesis

The scope of this thesis is focused on the development of a web-based Retrieval-Augmented Generation system for question answering over user-provided documents.

The system supports the process of uploading and processing documents and using their content as a knowledge source for answering user questions. The main scope of the developed system includes:

- Uploading supported document files.
- Extracting textual content from documents.
- Processing document content into smaller chunks.
- Generating semantic embeddings from text chunks.
- Storing embeddings in ChromaDB.
- Performing semantic similarity search based on user questions.
- Retrieving the most relevant document chunks.
- Supplying retrieved context to a Large Language Model.
- Generating natural language answers.
- Displaying source document information associated with retrieved content.
- Providing a web-based conversational interface.

The project focuses primarily on the RAG workflow rather than training a Large Language Model from scratch. The system uses an existing embedding model and accesses a Large Language Model through an external API.

The current implementation is designed as a prototype and research-oriented application. It demonstrates the complete process of document ingestion, embedding generation, vector retrieval, context construction, and LLM-based answer generation.

However, certain advanced features are outside the primary scope of the current implementation. These include:

- Training a Large Language Model from scratch.
- Fine-tuning a Large Language Model using a custom dataset.

- Developing a separate Small Language Model for comparison.
- Advanced multi-user authentication and authorization.
- Large-scale distributed document processing.
- Complex conversation memory across multiple persistent user sessions.
- Advanced administrative analytics.
- Enterprise-level cloud deployment and scaling.
- Support for every possible document format.
- Guaranteed correctness of every generated response.

The system is therefore intended to demonstrate the practical implementation and capabilities of an LLM-based RAG architecture for document question answering.

CHAPTER 2
LITERATURE REVIEW

2.1 Introduction

The rapid growth of digital documents has created a significant need for intelligent systems capable of retrieving relevant information and providing accurate answers from large collections of unstructured data. Organizations, educational institutions, businesses, and individual users frequently store important information in formats such as PDF, DOCX, and TXT files. Although these documents contain valuable knowledge, finding specific information manually can be time-consuming and inefficient.

Traditional keyword-based search systems can retrieve documents containing specific words, but they often fail to understand the actual meaning or semantic relationship behind a user's question. With recent advancements in Artificial Intelligence (AI), Natural Language Processing (NLP), and Large Language Models (LLMs), new approaches have emerged to improve information retrieval and question-answering systems.

One of the most important approaches is **Retrieval-Augmented Generation (RAG)**. A RAG system combines information retrieval with the generative capabilities of an LLM. Instead of relying only on the knowledge already contained within a language model, the system first retrieves relevant information from an external knowledge source and then uses that information to generate a response.

This chapter reviews the fundamental concepts, technologies, and previous approaches related to document-based question answering, Large Language Models, semantic search, vector databases, embeddings, and Retrieval-Augmented Generation. It also discusses the limitations of traditional approaches and identifies the research gap that motivates the development of the proposed system.

2.2 Traditional Information Retrieval Systems

Information Retrieval (IR) refers to the process of finding relevant information from a collection of documents based on a user's query. Traditional search systems generally depend on keyword matching techniques.

For example, if a user searches:

> "What are the requirements for admission?"

a traditional search system may attempt to find documents containing words such as:

- admission
- requirements
- apply

However, this approach has several limitations. A document may contain relevant information but use different words, such as:

"Students must submit the required academic certificates during enrollment."

Although this sentence is related to admission requirements, a simple keyword-based system may not identify it as highly relevant if the exact search terms are not present.

Traditional retrieval systems commonly use techniques such as:

- Boolean Retrieval
- Term Frequency–Inverse Document Frequency (TF-IDF)
- Bag of Words
- BM25

These techniques have been widely used because they are computationally efficient and relatively easy to implement. However, they primarily focus on lexical similarity rather than semantic meaning.

This limitation becomes significant when users ask questions using natural language. A user may phrase the same question in many different ways, and a keyword-based system may fail to retrieve the correct information.

Therefore, modern question-answering systems increasingly use semantic representations and neural language models to improve retrieval quality.

2.3 Limitations of Standalone Large Language Models

Although LLMs have demonstrated strong language understanding and generation capabilities, using them without external information sources can create several problems.

2.3.1 Limited Access to Private Information

An LLM does not automatically have access to a user's uploaded documents. If a user uploads several PDF files containing specific information, the model cannot answer questions from those documents unless the relevant information is provided to the model.

2.3.2 Hallucination

Hallucination occurs when an AI model generates information that appears reasonable but is not supported by factual data or the provided context.

For example, if a user asks about a specific policy that is not available in the model's knowledge, the model may generate a confident but incorrect answer.

This is a significant issue for document-based question-answering systems where accuracy and faithfulness to source documents are important.

2.3.3 Outdated Knowledge

The internal knowledge of an LLM may not include recent information. Documents may contain updated policies, newly published research, current regulations, or recent organizational information.

2.3.4 Lack of Source Grounding

A general LLM may provide an answer without identifying the document from which the information originated.

For systems dealing with user-provided documents, it is beneficial to retrieve relevant source information along with the generated response.

These limitations demonstrate the importance of combining LLMs with external knowledge retrieval mechanisms.

2.4 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) is an approach that combines two major components:

1. **Information Retrieval**
2. **Natural Language Generation**

Instead of sending only the user's question directly to an LLM, a RAG system first searches a knowledge base for relevant information.

The general workflow can be represented as:

User Question → Retrieval → Relevant Document Chunks → LLM → Generated Answer

In the proposed system, the workflow is broadly organized as follows:

1. The user uploads one or more documents.
2. The system extracts textual information from the uploaded documents.
3. The extracted text is divided into smaller segments called chunks.
4. Each chunk is converted into a numerical vector representation.
5. These vectors are stored in a vector database.
6. When a user asks a question, the question is converted into an embedding.
7. The system retrieves the most relevant document chunks.
8. The retrieved context and user question are sent to the LLM.
9. The LLM generates an answer based on the retrieved context.
10. The system returns the answer along with relevant source information.

The primary advantage of RAG is that the model can generate answers based on external and user-provided knowledge rather than relying entirely on its pretrained knowledge.

2.5 RAG Architecture

A typical RAG architecture consists of several interconnected components.

2.5.1 Document Ingestion

The first stage involves receiving documents from the user. Depending on the system design, documents may include formats such as:

- PDF
- DOCX
- TXT

The proposed system is designed around user-provided documents that are processed and transformed into searchable knowledge.

2.5.2 Text Extraction

After a document is uploaded, the system extracts textual content from the file.

The quality of extracted text is important because the RAG system depends on this information during retrieval and answer generation.

2.5.3 Text Chunking

Documents can contain large amounts of information. Sending an entire document directly to an LLM for every user query is inefficient and may exceed context limitations.

Therefore, the extracted text is divided into smaller pieces known as **chunks**.

Chunking provides several benefits:

- Faster retrieval
- More focused context
- Reduced unnecessary information
- Improved semantic search
- Better use of LLM context

For example, a 100-page document may be divided into hundreds of smaller text chunks. When a user asks a question, the system retrieves only the chunks that are most relevant.

2.5.4 Embedding Generation

An embedding is a numerical representation of text that captures semantic meaning.

For example, the sentences:

"What is the admission fee?"

and

"How much does enrollment cost?"

contain different words but have similar meanings.

Embedding models can represent semantically similar text in nearby regions of a vector space.

In the proposed system, the **sentence-transformers/all-MiniLM-L6-v2** model is used to generate embeddings.

This model converts document chunks and user queries into vector representations that can be compared using similarity-based retrieval.

2.5.5 Vector Storage and Retrieval

The generated embeddings are stored in a vector database.

When a user asks a question:

1. The question is converted into an embedding.
2. The embedding is compared with the stored document embeddings.
3. The most semantically relevant chunks are retrieved.

The proposed system uses **ChromaDB** as the vector database for storing and retrieving document embeddings.

2.5.6 Context Construction

After retrieving the most relevant chunks, the system combines them into a context.

The context is then provided to the LLM together with the user's question.

The LLM is instructed to answer only using the retrieved information.

This helps reduce hallucination and ensures that responses remain grounded in the uploaded documents.

2.5.7 Answer Generation

The final stage uses an LLM to generate a natural-language answer.

The current implementation of the proposed system uses an LLM accessed through the **Groq API**. The selected model is configured separately, allowing the application to change or replace the LLM without requiring major changes to the rest of the system architecture.

This modular approach improves flexibility and allows the system to adapt to future model availability and performance requirements.

2.6 Semantic Search

Semantic search is a search technique that focuses on the meaning of a query rather than relying only on exact keyword matching.

Consider the following question:

"How can I apply for the program?"

A document might contain:

> "Applicants must complete the registration process before enrollment."

Although the words are different, the meanings are related.

A semantic search system can identify this relationship by converting both the question and document text into embeddings.

The similarity between these embeddings can then be used to identify relevant information.

The proposed system uses semantic search to retrieve the most relevant document chunks before generating an answer.

This approach is particularly useful when:

- Users phrase questions differently from the original document.
- Exact keywords are not present.
- Documents contain large amounts of information.
- Natural-language interaction is required.

2.7 Text Embeddings

Text embeddings are numerical vector representations that capture semantic information.

For example, an embedding model converts:

> "The course registration deadline is Friday."

into a numerical vector containing multiple dimensions.

Another sentence with a similar meaning may generate a vector located close to the first one in the vector space.

The similarity between vectors can be measured to determine how relevant a document chunk is to a user query.

Embedding-based retrieval is one of the fundamental components of modern RAG systems.

The proposed system uses the **all-MiniLM-L6-v2** sentence-transformer model because it provides a lightweight and practical solution for generating semantic embeddings.

The embedding process allows the application to retrieve information based on meaning rather than exact keyword matching.

2.8 Vector Databases

A vector database is designed to store and search high-dimensional vector representations.

Unlike traditional relational databases, which primarily retrieve information using exact values or structured queries, vector databases support similarity-based searching.

In a RAG system, a vector database typically stores:

- Text chunk embeddings
- Original text chunks
- Metadata associated with the chunks

When a user submits a query, the system searches for vectors that are semantically similar to the query vector.

The proposed system uses **ChromaDB** for vector storage and similarity search.

The use of a vector database provides several advantages:

- Efficient similarity search
- Storage of semantic embeddings
- Retrieval of relevant document chunks
- Metadata support
- Scalability for larger document collections

In the proposed architecture, ChromaDB acts as the knowledge retrieval component of the application.

2.9 Document-Based Question Answering Systems

Document-based question answering systems are designed to answer user questions using information contained within specific documents.

Such systems can be useful in many real-world scenarios.

Education

Students may upload:

- Lecture notes
- Previous examination questions
- Course materials
- Research papers

The system can help users find and understand information from those documents.

Organizations

Organizations may use internal documents such as:

- Policies
- Employee guidelines
- Training materials

- Operational manuals

Employees can ask questions instead of manually searching through multiple documents.

Customer Support

Businesses may provide product documentation, FAQs, and policy documents to a RAG system.

Customers can then ask questions and receive answers based on official documentation.

Research

Researchers can upload multiple papers and use a document-based AI assistant to explore information more efficiently.

These use cases demonstrate the flexibility of RAG-based systems.

Existing AI-Powered Document Interaction Systems

Several modern applications allow users to interact with documents using AI.

One well-known example is Google's **NotebookLM**, which allows users to provide source materials and ask questions based on those sources.

Other document intelligence platforms and AI-powered knowledge systems also provide features such as:

- Document summarization
- Question answering
- Information retrieval
- Source-based responses
- Knowledge organization

However, commercial systems are generally large-scale products with extensive infrastructure and additional features.

The purpose of the proposed system is different. Rather than attempting to replicate a complete commercial document intelligence platform, this research focuses on implementing and understanding the core pipeline of an **LLM-based RAG question-answering system**.

The proposed application provides a practical implementation of the following core concepts:

- User document upload
- Text extraction
- Document chunking

- Embedding generation
- Vector storage
- Semantic retrieval
- Context-grounded answer generation
- Source information

Therefore, the system represents a focused implementation of the fundamental architecture used in modern RAG-based applications.

2.10 Research Gap

Although many commercial AI tools and document-based assistants already exist, there remains value in developing a focused and understandable RAG-based system for user-provided documents.

Several challenges are important in such systems:

1. **Handling user-provided knowledge**
   The system should process documents that may not be publicly available or included in an LLM's pretrained knowledge.
2. **Retrieving semantically relevant information**
   Users may ask questions using different wording from the original document. Semantic retrieval is required to identify relevant content.
3. **Reducing unsupported answers**
   The generated response should be grounded in retrieved document content rather than relying entirely on the LLM's internal knowledge.
4. **Providing source awareness**
   Users should be able to understand which uploaded document contributed to the answer.
5. **Developing a modular architecture**
   The retrieval system and LLM provider should be sufficiently separated so that models can be changed when necessary.

The proposed research addresses these areas by implementing an LLM-based RAG system that processes user-provided documents and uses semantic retrieval to provide context-aware answers.

2.11 Relevance of the Proposed System

The proposed system is particularly relevant because large amounts of useful information remain stored in documents that are difficult to explore efficiently.

A user may upload multiple documents and ask questions such as:

- "What are the main requirements mentioned in these documents?"
- "What is the procedure described for registration?"
- "Which topics appear frequently in previous examination papers?"

- "Summarize the important information related to this subject."
- "What information is available about a particular topic?"

The system retrieves relevant content from the uploaded knowledge base and uses the LLM to generate a concise and natural-language response.

An additional possible application is the analysis of previous academic question papers. For example, if multiple previous years' examination papers are uploaded, the system can retrieve recurring topics and provide suggestions based on patterns present in the uploaded data.

However, such suggestions should be understood as **document-based analysis rather than guaranteed prediction**. The system can identify repeated or frequently occurring topics from the available documents, but it cannot guarantee future examination questions.

This demonstrates that the proposed architecture can be extended beyond simple document question answering into other knowledge-based applications.

2.12 Conceptual Framework of the Proposed System

The conceptual framework of the proposed system can be represented as follows:

User

↓

Document Upload

↓

Document Processing

↓

Text Extraction

↓

Text Chunking

↓

Embedding Generation

↓

ChromaDB Vector Storage

↓

User Question

↓

Query Embedding

↓

Semantic Similarity Search

↓

Top Relevant Chunks

↓

Context Construction

↓

LLM through Groq API

↓

Generated Answer

↓

Answer with Source Information

This framework combines document processing, semantic search, vector retrieval, and generative AI into a unified question-answering system.

2.13 Summary

This chapter reviewed the major concepts and technologies related to the proposed research.

Traditional keyword-based information retrieval systems provide basic document searching capabilities but often struggle to understand semantic meaning. Advances in NLP and transformer-based Large Language Models have significantly improved natural-language understanding and generation. However, standalone LLMs have limitations when dealing with private, domain-specific, or user-provided information.

Retrieval-Augmented Generation addresses these limitations by combining semantic information retrieval with LLM-based answer generation. The RAG pipeline retrieves relevant information from an external knowledge base and provides that information to the language model as context.

The proposed system applies this architecture to user-provided documents. It uses document processing, text chunking, the **sentence-transformers/all-MiniLM-L6-v2** embedding model, **ChromaDB** for vector storage and semantic retrieval, and an LLM accessed through the **Groq API** for answer generation.

CHAPTER 3
DATASET AND KNOWLEDGE BASE ANALYSIS

3.1 Introduction

This chapter presents the dataset and knowledge base used in the development and evaluation of the proposed LLM-based RAG system. Since the quality of a document-based question-answering system depends largely on the quality and structure of its source documents, the documents used in this project were analyzed before being indexed into the system.

The dataset consists of academic, administrative, departmental, and faculty-related documents. These documents were selected to provide different types of information that can be accessed through natural-language questions.

This chapter describes the sources of the documents, their contents, document-level statistics, raw data characteristics, and the preparation performed before the documents were added to the searchable knowledge base.

3.2 Dataset Sources

The knowledge base was created from documents containing information relevant to the intended use of the system. The selected documents were collected from university and departmental information sources and were used as the source of information for document retrieval and question answering.

The documents contain different forms of information, including academic guidelines, departmental services, office and location information, and faculty-related information. This variety was useful for testing whether the system could retrieve relevant information from different types of document content.

ID	Document	Main Information
D1	Sonargaon University Information Book-1	Academic programs, regulations, examinations, policies and student information
D2	Sonargaon University Information Book-2	Departmental services, procedures and responsible offices
D3	Sonargaon University Room Info	Office names, room numbers and departmental locations
D4	Sonargaon University Teachers Info	All the info of Faculty of CSE Dept and their interest area in research.
D5	Sonargaon University Data	All the available data that are available to Faculty and Department

Table 3.1: Documents Used in the Knowledge Base

3.3 Raw Dataset Description

3.3.1 Sonargaon University Information Book -1

The Sonargaon University Information Book -1 contains academic and departmental information related to the Computer Science and Engineering department. It includes information such as academic programs, regulations, examinations, student policies and other departmental guidelines. The document was included in the knowledge base because it provides structured institutional information that can be queried through the system.

The actual Guide Book contains the CSE academic/program information and other departmental material.

SONARGAON UNIVERSITY
Department of Computer Science and Engineering (CSE)
CSE Guide Book

Document Date: July 29, 2026

---

1. Department Overview

1.1 Historical Background

The Department of Computer Science and Engineering (CSE) at Sonargaon University was established in 2013 under the Faculty of Science and Engineering.

---

2. CSE Study Program

2.1 Available Undergraduate Program

The Department of Computer Science and Engineering offers one undergraduate program:

- Program: Bachelor of Science in Computer Science and Engineering
- Abbreviation: B.Sc. in CSE
- Program Duration: 4 years
- Total Credit Requirement: 162.0 credit hours

The courses and syllabus are prepared based on the course curricula of reputed national public universities and foreign universities. A syllabus review committee consisting of internal faculty members and an external advisor periodically reviews the courses and their contents.

---

3. B.Sc. in Computer Science and Engineering

3.1 Program Description

The B.Sc. in Computer Science and Engineering program is designed to develop a wide and practical skillset in computing.

The program emphasizes:

- Programming skills

12. General Student Policies

Policies concerning student behavior, plagiarism, falsification of records, and related matters are available at the Office of the Registrar.

---

13. Code of Conduct

Sonargaon University aims to maintain a healthy academic environment.

Students are expected to:

- Attend classes regularly.
- Use campus resources appropriately to support academic achievement.
- Maintain discipline.
- Keep the campus clean.
- Behave as responsible members of the university community.
- Act as good neighbors and responsible citizens.

Copies of the SU Code of Conduct are available in the library and on the department's website.

---

14. Disciplinary Committee

Faculty members, students, and staff may invoke the Disciplinary Committee through a written report concerning an offense or grievance.

The Disciplinary Committee:

- Acts independently to ascertain facts.



- Enforces existing university rules and policies based on its findings.
- Does not change or modify existing university rules and policies.

The Disciplinary Committee operates through the Office of the Vice Chancellor.

Figure 3.1: Raw CSE Guide Book Used as a Knowledge Source

3.3.2 Sonargaon University Information Book -2

The Sonargaon University Information Book -2 contains information about services provided by the department and the procedures students need to follow for different academic and administrative activities. The document includes information about services such as course pre-registration, semester drop, department or package changes, retake and improvement, examination-related procedures, and other student services.

SONARGAON UNIVERSITY
Department of Computer Science and Engineering (CSE)

Address: 147/I, Green Road, Panthapath, Tejgaon, Dhaka-1215

Service Charter

The Department of Computer Science & Engineering (CSE) at Sonargaon University is committed to providing efficient academic and administrative services to its students. This Service Charter outlines the available services, required processes, and responsible offices.

---

1. Course Pre-registration

Service: Course pre-registration

Process:
1. The department announces the pre-registration schedule.
2. Students must clear all outstanding dues through the online payment system.
3. Students then complete the pre-registration process through the SU ERP.

Responsible Room: 501

---

2. Semester Drop

Service: Semester Drop

Process:
- Submit the semester drop application through the SU ERP.

Responsible Room: 501

4. Retake / Improvement Registration

Service: Retake / Improvement Registration

Process:
1. Apply through the SU ERP.
2. Contact the Department Office to complete the process.

Responsible Room: 501

---

5. Supplementary / Overlap Examination

Service: Supplementary, Overlap Examination

Process:
- Join the **SUcseBD Facebook Group** for detailed instructions regarding supplementary and overlap examinations.

Official Facebook Group: SUcseBD

---

6. Examination Clearance

Service: Examination Clearance

Process:
1. Clear all outstanding dues.
2. Download the admit card from the SU ERP.
3. Students must complete these requirements to become eligible for the examination.

Responsible Rooms: 501, 202

Figure 3.2: Raw CSE Service Charter Used as a Knowledge Source

3.3.3 Sonargaon University Room Info

The Sonargaon University Room Info contains the physical location of important university and departmental offices. It provides office names and corresponding room numbers. For example, the document identifies the office of the Coordinator of the Department of Computer Science and Engineering as Room 501 and the Office of Accounts as Room 313.

This document was useful for testing questions where the answer depends on specific location information.

SONARGAON UNIVERSITY
Department of Computer Science and Engineering (CSE)

Address: 147/I, Green Road, Panthapath, Tejgaon, Dhaka-1215

Department Layout Plan

The following offices and facilities are located at Sonargaon University.

---

1. Office of the Vice Chancellor

Office Room: 301

---

2. Office of the Pro-Vice Chancellor

Office Room: 202

---

3. Office of the Treasurer

Office Room: 507

---

4. Office of the Registrar

Office Room: 209

---

5. Office of the Controller of Examinations

Office Room: 211

---

6. Office of the Library

Office Location: Ground Floor

Figure 3.3: Sonargaon University Room Info

3.3.4 Sonargaon UniversityTeachers Info

The Sonargaon UniversityTeachers Info document contains information about faculty members, including their academic specialization and research interests. It was included to test questions that require identifying suitable faculty members based on a particular academic or research topic.

SONARGAON UNIVERSITY
Department of Computer Science and Engineering (CSE)

Faculty Members: Field of Interest and Field of Specialisation

This document contains faculty member names, email addresses, fields of interest, and fields of specialisation associated with the Department of Computer Science and Engineering (CSE).

---

1. Prof. Bulbul Ahamed

Email:
- bulbul_cse@su.edu.bd
- bulbul2767@gmail.com

Field of Interest:
- Speech Recognition & Speech Processing
- Natural Language Processing (NLP)
- Deep Learning & Explainable AI (XAI)
- AI in Healthcare & Medical Image Analysis
- Cybersecurity & Information Safety
- Internet of Things (IoT) & Smart Systems
- Educational Technology & Generative AI

Field of Specialisation:
- Speech Recognition & NLP
- Artificial Intelligence
- Machine Learning
- Healthcare Informatics & Medical AI
- Cybersecurity & Privacy
- Green IoT & Blockchain Technology
- E-Commerce & Digital Systems

---

2. Mohammad Naderuzzaman

Email:
- nader_u@yahoo.com
- naderuzzaman.cse@su.edu.bd

Field of Interest:
- Software Testing

- Big Data Processing
- Artificial Intelligence
- Swarm Drone Technology

Field of Specialisation:
- Management
- Human Resource Management

---

3. Salma Tabashum

Email:
- salma.cse@su.edu.bd

Field of Interest:
- Artificial Intelligence (AI)
- Machine Learning
- Data Science

Field of Specialisation:
- Natural Language Processing (NLP)
- Deep Learning
- Data Science

---

4. Md. Shamim Hossain

Email:
- shamim.ice.iu@gmail.com
- shamim.cse@su.edu.bd

Field of Interest:
- Big Data Engineering
- Data Science
- Machine Learning
- Wireless Communication
- Cognitive Network

Field of Specialisation:
- Big Data Engineering
- Data Science
- Machine Learning

Figure 3.4: Raw Faculty Information Dataset

3.4 Dataset Statistics

The source documents were analyzed before being indexed into the knowledge base. The analysis considered the document size, number of pages, word count, sentence count, and number of extracted chunks. These statistics provide a measurable description of the dataset used in the experiment.

Document	File Size	Pages	Words	Sentences
Sonargaon University Information Book -1	219 KB	21	3,.361	222
Sonargaon University Information Book -2	78 KB	6	645	67
Sonargaon University Room Info	55 KB	4	264	19
Sonargaon University Teachers Info	125 KB	13	1,.103	54
Sonargaon University Data	344 KB	32	5,180	460
Total	821 KB	76	10,553	822

Table 3.2: Dataset Statistics

3.5 Raw Dataset Analysis

The raw documents were not initially stored in a form that could be directly used for semantic retrieval. The information was distributed across multiple pages and presented using paragraphs, headings, lists, tables, and other textual structures. Therefore, the documents had to be processed before they could be indexed into the vector database.

For example, the CSE Service Charter contains procedural information in a structured table, while the Department Layout Plan contains office names and room numbers. The faculty information contains names together with areas of specialization and research interests. These differences in structure were considered during the preparation of the knowledge base.

3.6 Text Extraction

The textual content of the source documents was extracted before creating the knowledge base. For PDF documents, the text was extracted programmatically so that it could be processed by the subsequent stages of the RAG pipeline.

The extracted text was checked to ensure that the main information contained in the source documents was available in machine-readable form. This step was necessary because the subsequent chunking and embedding stages operate on extracted text rather than directly on the visual representation of the document.

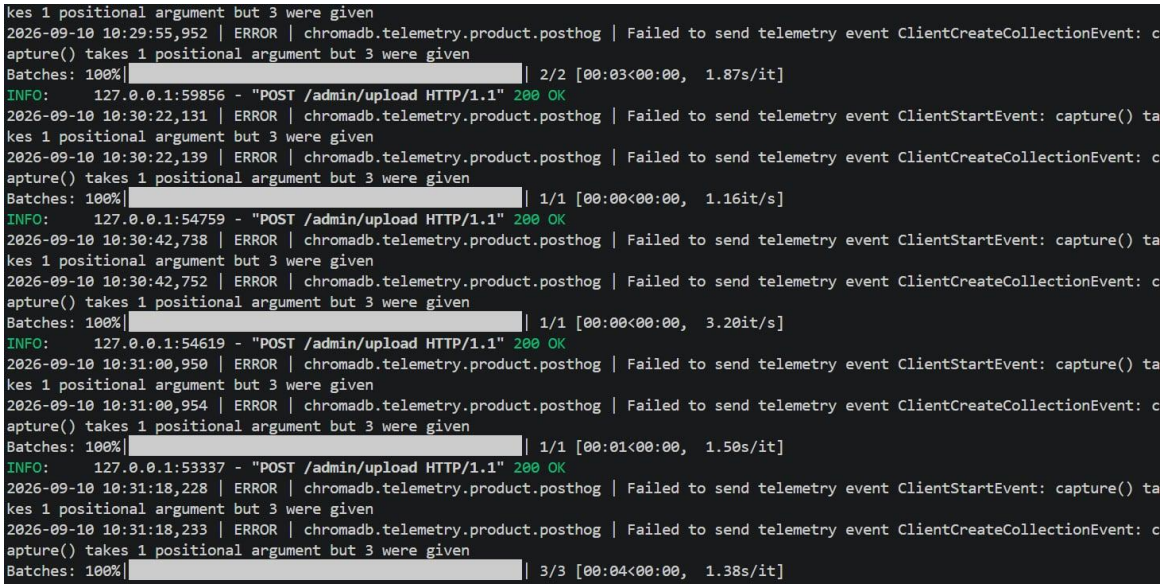


Figure 3.5: Example of Extracted Text from a Source Document

3.7 Chapter Summary

This chapter presented the documents used to construct the knowledge base and analyzed their content and structure. The dataset contains academic, administrative, location, and faculty-related information obtained from multiple source documents. The documents were examined in terms of their size, textual content, structure, and preparation requirements.

The extracted and prepared textual data forms the basis of the RAG pipeline. The next chapter presents the system architecture and methodology used to transform this prepared information into searchable vector representations and generate answers using the LLM.

CHAPTER 4
SYSTEM ARCHITECTURE AND METHODOLOGY

4.1 Introduction

This chapter presents the architecture and methodology used to design and develop the proposed system, titled **"An LLM-based RAG System for Question Answering over User-Provided Documents."**

The proposed system is designed to allow users to upload documents and interact with the information contained within those documents through a natural language chat interface. Instead of requiring users to manually search through large documents, the system processes the uploaded content, stores its semantic representations, retrieves relevant information based on a user's question, and generates an answer using a Large Language Model (LLM).

The system follows the **Retrieval-Augmented Generation (RAG)** approach. In this architecture, the language model does not answer a question solely based on its pretrained knowledge. Instead, the system first searches the uploaded documents for relevant information. The retrieved content is then provided to the LLM as context, allowing it to generate an answer grounded in the available documents.

The proposed system consists of two major parts:

1. **Frontend Application**
2. **Backend RAG Processing System**

The frontend provides a user-friendly interface where users can interact with the chatbot and view generated answers along with source information. The backend is responsible for document processing, text extraction, chunking, embedding generation, vector storage, semantic retrieval, context construction, and LLM-based answer generation.

The following sections describe each component of the proposed system and explain how these components work together to create a complete document-based question-answering application.

4.2 Overview of the Proposed System

The proposed system is an LLM-based document question-answering application built using a Retrieval-Augmented Generation architecture.

The main objective of the system is to enable users to obtain information from their uploaded documents through natural language questions.

A user can upload one or more supported documents to the system. After the documents are processed, their textual content is transformed into smaller segments and stored in a vector database. When the user submits a question, the system searches the vector database to identify the document segments that are most relevant to that question.

The retrieved information is then combined with the user's question and sent to a Large Language Model through an API. The LLM generates a response based on the retrieved context.

The overall operation of the system can be summarized as:

Document Upload → Text Processing → Chunking → Embedding Generation → Vector Storage → User Question → Semantic Retrieval → Context Construction → LLM → Generated Answer

The proposed system is designed to provide answers based primarily on the information contained in the uploaded documents. If the system cannot retrieve relevant information, it is designed to return an appropriate response indicating that the requested information could not be found in the available documents.

The system also maintains document metadata during processing. This allows the application to identify the original document associated with retrieved information and display source information to the user.

The major components of the proposed system are shown conceptually below:

User Interface Layer

- User interaction
- Question input
- Document upload functionality
- Display of generated answers
- Display of source information
- Loading state while the answer is being generated

Application and API Layer

- FastAPI backend
- API endpoints
- Request handling
- Chat service
- Document service
- Communication between frontend and backend

RAG Processing Layer

- Text extraction
- Document chunking
- Embedding generation
- Semantic search

- Context construction

Knowledge Storage Layer

- ChromaDB vector database
- Text chunks
- Vector embeddings
- Document metadata

Language Model Layer

- Groq API
- Large Language Model
- Context-based answer generation

4.3 Overall System Architecture

The proposed system follows a layered architecture in which different responsibilities are separated into individual components.

The overall architecture can be represented as follows:

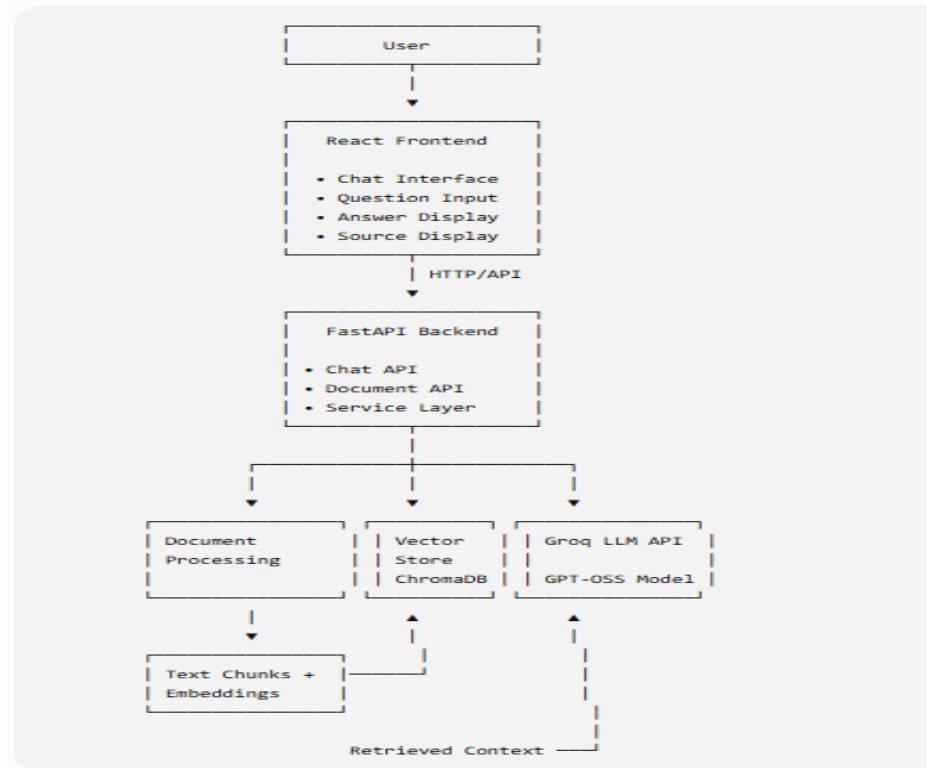


Figure 4.1: Architecture of User's Journey

The architecture separates the user interface from the processing logic. This separation makes the system easier to maintain and extend.

The frontend does not directly interact with the vector database or the LLM. Instead, it communicates with the FastAPI backend through API requests.

The backend acts as the central processing layer. It receives user requests, communicates with the vector store, retrieves relevant document chunks, constructs the required context, and sends the final prompt to the LLM.

This modular design also makes it possible to replace individual components in the future. For example, the LLM provider can be changed without requiring major changes to the frontend or vector retrieval system.

4.4 Document Upload and Processing

The document upload component allows users to provide documents that will serve as the knowledge source for the RAG system.

When a document is uploaded, the backend receives the file through the document upload API. The document is then passed to the document processing service.

The main purpose of this stage is to convert the uploaded file into processable textual information.

The general workflow is:

User Uploads Document → Backend Receives File → File Validation → Text Processing → Chunking → Embedding → Storage

During this process, relevant metadata associated with the document is maintained. Metadata may include information such as:

- Original filename
- File-related information
- Chunk identification
- Source information

Maintaining the original filename is particularly important because it enables the system to identify the source of retrieved information later.

After processing, the document is transformed into a collection of smaller text chunks. These chunks are then converted into vector embeddings and stored in ChromaDB.

Once the processing is complete, the uploaded document becomes part of the application's searchable knowledge base.

4.5 Text Extraction

Text extraction is the process of obtaining readable textual content from an uploaded document.

Documents are generally stored in file formats that cannot be directly used for semantic search or embedding generation. Therefore, the content of the document must first be extracted and converted into plain text.

The extracted text becomes the primary knowledge source for the RAG pipeline.

The quality of text extraction is important because the later stages of the system depend on the extracted content. If the relevant information is not successfully extracted, the retrieval system will not be able to retrieve it.

After extraction, the text is prepared for further processing.

The general process can be represented as:

Uploaded Document → Text Extraction → Cleaned Text → Chunking

The extracted content is not directly sent to the Large Language Model as an entire document. Instead, it is divided into smaller chunks to improve retrieval efficiency and relevance.

4.6 Document Chunking

Documents may contain thousands of words and multiple pages of information. Sending an entire document to the LLM every time a user asks a question would be inefficient and could introduce unnecessary information into the model context.

To address this problem, the extracted document text is divided into smaller segments known as **chunks**.

The chunking process provides several advantages:

- More efficient retrieval
- More focused context
- Reduced unnecessary information
- Improved semantic similarity search
- Better utilization of the LLM context window

For example, a document containing information about admission procedures, course fees, examination rules, and scholarship policies can be divided into multiple smaller sections.

If the user asks:

"What is the scholarship policy?"

the retrieval system does not need to send the entire document to the LLM. Instead, it searches for the chunks most closely related to scholarships.

The relevant chunks are then selected and used as contextual information for answer generation.

Chunking is therefore an important stage in the RAG pipeline because it connects large document content with efficient semantic retrieval.

Each chunk is stored along with relevant metadata so that the original source document can later be identified.

4.7 Text Embedding Generation

After document chunking, each text chunk is converted into a numerical representation called an **embedding**.

An embedding represents the semantic meaning of text in a multidimensional vector space.

The proposed system uses the following embedding model:

sentence-transformers/all-MiniLM-L6-v2

This model converts textual information into vector representations that can be compared based on semantic similarity.

For example, the following two questions may use different words:

"What is the admission fee?"

and

"How much does enrollment cost?"

However, both questions have similar meanings. An embedding model can represent semantically related text in nearby locations within the vector space.

The embedding generation process can be represented as:

Text Chunk → Embedding Model → Numerical Vector

Similarly, when a user asks a question:

User Question → Embedding Model → Query Vector

The query vector can then be compared with the stored document vectors to identify the most relevant information.

The use of semantic embeddings allows the system to retrieve information based on meaning rather than depending entirely on exact keyword matching.

4.8 Vector Storage Using ChromaDB

After generating embeddings, the document chunks and their corresponding vector representations are stored in a vector database.

The proposed system uses **ChromaDB** as the vector storage and semantic retrieval system.

Each stored entry may contain:

- Unique chunk identification
- Original text chunk
- Vector embedding
- Metadata
- Original document information

The structure can be conceptually represented as:

Table 4.8.1: Document Chunk Storage Structure in ChromaDB

Chunk ID	Text Content	Embedding	Metadata
Chunk 001	Admission information...	Vector representation	Filename
Chunk 002	Scholarship information...	Vector representation	Filename
Chunk 003	Examination rules...	Vector representation	Filename

Table 4.1: Document Chunk Storage Structure in ChromaDB

ChromaDB enables the application to search through the stored document chunks based on semantic similarity.

When the user submits a question, the system performs a similarity search to find the chunks most closely related to the meaning of that question.

The retrieved chunks are then passed to the next stage of the RAG pipeline.

Using a vector database allows the application to efficiently manage document knowledge and perform semantic retrieval without requiring the entire document collection to be sent to the LLM.

4.9 Query Processing and Semantic Retrieval

When a user submits a question through the frontend interface, the question is sent to the backend through the chat API.

The backend passes the question to the retrieval service.

The query is processed using the same semantic embedding approach used for document chunks.

The process can be represented as:

User Question → Query Embedding → Vector Similarity Search → Relevant Chunks

The proposed system retrieves the top relevant chunks from ChromaDB. In the current implementation, the retrieval process searches for a limited number of the most relevant chunks rather than returning the entire knowledge base.

This approach improves efficiency and helps ensure that the LLM receives focused contextual information.

For example, if a user has uploaded several documents containing thousands of text chunks and asks:

"What are the requirements for course registration?"

the system converts this question into an embedding and searches the vector database.

The retrieval system may return the chunks containing information related to:

- Registration procedures
- Required documents
- Academic requirements
- Registration deadlines

Only the most relevant chunks are selected for the answer generation process.

If no relevant information is found, the system can return a predefined response indicating that the requested information could not be located in the uploaded documents.

4.10 Context Construction

After semantic retrieval, the system receives a collection of relevant document chunks.

These chunks are combined to create the **context** that will be provided to the Large Language Model.

The context construction process can be represented as:

Retrieved Chunk 1

-

Retrieved Chunk 2

-

Retrieved Chunk 3

↓

Final Context

The user's original question is then combined with this context.

A simplified structure is shown below:

The purpose of context construction is to provide the LLM with the specific information needed to answer the user's question.

Instead of asking the model to rely entirely on its general knowledge, the retrieved document content acts as an external knowledge source.

This approach helps keep the generated response relevant to the uploaded documents.

4.11 Answer Generation Using a Large Language Model

After the context is constructed, the backend sends the user's question and retrieved document information to the Large Language Model.

The proposed system uses an LLM accessed through the **Groq API**.

The system architecture separates the LLM service from other application components. This allows the model provider or model name to be modified through configuration without requiring major changes to the overall application structure.

The current implementation uses a Groq-hosted model configured through the application's settings.

The LLM receives:

1. System instructions
2. Retrieved document context
3. User question

The model is instructed to generate an answer based on the provided context.

The prompt also includes rules to improve response quality. These include instructions such as:

- Answer only using the provided context.
- Do not generate unsupported information.
- Return a specific message if the answer is not available.
- Use structured formatting when appropriate.
- Use numbered lists for multiple steps.
- Use bullet points for requirements.
- Keep answers concise and readable.

This approach helps the system generate responses that are more structured and grounded in the uploaded documents.

The generated answer is returned to the backend and then sent to the frontend for display.

4.12 Source Identification

An important feature of the proposed system is source identification.

During document processing, metadata related to the original document is stored along with the text chunks.

When the retrieval system selects relevant chunks, the associated metadata is also available.

The backend collects the source document names from the retrieved results.

The general process can be represented as:

Retrieved Chunk → Metadata → Original Filename

If multiple chunks come from the same document, the system can remove duplicate source names before returning the result.

The frontend can then display the document source associated with the generated answer.

For example:

Answer:
The course registration must be completed before the specified deadline.

Sources:

- Academic Regulations.pdf
- Student Guidelines.pdf

Source identification improves transparency by showing users which uploaded documents contributed to the generated answer.

This feature also helps distinguish the proposed application from a general chatbot that generates responses without providing information about the origin of its answers.

4.13 Frontend and Backend Communication

The proposed system uses a separate frontend and backend architecture.

The frontend is developed using **React**, while the backend is developed using **Python and FastAPI**.

Communication between the two components takes place through HTTP-based API requests.

The frontend uses **Axios** to communicate with the backend.

The general communication process is:

React Frontend → Axios Request → FastAPI API → RAG Processing → JSON Response → React Frontend

For chat interaction, the frontend sends the user's question to the backend.

A conceptual request structure is:

```
{
 "question": "What information is available about the admission process?"
}
```

The backend processes the request and performs semantic retrieval and answer generation.

The response may contain:

```
{
 "answer": "The admission process requires...",
 "sources": [
   "document1.pdf"
 ]
}
```

The frontend receives the response and displays:

- The generated answer
- Source document information

During answer generation, the frontend also displays a loading animation to indicate that the system is processing the user's request.

This improves the user experience by providing feedback while the backend communicates with the LLM.

The separation between frontend and backend also improves maintainability. The user interface can be modified independently without requiring changes to the core RAG processing pipeline.

4.14 Complete System Workflow

The complete workflow of the proposed system consists of two major processes:

1. **Document Knowledge Base Creation**
2. **Question Answering Process**

4.14.1 Document Knowledge Base Creation

The process begins when the user uploads a document.

The system follows these steps:

Step 1: Document Upload

The user uploads a document through the application.

Step 2: Document Processing

The backend receives the uploaded document and prepares it for processing.

Step 3: Text Extraction

Textual content is extracted from the uploaded document.

Step 4: Document Chunking

The extracted text is divided into smaller segments.

Step 5: Embedding Generation

Each chunk is converted into a semantic vector using the embedding model.

Step 6: Vector Storage

The text chunks, embeddings, and metadata are stored in ChromaDB.

After this process is completed, the uploaded document becomes part of the searchable knowledge base.

4.14.2 Question Answering Process

Once documents have been processed, users can ask questions through the chat interface.

The question-answering workflow is as follows:

Step 1: User Submits a Question

The user enters a natural-language question into the frontend chat interface.

Step 2: API Request

The frontend sends the question to the FastAPI backend.

Step 3: Query Processing

The question is processed and converted into a semantic representation.

Step 4: Semantic Retrieval

The system searches ChromaDB to identify the most relevant document chunks.

Step 5: Context Construction

The retrieved chunks are combined to create contextual information.

Step 6: LLM Processing

The context and the user's question are sent to the configured Large Language Model through the Groq API.

Step 7: Answer Generation

The LLM generates an answer based on the provided document context.

Step 8: Source Identification

The backend collects the filenames associated with the retrieved chunks.

Step 9: Response to Frontend

The generated answer and source information are returned to the React frontend.

Step 10: Result Display

The frontend displays the answer to the user along with the relevant source information

The complete workflow can be summarized as:

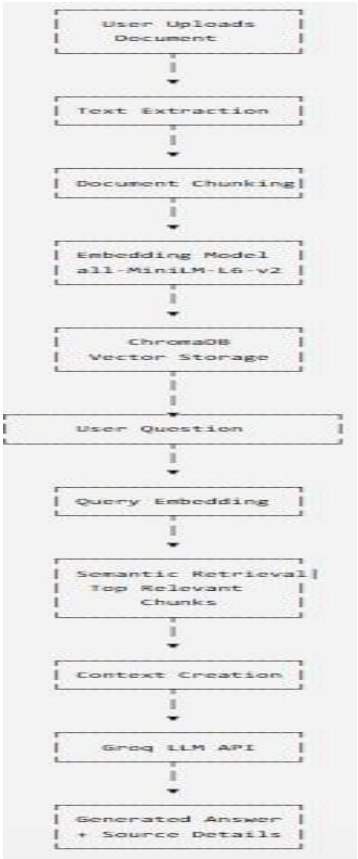


Figure 4.2: Question Answering Process

4.15 Chapter Summary

This chapter presented the architecture and methodology of the proposed **LLM-based RAG system for question answering over user-provided documents**.

The system follows a Retrieval-Augmented Generation architecture in which uploaded documents are processed and transformed into searchable knowledge. The document processing pipeline includes text extraction, chunking, embedding generation, and vector storage using ChromaDB.

When a user asks a question, the system performs semantic retrieval to identify the most relevant document chunks. These chunks are combined into a context and sent to a Large Language Model through the Groq API. The generated answer is then returned to the user together with relevant source information.

The system uses a modular architecture consisting of a React frontend, FastAPI backend, embedding model, ChromaDB vector database, and an externally accessed LLM. This design allows individual components to be modified or replaced while maintaining the overall structure of the application.

The methodology described in this chapter provides the technical foundation for the implemenaztation and evaluation of the proposed system. The following chapter can present the **system implementation, interface design, testing process, results, and discussion**.

CHAPTER 5
SYSTEM IMPLEMENTATION

5.1 Introduction

This chapter describes the implementation of the proposed Large Language Model (LLM)-based Retrieval-Augmented Generation (RAG) system for question answering over user-provided documents. The system was developed as a web-based application that allows users to provide documents, retrieve relevant information from those documents, and ask questions through a conversational interface.

The implementation follows the architecture and methodology discussed in Chapter 4. Instead of directly asking a Large Language Model to answer questions from its general knowledge, the system first processes the user-provided documents and stores their semantic representations in a vector database. When a user submits a question, the system searches the stored document representations to identify relevant information. The retrieved information is then provided to the LLM as context so that the generated answer is grounded in the uploaded documents.

The system consists of two major application layers: a frontend interface and a backend processing system. The frontend provides the user interface for interacting with the application, while the backend manages document processing, vector storage, semantic retrieval, prompt construction, and communication with the LLM service.

The final implementation uses React for the frontend, FastAPI for the backend, ChromaDB for vector storage, Sentence Transformers for embedding generation, and the Groq API for LLM-based answer generation. The LLM used in the final implementation is the openai/gpt-oss-120b model accessed through Groq.

The implementation was designed to remain modular so that individual components, such as the LLM provider or vector database, can be replaced or extended in future versions without redesigning the entire application.

5.2 Development Environment

The proposed system was developed using a modern web application development environment. The backend and frontend were maintained as separate application components to provide clear separation between the user interface and the RAG processing pipeline.

The major development components are:

Programming Language: Python

Frontend Framework: React

Backend Framework: FastAPI

Vector Database: ChromaDB

Embedding Model: sentence-transformers/all-MiniLM-L6-v2

LLM Provider: Groq API

LLM Model: openai/gpt-oss-120b

HTTP Client: Axios

Development Server: Uvicorn

Development Environment: Visual Studio Code

Document Processing: PyMuPDF and Python-based document processing libraries

Source Code Management: Git and GitHub

The backend was developed inside a Python virtual environment to isolate project dependencies from the system-level Python installation. The frontend and backend were developed independently and communicate through HTTP-based REST API requests.

5.3 Technology Stack

The complete technology stack of the implemented system can be divided into several categories.

Table 5.3.1: Technology Stack of the Proposed System

Component	Technology Used	Purpose
Frontend	React	User interface and chat interaction
Backend	FastAPI	API development and application logic
Server	Uvicorn	Running the FastAPI application
Programming Language	Python	Backend and RAG implementation
Vector Database	ChromaDB	Storing and retrieving document embeddings
Embedding Model	all-MiniLM-L6-v2	Converting text into vector representations
LLM Provider	Groq	Providing LLM inference
LLM	openai/gpt-oss-120b	Generating answers
HTTP Communication	Axios	Frontend-backend communication
Document Processing	PyMuPDF / Python libraries	Extracting text from documents
Version Control	Git / GitHub	Source code management

Table 5.1: Technology Stack of the Proposed System

Each technology was selected according to the functional requirements of the proposed system. The frontend requires a responsive interactive environment, while the backend requires efficient API handling and integration with the RAG components.

5.4 Frontend Implementation

The frontend of the application was implemented using React. Its primary responsibility is to provide a simple and accessible interface through which users can interact with the document-based question answering system.

The interface was intentionally designed to focus on the main functionality of the application rather than providing unnecessary features. The user can access the chat interface directly and submit questions to the system.

The frontend contains several functional components, including:

1. Chat interface
2. Message display
3. User input component
4. Loading indicator
5. Backend connection handling
6. API communication
7. Document upload interaction

The React application uses reusable components to separate different interface responsibilities. For example, the chat window is responsible for displaying conversation messages, while the input component handles user questions and submission.

The frontend communicates with the backend using HTTP requests. Axios is used as the HTTP client, allowing the React application to send questions and receive responses from the FastAPI backend.

A centralized API configuration was also implemented so that the frontend can communicate with the backend through a configurable API base URL.

5.5 Backend Architecture

The backend was implemented using the FastAPI framework in Python. FastAPI was selected because it provides a lightweight and structured environment for developing REST APIs and can easily integrate with Python-based machine learning and document-processing libraries.

The backend acts as the central processing layer of the application. It receives user requests from the frontend, performs document retrieval, communicates with the LLM, and returns the generated response.

The backend follows a service-oriented structure in which different responsibilities are separated into individual services.

The major backend components include:

- Configuration management
- Document service
- Document processing service
- Document indexing service

- Vector store service
- Chat service
- Groq service
- API routing

This separation reduces dependency between different parts of the application and makes the system easier to maintain and extend.

The main flow of a question through the backend is:

1. The frontend sends the user's question.

2. The FastAPI backend receives the request.

3. The chat service processes the question.

4. ChromaDB performs semantic retrieval.

5. Relevant document chunks are collected.

6. The retrieved chunks are passed to the Groq service.

7. A prompt containing the context and question is constructed.

8. The LLM generates the answer.

9. The backend returns the answer and source information to the frontend.

5.6 Document Processing Implementation

The document upload functionality allows users to provide documents that will become the knowledge source for the RAG system.

The backend exposes a document upload API endpoint that accepts files through an HTTP multipart request.

The upload process begins when the user selects a document through the frontend interface. The selected file is transmitted to the backend using a FormData request.

The backend receives the uploaded file through FastAPI's file upload mechanism. The document service then stores and processes the uploaded document.

The document upload process can be summarized as:

1. User selects a document.

2. Frontend creates a multipart form request.

3. File is sent to the FastAPI backend.

4. Backend receives the file.

5. Document metadata is collected.

6. Document content is processed.

7. Extracted text is divided into chunks.

8. Embeddings are generated.

9. Embeddings and associated information are stored in ChromaDB.

This process transforms an ordinary uploaded document into searchable information that can later be used to answer user questions.

After receiving a document, the system extracts its textual content before performing semantic processing.

For PDF documents, a PDF processing library is used to access the document pages and extract textual information.

The extraction process is important because the RAG pipeline operates on textual information rather than directly reasoning over the original PDF file.

The extracted content is collected from the document and passed to the subsequent processing stage.

The basic process is:

PDF Document → Page Content → Extracted Text → Text Processing

The extracted text may contain unnecessary whitespace or formatting artifacts. Therefore, the implementation performs basic cleaning before the text is passed to the chunking stage.

The purpose of this stage is not to generate an answer but to transform the uploaded document into a textual representation suitable for semantic indexing.

Large documents cannot efficiently be provided to an LLM as a single context block. Therefore, the extracted document text is divided into smaller segments known as chunks.

The chunking process is an important component of the RAG system because retrieval operates on these individual chunks.

Instead of searching an entire document as one large unit, the system searches smaller sections of the document and identifies the portions that are semantically related to the user's question.

The chunking process can be represented as:

Extracted Document Text → Text Splitter → Multiple Text Chunks

For example, a document containing information about university admission may be divided into chunks containing information about:

1. Admission requirements
2. Application procedure
3. Tuition fees
4. Scholarship policies
5. Department information

When a user asks a question about tuition fees, the retrieval system can identify the chunks that contain information related to fees rather than processing the entire document.

The chunking strategy therefore improves retrieval efficiency and allows the LLM to receive more relevant context.

5.7 Embedding and Vector Database Implementation

After the document is divided into chunks, each chunk is converted into a numerical vector representation known as an embedding.

The implementation uses the sentence-transformers/all-MiniLM-L6-v2 model for generating embeddings.

An embedding represents the semantic meaning of text in a numerical form. Texts with similar meanings tend to produce vector representations that are closer to one another in the embedding space.

For example, the following questions have different wording but similar meaning:

"What is the tuition fee?"

"How much does the program cost?"

A semantic embedding model can represent these questions in a way that allows the retrieval system to recognize their similarity.

The embedding generation process is:

Text Chunk → Sentence Transformer → Numerical Vector

The generated vector is stored together with the original chunk so that the original information can be retrieved when a relevant query is submitted.

ChromaDB is used as the vector database of the system.

Traditional databases primarily perform searches based on exact values or keywords. However, the proposed RAG system requires semantic similarity search. Therefore, the document embeddings are stored in a vector database.

ChromaDB stores the vector representations of document chunks along with associated metadata.

Each indexed chunk can be associated with information such as:

- Chunk identifier
- Original filename
- Document metadata
- Extracted content
- Vector representation

The vector database allows the system to compare a user's question embedding against stored document embeddings.

The general process is:

Document Chunk → Embedding → ChromaDB

and during question answering:

Question → Query Embedding → ChromaDB → Relevant Chunks

This provides the retrieval mechanism required by the RAG architecture.

5.8 Semantic Retrieval Implementation

Semantic search is performed when the user submits a question.

The user's question is first converted into an embedding using the same embedding model used for the document chunks. This ensures that both the query and document chunks are represented within the same vector space.

The query embedding is then compared with the vectors stored in ChromaDB.

The implementation retrieves the most relevant document chunks according to their semantic similarity. In the implemented system, the chat service requests the top relevant chunks from the vector store.

The retrieval process can be represented as:

1. User submits a question.

2. Question is converted into an embedding.

3. ChromaDB compares the query vector with stored vectors.

4. The most relevant chunks are selected.

5. Selected chunks are returned to the chat service.

The retrieved chunks form the evidence that is provided to the LLM.

This is one of the most important characteristics of the system because it reduces the dependency on the LLM's pretrained knowledge and allows the application to answer questions based on user-provided documents.

5.9 LLM Integration

The answer generation component was implemented using the Groq API.

During the development of the project, Gemini was initially considered for LLM-based generation. However, limitations related to API quota and request availability made it unsuitable for the final implementation. The system was therefore migrated to Groq for LLM inference.

The final implementation uses:

Groq API → openai/gpt-oss-120b

The Groq service is implemented as a separate backend service. This keeps the LLM-specific implementation isolated from the rest of the application.

The Groq client is initialized using an API key stored in the application's configuration rather than being directly embedded into the application logic.

When the RAG pipeline retrieves relevant document chunks, the Groq service receives:

- User question
- Retrieved document context

The service then constructs a prompt and sends it to the selected LLM.

The generated response is returned to the chat service and subsequently sent to the frontend.

5.10 Prompt and Context Construction

Prompt construction is an important part of the implemented RAG system.

The system does not simply send the user's question directly to the LLM. Instead, the retrieved document chunks are inserted into a structured prompt.

The prompt instructs the LLM to answer using the provided context.

The general prompt structure is:

System Instructions

↓

Retrieved Document Context

↓

User Question

↓

Generated Answer

The implemented prompt contains instructions that encourage the model to:

- Answer using the provided context.
- Avoid relying on unrelated external knowledge.
- State when the required information cannot be found.
- Present multi-step answers using numbered lists.
- Present requirements using bullet points.
- Keep responses concise and readable.

The system also defines a fallback response:

"I couldn't find that information in the uploaded documents."

This behavior is useful because the system is intended to answer questions about user-provided documents rather than act as a general-purpose chatbot.

The overall generation process can therefore be expressed as:

Question + Retrieved Context → Prompt → LLM → Answer

5.11 Source Attribution

In addition to generating an answer, the backend keeps track of the documents from which the retrieved information originated.

Each retrieved result contains metadata associated with the original document. The chat service extracts the original filename from the metadata of the retrieved chunks.

The system collects the unique filenames associated with the retrieved results and returns them as source information.

This allows the frontend to provide information about the document that contributed to the answer.

For example, if the user uploads:

- CSE_Admission_Guide.pdf
- Tuition_Fee_Structure.pdf
- Scholarship_Policy.pdf

and asks:

"What is the scholarship eligibility requirement?"

the retrieval process may identify chunks from Scholarship_Policy.pdf.

The application can then associate the generated response with that document.

Source identification improves transparency because the user can understand where the retrieved information originated.

5.12 API Integration

The frontend and backend communicate using REST-style HTTP APIs.

The React frontend uses Axios to send requests to the FastAPI backend.

The application defines a centralized HTTP client configuration containing the backend API base URL. This allows the backend address to be changed without modifying every individual API request.

The major interactions include:

Health Check

The frontend sends a request to the backend health endpoint to determine whether the backend service is available.

Chat Request

The frontend sends the user's question to the chat endpoint.

The request contains the question in JSON format.

Conceptually:

POST /api/v1/chat

{

"question": "User question"

}

The backend processes the question through the RAG pipeline and returns the generated answer.

Document Upload

Documents are submitted through the document upload endpoint using multipart form data.

Conceptually:

POST /api/v1/documents/upload

The backend receives the document and processes it for retrieval.

The overall communication process is:

React Frontend

↓

HTTP Request

↓

FastAPI Backend

↓

RAG Pipeline

↓

LLM

↓

FastAPI Response

↓

React Frontend

5.13 Error Handling

Error handling was incorporated into different layers of the application to prevent individual failures from crashing the entire system without meaningful feedback.

The backend validates important configuration values before initializing services. For example, the Groq service verifies that the required API key is available before creating the API client.

The LLM service also checks whether a generated response is empty. If the LLM does not return valid content, the backend raises an appropriate runtime error.

Similarly, the chat service handles the situation where no relevant document chunks are retrieved. Instead of sending an empty context to the LLM, the system returns a predefined response indicating that the requested information could not be found.

The frontend also displays a loading indicator while the LLM is generating a response.

These mechanisms improve the reliability and usability of the application.

5.14 User Interface Implementation

The user interface was designed as a minimal document-question-answering environment.

The main interface focuses on the conversation between the user and the AI assistant.

The interface contains:

- Application header
- Chat area
- User messages
- Assistant responses
- Message timestamps
- Loading animation
- Question input field
- Send button
- Source/document information

The interface was intentionally kept simple so that the user can begin asking questions without navigating through unnecessary screens.

When a question is submitted, the user's message is immediately displayed in the conversation area. While the backend processes the request and the LLM generates the answer, a loading animation is displayed.

After the response is received, the generated answer is added to the conversation.

The frontend also supports structured rendering of responses. For example, multi-step answers can be displayed as numbered items rather than as a single large paragraph.

This improves readability, particularly when the retrieved information contains procedures, requirements, or multiple conditions.

5.15 Complete Implementation Workflow

The complete implementation of the proposed system can be summarized through two primary workflows: document indexing and question answering.

Document Indexing Workflow

When a user provides a document, the following sequence occurs:

1. The user selects a document.
2. The React frontend sends the document to FastAPI.
3. The backend receives the uploaded file.
4. The document processing service extracts the text.
5. The extracted text is cleaned and processed.
6. The text is divided into smaller chunks.
7. Sentence Transformers generate embeddings for each chunk.
8. The embeddings are stored in ChromaDB.
9. Metadata such as the original filename is stored with the chunks.
10. The document becomes available for semantic

retrieval.

The workflow can be represented as:

User Document

↓

Frontend Upload

↓

FastAPI Backend

↓

Text Extraction

↓

Text Chunking

↓

Embedding Generation

↓

ChromaDB

↓

Indexed Knowledge Base

Question Answering Workflow

After documents have been indexed, the user can ask questions.

The question answering process is:

1. User enters a question.
2. React sends the question to the FastAPI backend.
3. Chat service receives the question.
4. The vector store performs semantic search.
5. The most relevant document chunks are retrieved.
6. Retrieved chunks are combined into a context block.
7. The context and question are inserted into the LLM prompt.
8. The prompt is sent to the Groq API.
9. The openai/gpt-oss-120b model generates the answer.
10. The backend identifies the source document metadata.
11. The answer and source information are returned to the frontend.
12. The frontend displays the answer to the user.

The complete workflow can be represented as:

User Question

↓

React Frontend

↓

FastAPI Chat API

↓

Query Embedding

↓

ChromaDB Semantic Search

↓

Relevant Document Chunks

↓

Context Construction

↓

Groq API

↓

openai/gpt-oss-120b

↓

Generated Answer

↓

Source Identification

↓

FastAPI Response

↓

React Chat Interface

5.16 Chapter Summary

This chapter presented the implementation of the proposed LLM-based Retrieval-Augmented Generation system for question answering over user-provided documents.

The system was implemented using React as the frontend framework and FastAPI as the backend framework. Documents uploaded by users are processed through text extraction and chunking before being converted into vector representations using the sentence-transformers/all-MiniLM-L6-v2 embedding model. The resulting embeddings are stored in ChromaDB for semantic retrieval.

When a user submits a question, the system searches the vector database for relevant document chunks. These chunks are then combined with the user's question to construct a context-aware prompt. The prompt is sent to the openai/gpt-oss-120b Large Language Model through the Groq API. The generated response is returned to the frontend together with source information associated with the retrieved documents.

The implemented system therefore provides an end-to-end pipeline from document upload to context retrieval and LLM-based answer generation. The modular implementation also provides a foundation for future improvements such as persistent database management, improved retrieval strategies, authentication, document management, evaluation metrics, and deployment as a production web application.

CHAPTER 6
RESULTS AND DISCUSSION

6.1 Introduction

This chapter presents the results obtained from the implementation and testing of the proposed Large Language Model (LLM)-based Retrieval-Augmented Generation (RAG) system. The primary objective of the evaluation was to determine whether the developed application could successfully process user-provided documents, retrieve relevant information from those documents, and generate meaningful answers to user questions.

The system was evaluated from several perspectives, including document processing, semantic retrieval, question answering, source identification, and overall usability. Rather than evaluating the Large Language Model independently, the evaluation focused on the complete RAG pipeline because the main purpose of the proposed system is to combine document retrieval with LLM-based generation.

The testing was conducted using documents containing structured and unstructured textual information. Different types of questions were submitted to the system, including direct factual questions, questions requiring information from multiple portions of a document, and questions for which the required information was not available in the uploaded documents.

The results demonstrate that the implemented system can establish an end-to-end workflow from document upload to context retrieval and answer generation. However, the system also has limitations related to retrieval quality, document complexity, external API dependency, and the absence of advanced document management functionality.

6.2 Experimental Environment

The experimental environment consisted of the frontend application, backend application, vector database, embedding model, and external LLM inference service.

The frontend was executed as a React-based web application, while the backend was executed using FastAPI and Uvicorn. The backend was responsible for document processing, embedding generation, semantic retrieval, prompt construction, and communication with the LLM.

The embedding generation stage used the sentence-transformers/all-MiniLM-L6-v2 model. ChromaDB was used as the vector storage and semantic retrieval layer.

For answer generation, the system used the Groq API with the openai/gpt-oss-120b model.

Component	Configuration
Frontend	React
Backend	FastAPI
Backend Server	Uvicorn
Programming Language	Python
Embedding Model	sentence-transformers/all-MiniLM-L6-v2
Vector Database	ChromaDB
LLM Provider	Groq API
LLM Model	openai/gpt-oss-120b
API Communication	REST/HTTP
Frontend HTTP Client	Axios
Development Environment	Visual Studio Code
Version Control	Git

Table 6.1: Experimental Environment

The system was tested locally during development. This environment was sufficient for validating the complete application workflow and identifying implementation-level issues before possible production deployment.

6.3 Document Upload Results

The document upload functionality was successfully integrated into the application.

When a document is uploaded, the frontend sends the file to the backend using a multipart HTTP request. The backend receives the file and passes it through the document processing pipeline.

The document processing workflow successfully performs the following operations:

1. Receives the uploaded document.
2. Extracts textual information.
3. Divides the extracted text into smaller chunks.
4. Generates embeddings for the chunks.
5. Stores the embeddings and associated metadata in ChromaDB.

This process converts the uploaded document into a searchable knowledge source.

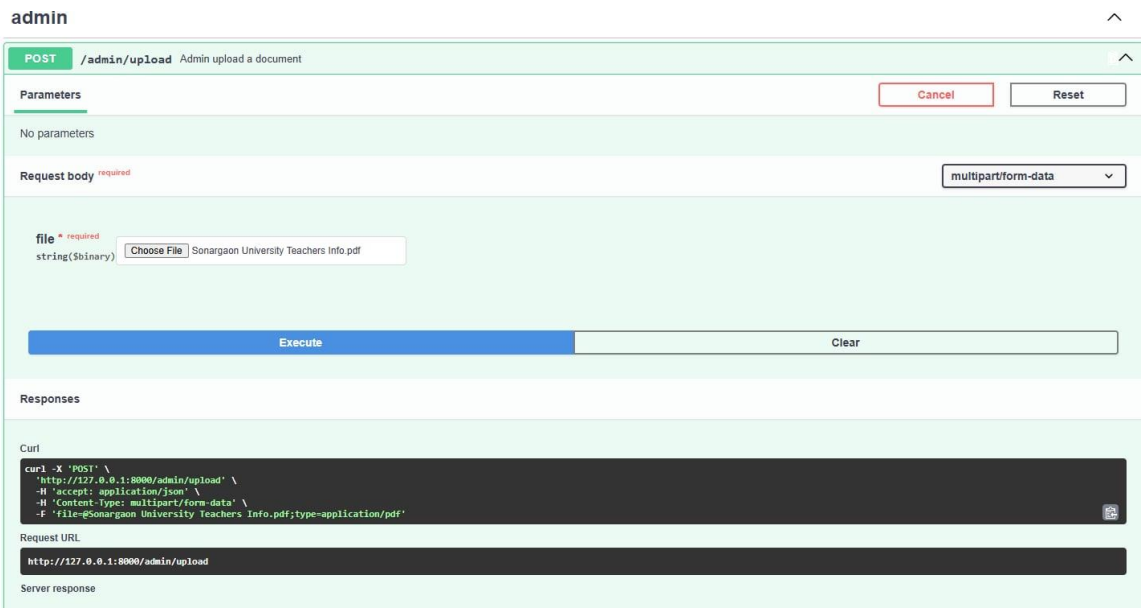


Figure 6.1: Document uploading process from the admin side

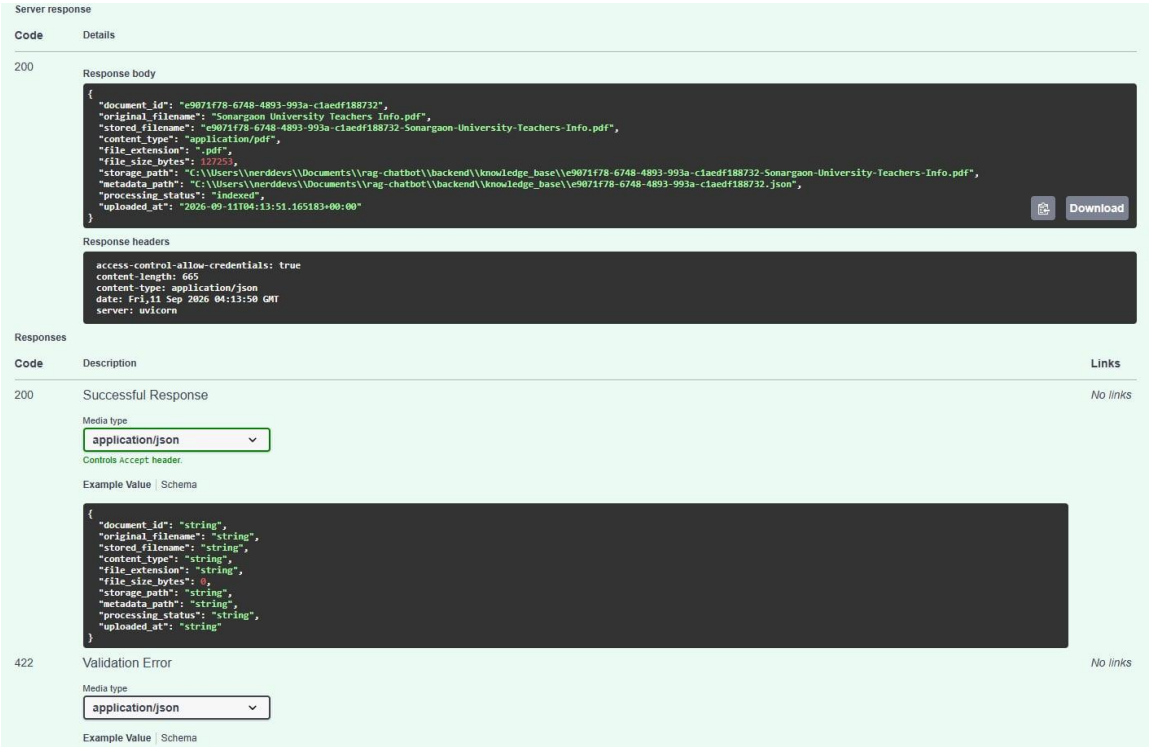


Figure 6.2: Document uploading process from the admin side

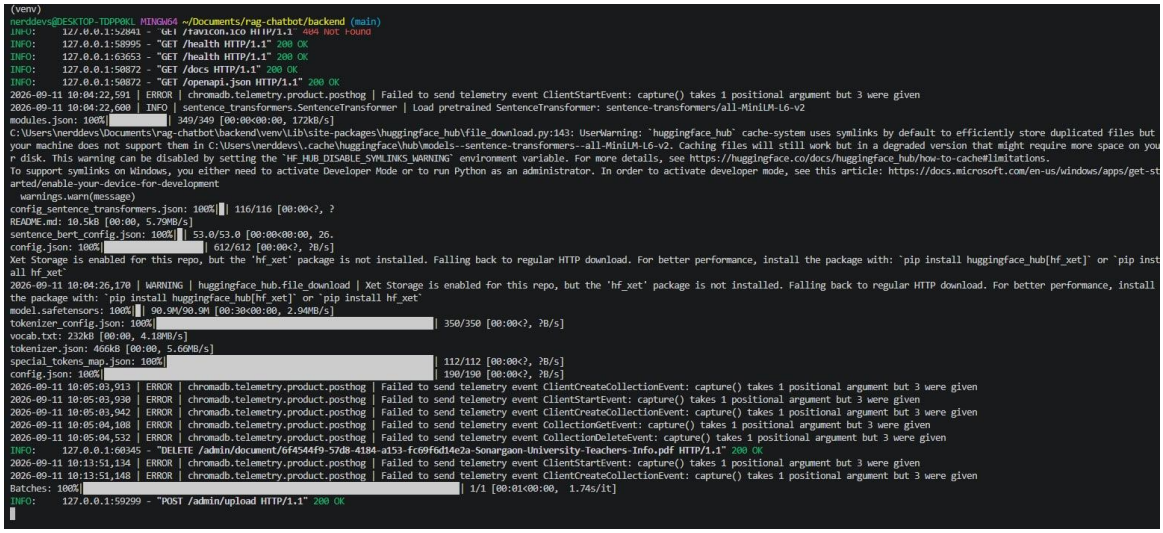


Figure 6.3: Backend Server log of document upload

6.4 Question Answering Results

The question answering functionality was evaluated by submitting questions related to information contained within uploaded documents.

The implemented RAG pipeline follows the sequence:

Question → Semantic Retrieval → Relevant Context → LLM → Answer

When a question is submitted, the system does not immediately send the question to the LLM. Instead, the question is first processed by the retrieval component.

The retrieval component searches ChromaDB for document chunks that are semantically related to the question. The retrieved chunks are then provided to the LLM as context.

The LLM generates an answer using the retrieved information.

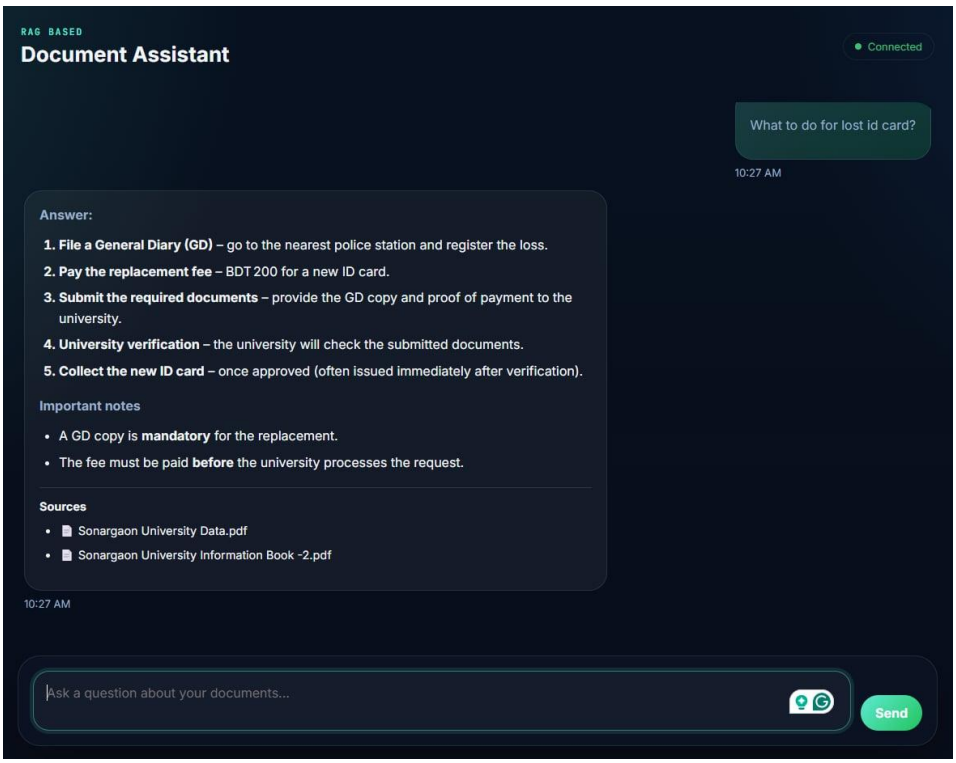


Figure 6.4: Question Answering Result

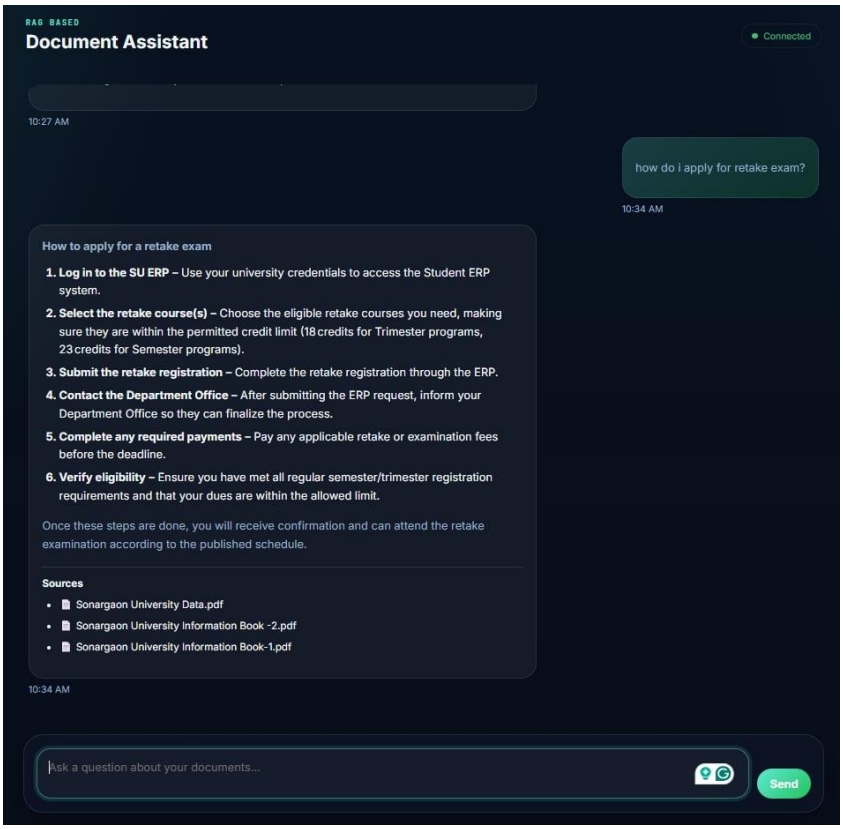


Figure 6.5: Question Answering Result

The implemented system also maintains source information associated with retrieved document chunks.

Each indexed chunk contains metadata related to its original document. When relevant chunks are retrieved, the chat service extracts the original filename from the metadata.

The application can therefore determine which uploaded document contributed to the retrieved context.

For example, if the user uploads:

- CSE_Course_Outline.pdf
- University_Admission_Guide.pdf
- Scholarship_Policy.pdf

and asks a question regarding scholarship requirements, the retrieval system may identify chunks originating from Scholarship_Policy.pdf.

The source information can then be displayed alongside the generated answer.

6.5 Semantic Retrieval Results

Semantic retrieval is one of the core components of the proposed system.

The evaluation demonstrated that the system can retrieve information based on semantic similarity rather than requiring the user to use exactly the same words that appear in the document.

For example, a document may contain:

"The total tuition cost of the program is..."

while a user may ask:

"How much does the program cost?"

Although the wording is different, the embedding model can represent both expressions

in a semantically related manner.

The semantic retrieval process can therefore be summarized as:

User Question

↓

Query Embedding

↓

Vector Similarity Search

↓

Top Relevant Chunks

↓

Context Construction

↓

LLM

The results indicate that semantic retrieval provides the necessary connection between natural-language user questions and information contained in the uploaded documents.

However, retrieval performance depends on factors such as chunk size, document structure, embedding quality, and the complexity of the user's question.

```
# Faculty Search by Research Area

## Artificial Intelligence (AI)
Faculty members with Artificial Intelligence-related interests or specialisations include:
- Prof. Bulbul Ahamed
- Mohammad Naderuzzaman
- Salma Tabashum
- Md. Rashedul Islam
- Zarin Hadika
- Tasnia Haque Kheya
- Nayeem Ullah
- Oishika Khair Esha
- Md. Arafat
- Arifur Rahman
- Asif Ahmed
```

Figure 6.6: Source Data

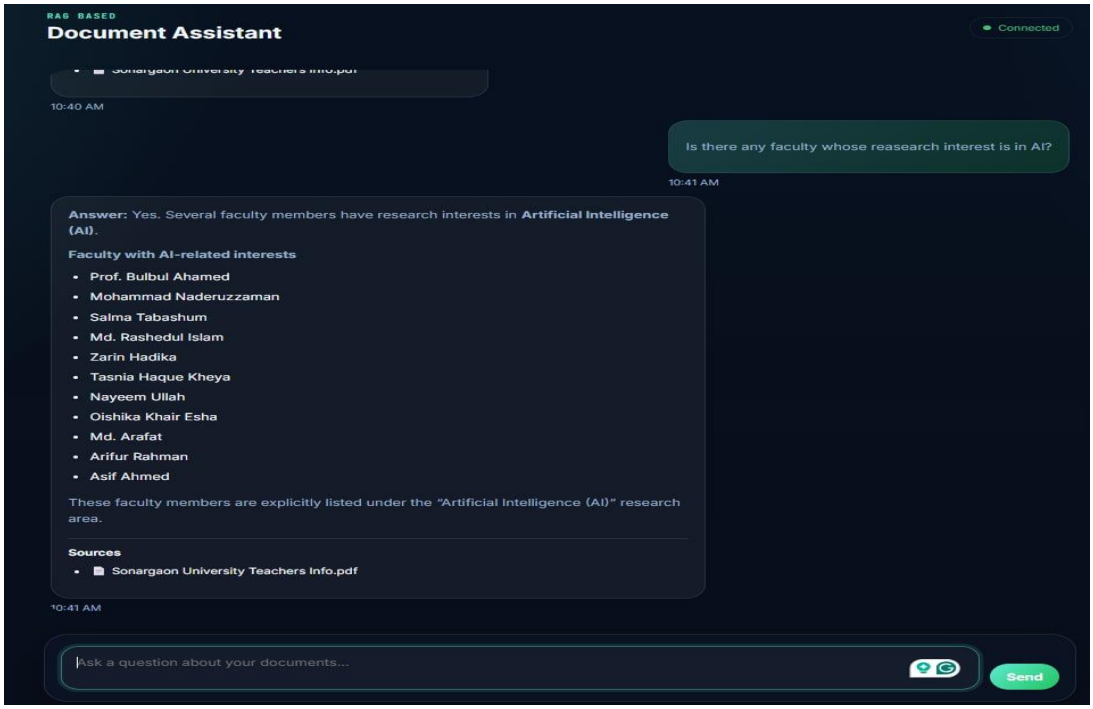


Figure 6.7: Semantic Retrieval and Source Result

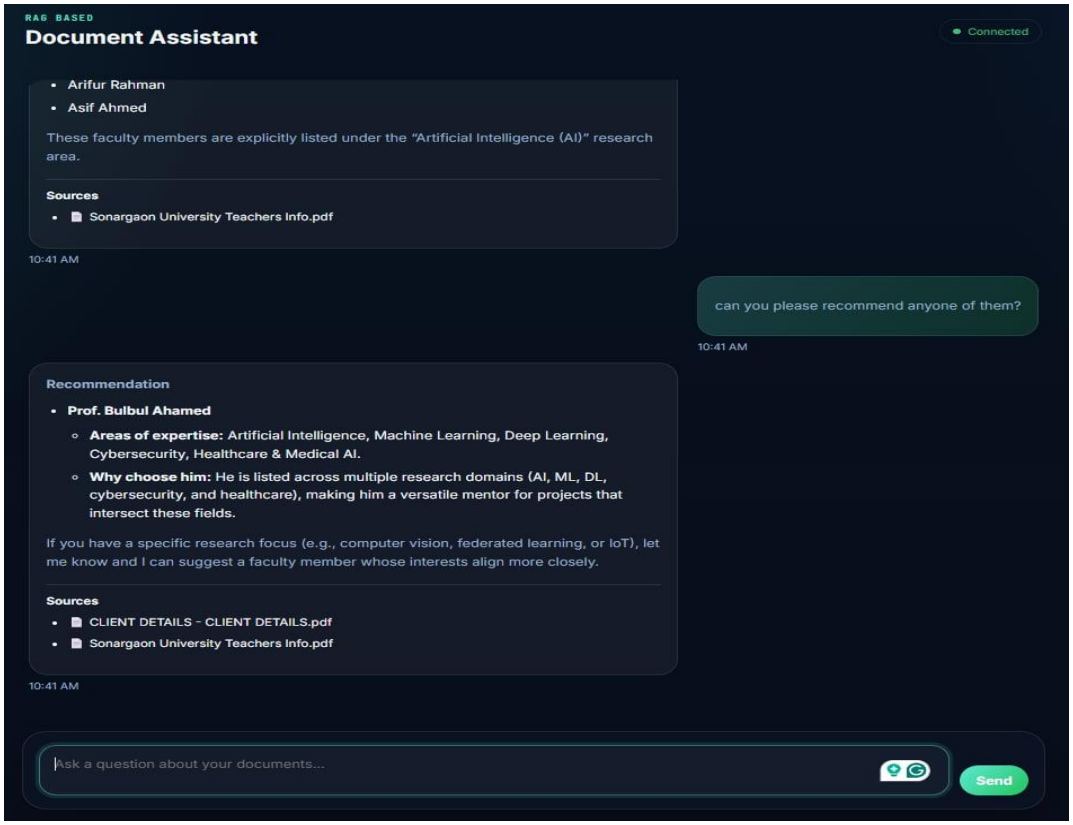


Figure 6.8: Semantic Retrieval and Source Result

6.6 Discussion

The implementation and testing demonstrate that the proposed RAG architecture can successfully combine semantic document retrieval with LLM-based natural-language generation.

A major advantage of the implemented architecture is that the LLM is not required to contain all the required information within its pretrained knowledge. Instead, the system can dynamically retrieve relevant information from documents supplied by the user.

This makes the system particularly suitable for domain-specific or private information that may not be available in the general knowledge of an LLM.

The retrieval layer plays a critical role in this process. If the appropriate information is successfully retrieved, the LLM receives relevant context and can generate an appropriate response. However, if the retrieval stage fails to identify the required information, the quality of the final answer may also decrease.

Therefore, the overall performance of a RAG system should not be considered solely an LLM problem. It depends on the complete pipeline:

Document Processing → Chunking → Embedding → Retrieval → Context Construction → LLM Generation

The current implementation demonstrates this pipeline successfully at an application level.

CHAPTER 7
CONCLUSION AND FUTURE WORK

7.1 Conclusion

This project presented the design and implementation of an LLM-based Retrieval-Augmented Generation system for question answering over user-provided documents.

The primary motivation behind the project was to develop an application that could provide conversational access to information contained in documents without requiring users to manually search through lengthy files.

The proposed system combines document processing, text chunking, semantic embedding generation, vector storage, semantic retrieval, context construction, and Large Language Model-based answer generation into a unified application.

The system was implemented using React for the frontend and FastAPI for the backend. Uploaded documents are processed and divided into smaller textual chunks. The sentence-transformers/all-MiniLM-L6-v2 model is used to generate embeddings for these chunks, while ChromaDB provides vector storage and semantic retrieval functionality.

When a user submits a question, the system generates a representation of the question and searches the stored document embeddings to identify relevant information. The retrieved content is then included as context in a prompt sent to the openai/gpt-oss-120b model through the Groq API.

The generated response is subsequently returned to the React frontend and displayed to the user.

The implementation demonstrates that RAG can provide an effective mechanism for connecting an LLM with user-provided knowledge sources. Instead of requiring model retraining whenever new information is introduced, users can add new documents to the knowledge base and make the information available for retrieval.

The project therefore demonstrates a practical implementation of an LLM-based RAG architecture and establishes a foundation that can be extended into a more advanced document intelligence platform.

7.2 Contributions of the Project

The project makes several contributions from both technical and application perspectives.

7.2.1 End-to-End RAG Implementation

The project implements a complete RAG pipeline rather than focusing on only one component.

The system covers:

Document Upload → Text Extraction → Chunking → Embedding → Vector Storage → Retrieval → Context Construction → LLM Generation → Response

This demonstrates the integration of multiple technologies into a functional application.

7.2.2 User-Provided Knowledge Integration

The system allows users to introduce their own documents as knowledge sources.

This provides flexibility because the application does not depend exclusively on a predefined dataset.

7.2.3 Semantic Document Retrieval

The use of embedding-based retrieval allows users to ask questions using natural language without requiring exact keyword matching.

7.2.4 LLM-Based Natural Language Generation

The system integrates an externally hosted Large Language Model to transform retrieved document information into natural-language responses.

This provides a conversational interface over otherwise static documents.

7.2.5 Source-Aware Responses

Document metadata is preserved during the indexing process, enabling the application to identify the document associated with retrieved information.

This provides a foundation for more advanced citation and source-verification functionality.

7.2.6 Modular Software Architecture

The separation of frontend, backend, document processing, vector storage, retrieval, and LLM services provides a maintainable architecture.

Future developers can replace or improve individual components without rebuilding the complete system.

7.3 Limitations

Although the project successfully implements the intended RAG functionality, several limitations remain.

First, the current system primarily focuses on textual information. Documents containing complex visual elements, scanned pages, diagrams, or complicated tables may require additional processing techniques.

Second, retrieval quality depends on the embedding model and chunking strategy. Poorly selected chunk boundaries or insufficient semantic representation can result in irrelevant or incomplete context being retrieved.

Third, the system depends on an external LLM service for answer generation. Therefore, network availability and API limitations can affect system availability.

Fourth, the current implementation does not include a complete production-level user authentication system. In a multi-user deployment, appropriate access control would be required to prevent users from accessing documents belonging to other users.

Fifth, the current vector storage implementation is primarily intended for the implemented application and development environment. A production system would require a more robust persistent storage architecture and appropriate backup and recovery mechanisms.

Sixth, the current system does not perform formal quantitative evaluation of answer accuracy and retrieval performance. A larger evaluation dataset and standardized metrics would be required to make statistically supported claims about system performance.

Finally, the current implementation does not contain a specialized prediction model. Although collections of documents such as previous examination papers can be analyzed for patterns, the system should not be considered a guaranteed examination-question prediction system.

7.4 Future Work

The proposed system provides a foundation for several possible improvements.

7.4.1 Improved Document Processing

Future versions can support a wider range of document formats and more complex document structures.

Possible improvements include:

- Advanced PDF parsing
- OCR for scanned documents
- Table extraction
- Image and diagram processing
- Structured document parsing
- Improved preservation of headings and sections

These improvements would allow the system to work effectively with more complex real-world documents.

7.4.2 Improved Retrieval Techniques

The current retrieval mechanism can be enhanced using more advanced approaches.

Future implementations may incorporate:

- Hybrid keyword and semantic search
- Re-ranking models

- Metadata filtering
- Query expansion
- Multi-stage retrieval
- Better chunking strategies
- Parent-child document retrieval

These techniques could improve the relevance of the context provided to the LLM.

7.4.3 Persistent Database Architecture

A production version should implement a persistent database architecture for storing:

- User accounts
- Uploaded documents
- Document metadata
- Knowledge-base information
- Conversation history
- User permissions
- Vector collection information

This would allow the application to support multiple users and persistent knowledge bases.

7.4.4 Authentication and Authorization

Future versions should include secure user authentication and authorization.

Each user could have an independent document collection so that uploaded documents remain private.

Possible functionality includes:

- User registration
- Login
- Password management
- Role-based access
- Private document collections
- Administrative controls

7.4.5 Advanced Source Citation

The current system identifies the original document associated with retrieved chunks.

Future versions can provide more detailed citations, such as:

- Document name

- Page number
- Section heading
- Exact source passage
- Clickable citation
- Highlighted source text

This would make the system more transparent and useful for academic and professional applications.

7.4.6 Conversation Memory

The current system primarily focuses on document-grounded individual questions.

A future version could introduce conversation memory so that the system understands references to previous questions.

For example:

User: "What are the admission requirements?"

AI: "The admission requirements are..."

User: "What about the minimum GPA?"

A conversational memory system could understand that "the minimum GPA" refers to the previously discussed admission requirements.

7.4.7 Examination Question Analysis

One particularly interesting future application is academic examination analysis.

Users could upload multiple years of question papers and the system could analyze:

- Frequently appearing topics
- Repeated questions
- Topic frequency
- Chapter distribution
- Question types
- Changes in question patterns
- Frequently tested concepts

The system could then provide statements such as:

> "Database normalization appeared in 4 of the last 5 uploaded examination papers."

or:

"Operating system scheduling questions appeared frequently across the uploaded papers."

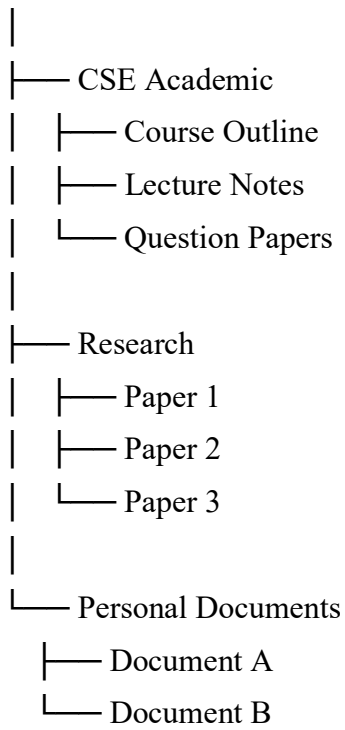
A more advanced version could combine RAG with statistical analysis or machine learning to generate probability-based topic suggestions.

Importantly, such a system should describe its output as **analytical suggestions**, not guaranteed predictions.

7.4.8 Multiple Knowledge Bases

Future versions could allow users to create separate knowledge bases.

For example:



Users could select a specific knowledge base before starting a conversation.

This would make the system more scalable and useful for real-world applications.

7.4.9 Local and Privacy-Focused Deployment

Another future direction is integration with locally hosted models.

A local model could allow sensitive documents to be processed without sending the retrieved context to an external API.

This could be particularly useful for:

- Corporate documents
- Confidential research

- Internal policies
- Private academic materials
- Sensitive organizational information

Such a deployment would require additional hardware and model optimization considerations.

7.4.10 Quantitative Evaluation

Future research should introduce a formal evaluation framework.

A benchmark dataset could be created containing:

- Document collection
- User questions
- Expected answers
- Relevant source chunks

The system could then be evaluated using metrics such as:

- Retrieval Precision
- Retrieval Recall
- F1-score
- Answer correctness
- Answer relevance
- Context relevance
- Faithfulness
- Response latency

This would allow future versions of the project to provide quantitative evidence of improvements.

7.5 Final Remarks

The developed system demonstrates the practical application of Retrieval-Augmented Generation for document-based question answering.

The project successfully combines modern web technologies, semantic search, vector databases, embedding models, and Large Language Models into a unified application.

The most important characteristic of the system is its ability to connect a general-purpose LLM with user-provided knowledge. This enables the application to operate over documents that are specific to the user or organization without requiring the LLM itself to be retrained for every new document collection.

Although the current implementation represents a functional prototype rather than a fully production-ready commercial platform, it provides a strong technical foundation for further development.

With improvements in retrieval, document processing, persistence, authentication, citation, evaluation, and analytical capabilities, the proposed architecture could evolve from a basic document question-answering application into a broader document intelligence platform.

Therefore, the project demonstrates both the practical feasibility of LLM-based RAG systems and their potential for applications in academic, research, organizational, and professional environments.

REFERENCES

[1] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," Advances in Neural Information Processing Systems, vol. 33, pp. 9459–9474, 2020.

[2] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pp. 3982–3992, 2019.

[3] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih, "Dense Passage Retrieval for Open-Domain Question Answering," in Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 6769–6781, 2020.

[4] S. Es, J. James, L. Espinosa Anke, and S. Schockaert, "RAGAs: Automated Evaluation of Retrieval Augmented Generation," in Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations, pp. 150–158, 2024.

[5] J. Saad-Falcon, O. Khattab, C. Potts, and M. Zaharia, "ARES: An Automated Evaluation Framework for Retrieval-Augmented Generation Systems," in Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, vol. 1, pp. 338–354, 2024.

[6] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the Middle: How Language Models Use Long Contexts," Transactions of the Association for Computational Linguistics, vol. 12, pp. 157–173, 2024.