



JANVIER 2025

Chat with multiple pdfs using llm



Project Report

Field of study

Computer Science : Cybersecurity

By

Hiba EZZAHI Niama FTASSI

&

Abdelhakim HANI Hamza EZZAHAOUI

***Chat with multiple pdfs
using llm***

Academic Advisor :

Dr. Yasser ADERGHAL

Professional Mentor:

Dr. HAKIM HAFIDI

2024 -2025

Acknowledgements :

The successful completion of this project would not have been possible without the guidance, support, and encouragement of many individuals. I would like to take this opportunity to express my heartfelt gratitude to everyone who contributed to this endeavor.

First and foremost, I extend my deepest thanks to **Mr. Hakim Hafidi** and **Mr. Yasser Aderghal**, who provided invaluable supervision throughout this project. Their expertise, availability, and insightful advice played a crucial role in overcoming challenges and achieving the project's objectives. Their ability to simplify complex concepts and guide us with patience and clarity made this experience incredibly enriching.

I am equally grateful to the entire academic team at our institution, whose support and commitment to our education provided a solid foundation for this project. Their dedication to fostering a positive learning environment and their readiness to assist us at every step greatly contributed to our progress.

I would also like to thank my project teammates, **Hiba Ezzahi**, **Niama Ftassi**, **Abdelhakim Hani**, and **Hamza Ezzahaoui**, for their exceptional collaboration and unwavering dedication. Working alongside such a committed and talented group was both a privilege and a pleasure. The teamwork, mutual support, and shared determination we exhibited throughout this journey were instrumental in overcoming challenges and achieving our goals.

Lastly, I am profoundly grateful to my family and friends for their constant encouragement, understanding, and emotional support during this project. Their belief in my abilities and their words of motivation provided me with the strength to persevere and succeed.

To everyone who contributed directly or indirectly to the success of this project, I extend my sincerest thanks and appreciation.

Abstract :

The PDF Chat Application is an innovative tool designed to enhance user interaction with PDF documents through the use of advanced natural language processing (NLP) techniques and artificial intelligence (AI). This application allows users to upload PDF documents and ask questions regarding their content, providing accurate and contextually relevant answers generated by Google's Gemini AI model. The application extracts text from PDFs, breaks it into manageable chunks, converts these chunks into embeddings, and stores them in a vector database for efficient similarity search.

Utilizing cutting-edge technologies such as FAISS for vector-based search, Langchain for large language model integration, and Streamlit for an interactive web interface, the system enables rapid querying and information retrieval. By transforming text from PDFs into embeddings and leveraging powerful AI, the application delivers a seamless user experience for document analysis, making it an effective tool for research, data extraction, and knowledge discovery.

This report outlines the architecture, implementation steps, and challenges involved in the development of the PDF Chat Application, highlighting its potential applications and the technologies that make it a powerful tool for interactive document querying.

Keywords :

-
- PDF Chat Application ,Natural Language Processing (NLP) ,Google Gemini AI, Text Extraction, Document Chunking , Embedding Generation, FAISS , Vector Search, Langchain , AI-driven Querying ,Vector Store.

List of Abbreviations

- **AI** : Artificial Intelligence
- **FAISS** : Facebook AI Similarity Search
- **PDF** : Portable Document Format
- **API** : Application Programming Interface
- **LLM** : Large Language Model
- **NLP** : Natural Language Processing
- **GPU** : Graphics Processing Unit
- **UI** : User Interface
- **Gemini AI** : Google's Generative AI
- **PyPDF2** : Python PDF Library
- **Langchain** : A framework for large language model applications
- **Streamlit** : A framework to create interactive web

Table of contents

<i>Acknowledgements</i> :	3
<i>Abstract</i> :	4
Keywords :	4
<i>List of Abbreviations</i>	5
CHAPTER 1	8
Key Components :	9
I. Introduction :	10
1. System Workflow :	11
2. Project Architecture :	12
3. Performance and Optimization :	12
4. Challenges Faced :	13
CHAPTER 2	15
I. PDF Chat Application Step by Step Implementation Guide :	16
1. Step 1 : Initial Setup and Imports :	16
2. Step 2 : required libraries :	16
3. Step 3 - Text Chunking :	16
4. Step 4 : Vector Store Creation :	17
5. Step 5 : Conversation Chain Setup :	17
6. Step 6 : User Input Processing :	17
7. Step 7 : Main Application Interface :	17
II. File Structure :	17
1. The project file structure :	17
2. Running the Application :	18
a. streamlit run app.p :	18
b. Usage Flow :	18
c. Error Handling :	18
d. Performance Considerations :	18
CHAPTER 3	19
I. Application interface :	20
CHAPTER 4	23
Conclusion :	24

List of Figures

Figure 1 : Project Architecture	12
Figure 2 : File structure	18
Figure 3 : App.....	20
Figure 4 : App.....	20
Figure 5 : App.....	21
Figure 6 : App.....	21
Figure 7 : App.....	22
Figure 8 : App.....	22

CHAPTER 1

Introduction

Key Components :

The PDF Chat Application leverages several key libraries and tools to deliver its core functionality. Streamlit is utilized to build the interactive user interface, allowing users to upload PDF files, input queries, and receive AI-generated responses. PyPDF2 plays a crucial role in processing and extracting text from the uploaded PDF files, ensuring that the document's content is accessible for further analysis.

To handle the interaction with large language models, the application uses Langchain, a powerful framework that aids in processing text and generating embeddings. These embeddings are stored in a FAISS vector database, which allows the system to perform efficient similarity searches for retrieving relevant chunks of text in response to user queries. Finally, `google.generativeai`, the API for Google's Gemini AI model, is employed to generate accurate and contextually relevant responses based on the retrieved text.

Together, these technologies form the backbone of the application, enabling seamless document querying and AI-driven interaction.

I. Introduction :

The PDF Chat Application is a dynamic and interactive solution designed to facilitate the extraction and querying of information from PDF documents using Google's Gemini AI model. In today's fast-paced digital world, extracting useful data from documents in an efficient and accessible way is crucial, especially for tasks like research, document analysis, and knowledge retrieval. This application provides users with the ability to upload PDF files and pose questions regarding their content, with the system utilizing cutting-edge technologies to deliver accurate and contextually relevant answers.

The core functionality of the application revolves around processing PDF files, converting them to text, and breaking down the content into manageable chunks. These chunks are then transformed into embeddings, which are stored in a vector database (FAISS), enabling the system to efficiently retrieve relevant sections of the document in response to user queries. By leveraging Google's Gemini AI, the application generates accurate and meaningful responses based on the context of the document, making it a valuable tool for users seeking quick insights and answers from large collections of data.

This report outlines the design, implementation, and key functionalities of the PDF Chat Application, as well as the technologies involved, and the challenges encountered during the development process. The application offers seamless user experience, combining advanced document processing techniques with natural language processing (NLP) models to enable intuitive interactions with document content.

This introduction sets the stage for a more detailed exploration of your application's architecture, features, and functionality in the report.

1. System Workflow :

The system workflow of the PDF Chat Application is designed to ensure smooth processing of PDF documents and efficient querying using AI.

- **User Uploads PDF Documents :**

The user uploads a PDF file via the Streamlit web interface. Upon upload, PyPDF2 processes the document and extracts the text, making it ready for further analysis.

- **Text Processing and Chunking :**

After extracting the text, it is split into smaller, manageable chunks. This process ensures that the text is easier to handle and can be interpreted more effectively by the AI model. The chunking process is performed using Langchain or custom functions, optimizing the text for further AI processing.

- **Embedding Creation :**

Each chunk of text is then converted into an embedding using a suitable embedding model (through Langchain). These embeddings are stored in a FAISS vector database, which facilitates fast and efficient similarity searches when responding to queries.

- **User Queries the Document :**

The user inputs a query in natural language via the Streamlit interface. This query is converted into an embedding, which is then compared against the stored document embeddings in the FAISS vector database to retrieve the most relevant text chunks.

- **AI Model Generates Responses :**

The relevant document chunks retrieved from FAISS are passed to Google Gemini AI, which processes the text, understands the context, and generates a response that directly addresses the user's query.

- **Displaying Responses :**

The AI-generated response is then displayed in the main interface area of the Streamlit app, offering the user a detailed answer based on the content of the uploaded PDF document.

2. Project Architecture :

The application architecture follows a modular approach with a clear separation of concerns: Frontend (Streamlit): Handles user interactions, file uploads, and displays responses. Backend (PyPDF2, Langchain, FAISS, Gemini AI): Handles PDF processing, text chunking, embedding generation, and AI-based query answering.

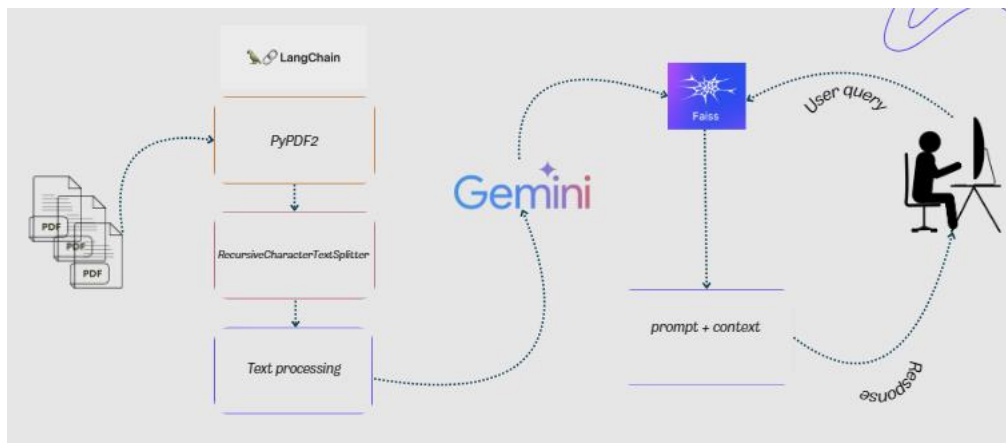


Figure 1 : Project Architecture

3. Performance and Optimization :

Performance and optimization are crucial aspects of the PDF Chat Application, ensuring that the system runs efficiently even with large documents and multiple user queries. The text chunking process plays a key role in this by dividing the document's content into manageable segments, which helps improve the AI's ability to process and respond to queries effectively. By breaking the text into smaller chunks, the system can focus on relevant portions of the document without being overwhelmed by excessive data.

Additionally, FAISS (the vector store) significantly optimizes performance by enabling fast and efficient retrieval of relevant document chunks. Through vector similarity search, FAISS ensures that the system can quickly identify and return the most pertinent chunks, even for large documents. This capability is essential for scaling the application and maintaining responsiveness when handling multiple queries.

Finally, embedding storage further enhances efficiency by ensuring that embeddings are stored for quick retrieval. This approach eliminates the need to reprocess the entire PDF each time a query is made, allowing the system to quickly access relevant information. By optimizing these key processes, the application delivers high performance and responsiveness, even when dealing with large-scale documents and numerous interactions.

4. Challenges Faced :

Several challenges were encountered during the development of the PDF Chat Application, each of which required thoughtful solutions to ensure optimal performance and user experience.

One of the primary challenges was PDF formatting. PDF documents often have complex layouts, including tables, images, and varying text structures, which can complicate the text extraction process. This can result in imperfect or incomplete text extraction. To mitigate this issue, additional preprocessing steps, such as handling embedded elements or manual cleanup, may be required to ensure the extracted text is clean and usable for further processing.

Another challenge revolved around the chunking strategy. Deciding the optimal chunk size for text splitting is crucial for balancing performance and accuracy. If the chunks are too large, the AI model might not be able to efficiently focus on relevant portions of the text, leading to imprecise responses. On the other hand, if the chunks are too small, they may lack the necessary context, which could result in incomplete or fragmented answers. Striking the right balance between chunk size and context preservation is key to achieving accurate responses.

Finally, response accuracy posed another significant challenge. Ensuring that the AI model generates accurate and contextually relevant answers requires continuous fine-tuning of both the model and the underlying processes. The system must be regularly tested and adjusted to handle a wide range of queries and document types effectively, making it necessary to maintain a rigorous testing and optimization process to enhance the quality of the responses.

These challenges required careful consideration and adaptation of the system, ensuring that it could handle diverse document types, provide accurate query responses, and deliver a seamless user experience.

CHAPTER 2

PDF Chat Application

Step by Step Implementation Guide

I. PDF Chat Application Step by Step Implementation

Guide :

The PDF Chat Application is a project that enables users to upload PDF documents and interact with their content by asking questions using Google Gemini AI. This guide presents a step-by-step implementation of the project, from initial setup to the final integration of AI for answering queries.

1. Step 1 : Initial Setup and Imports :

The first step in setting up the project is configuring the environment. Begin by creating a file in the project directory and adding the Google API key in the file, like so:
`GOOGLE_API_KEY = api key.`

2. Step 2 : required libraries :

Next, the required libraries need to be installed using the following command: `pip install streamlit Py PDF2 langchain langchain-google-genai faiss-cpu python-dotenv google-generativeai`. After installing the dependencies, the environment variables are loaded using Python code.

3. Step 3 : Text Chunking :

The next step involves splitting the extracted text into smaller, manageable chunks using the function `get_text_chunks(text)`. This process is done using the `RecursiveCharacterTextSplitter` from Langchain, which divides the text into chunks of a maximum size of 10,000 characters, with a 1,000-character overlap to maintain context between chunks. This helps make the text more manageable for the AI model while ensuring that contextual links between sections of text are preserved.

4. Step 4 : Vector Store Creation :

The next step is to convert the text chunks into embeddings (vector representations), which are then stored in a FAISS vector database for fast retrieval. The function `get_vector_store(text_chunks)` initializes Google's embedding model and creates a FAISS vector store to store the embeddings of the text chunks. This index is then saved locally for future use.

5. Step 5 : Conversation Chain Setup :

Once the embeddings are created and stored, the function `get_conversational_chain()` sets up the question-answering pipeline. It defines a prompt template with placeholders for context and the question, then initializes the Gemini Pro model to handle the queries. A QA chain is created by configuring the model and defining a custom prompt for interaction.

6. Step 6 : User Input Processing :

The function `user_input(user_question)` processes the user's query. It starts by loading the saved FAISS index. Then, it performs a similarity search to find the most relevant text chunks. The user's question is then passed through the QA pipeline to generate a response, which is displayed in the Streamlit interface.

7. Step 7 : Main Application Interface :

The `main()` function configures the Streamlit user interface. It sets the page title, creates the header, and defines the main interface with a question input field and a response display area. A sidebar is also created for PDF uploads and the processing button.

The document processing steps, including text extraction, chunking, embedding creation, and indexing, are handled here, followed by a success message once processing is completed.

II. File Structure :

1. The project file structure :

The project file structure consists of the following :


- 
- app_directory/
 - .env
 - app.py
 - faiss_index/
 - requirements.txt

Figure 2 : File structure

2. Running the Application :

To run the application, save the code in app.py, install all the required dependencies, and then launch the application with the following command.

a. `streamlit run app.p` :

b. Usage Flow :

The usage flow is split into two main parts: the user interface and the backend process. On the user side, access the web interface, upload the PDFs via the sidebar, click "Submit & Process," and ask questions in the main interface.

On the backend, the PDFs are processed to extract text, the text is split into chunks, the chunks are converted into embeddings, the embeddings are stored in FAISS, relevant chunks are retrieved for the query, and AI generates answers from those chunks.

c. Error Handling :

The application includes several error handling mechanisms:

- **API Key** : Ensures the API key is present.
- **Safe Deserialization** : Safeguards the FAISS index deserialization.
- **Processing Status** : Displays status indicators.
- **Success/Failure Messages** : Clear and informative messages to the user.

d. Performance Considerations :

Performance considerations include optimizing the chunk size to balance accuracy and speed, using overlap for context preservation, and storing embeddings locally to improve speed. Additionally, the AI model is configured with a temperature of 0.3 for more focused responses, and the embedding model is optimized for performance.

CHAPTER 3

Application

I. Application interface :

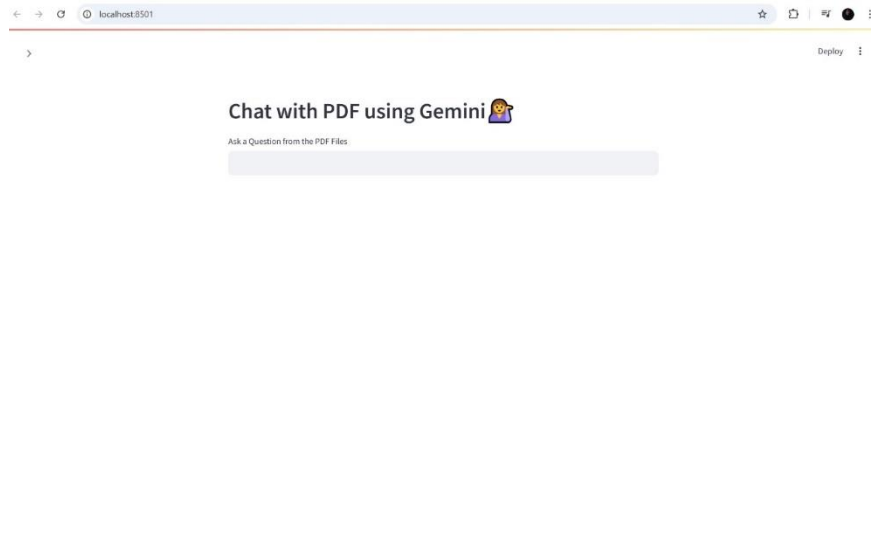


Figure 3 : App

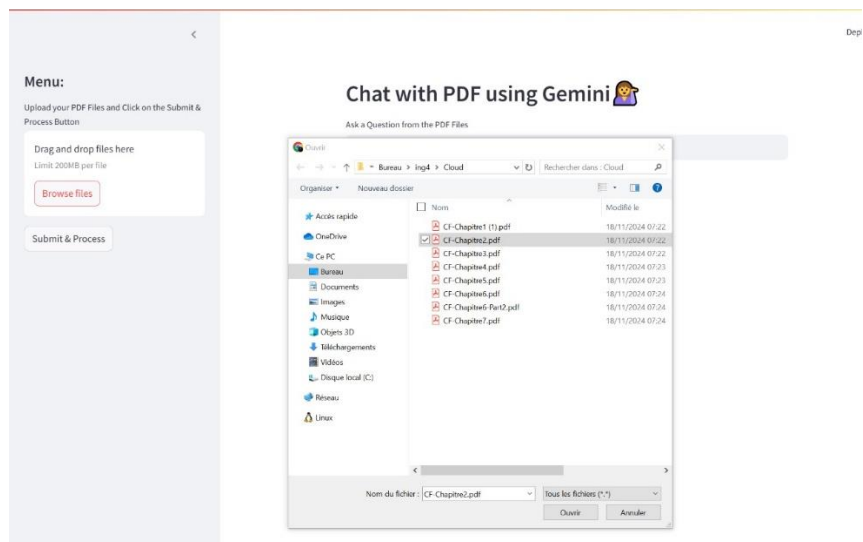


Figure 4 : App




Menu:

Upload your PDF Files and Click on the Submit & Process Button

Drag and drop files here

Limit 200MB per file

Browse files

	CF-Chapitre5.pdf 0.5MB	×
	CF-Chapitre4.pdf 1.2MB	×
	CF-Chapitre3.pdf 0.8MB	×

Showing page 1 of 2 < >

Submit & Process

Chat with PDF using Gemini

Ask a Question from the PDF Files

Figure 5 : App




Menu:

Upload your PDF Files and Click on the Submit & Process Button

Drag and drop files here

Limit 200MB per file

Browse files

	CF-Chapitre5.pdf 0.5MB	×
	CF-Chapitre4.pdf 1.2MB	×
	CF-Chapitre3.pdf 0.8MB	×

Showing page 1 of 2 < >

Submit & Process

Done

Chat with PDF using Gemini

Ask a Question from the PDF Files


Figure 6 : App


Menu:


Upload your PDF Files and Click on the Submit & Process Button

Drag and drop files here
Limit 200MB per file

[Browse files](#)

 CF-Chapitre5.pdf
0.34MB

 CF-Chapitre4.pdf
1.24MB

 CF-Chapitre3.pdf
0.88MB

Showing page 1 of 2 < >

[Submit & Process](#)

Chat with PDF using Gemini

Ask a Question from the PDF Files

provide a full summary

Reply: Virtualized servers are managed entirely by software. Management software can create or remove a new virtualized server at any time, without changing or rebooting physical servers. Elastic Compute and Virtualization Because it consists of a software artifact, a virtualized server is similar to a computer program. Each virtualized server must run on a physical server, and multiple virtualized servers can run on a given physical server concurrently. Elastic Compute and Virtualization Although multiple virtualized servers can run on a single physical server at the same time, each virtualized server is completely isolated from the others. The data and computations performed by one virtualized server cannot be observed or affected by another. Elastic Compute and Virtualization From a cloud provider's point of view, the ability to virtualize servers provides the basis for elastic computing and makes cloud computing economically viable. A cloud provider only needs to use computer software to increase or decrease the number of servers a customer is leasing. Because the provider does not need to reconfigure or reboot physical servers, a provider can accommodate the creation of thousands of virtualized servers as needed. Elastic Compute and Virtualization Because virtualization technologies guarantee isolation, a cloud provider can place virtualized servers on physical servers without regard to the owner, the apps they will run, or the data they will handle. Isolation allows a provider to mix virtualized servers from multiple customers on the same physical server without any interference and without any chance of data from one customer's virtualized server "leaking" to another customer. Elastic Compute and Virtualization A provider needs to choose where to run a virtualized server, the choice can be made in a way that provides the best benefit to the provider. In particular, a provider tries to avoid placing too many virtualized servers on the same physical server because each virtualized server will receive less processing power, causing customer complaints. A provider can use virtualized server placement to balance the load across all physical servers in the data center. Elastic Compute and Virtualization Using virtualized servers provides invaluable advantages for cloud providers, including the ability to scale the service and the ability to balance the load and avoid overloading a physical server while other physical servers remain idle. Elastic Compute and Virtualization To a customer, a virtualized server appears to act like a physical server. A virtualized server allows apps to communicate over the Internet. That is, like a physical server, each virtualized server is assigned an Internet address. But if a virtualized server merely acts like a physical server, how does a customer benefit? Elastic Compute and Virtualization Cloud providers and third-party vendors offer software that makes it easy to create new apps for a cloud environment. In addition, they offer software that can deploy an app in the cloud, including software that replicates an app on multiple virtualized servers. Elastic Compute and Virtualization Scaling an app to handle more users means adding more copies. If an app runs on virtualized servers, new copies can be created quickly (e.g., as requests arrive over the Internet). Elastic Compute and Virtualization Before deploying new apps or new versions of software, most large organizations deploy the software on an isolated test system before installing it in production. Virtualized servers allow an organization to create isolated virtualized servers for a test system without interfering with the production systems. Elastic Compute and Virtualization Elasticity


Figure 7 : App

Menu:

Upload your PDF Files and Click on the Submit & Process Button

Drag and drop files here
Limit 200MB per file

[Browse files](#)

 Project1-LCS.pdf
1.1MB

[Submit & Process](#)

Chat with PDF using Gemini

Ask a Question from the PDF Files

provide a summary of LCS

Reply: Longest Common Subsequence (LCS) is an algorithm used to determine the similarity between two sequences. It finds the length of the longest subsequence that appears in the same relative order in both sequences, but not necessarily contiguous. For example, "abc", "abg", "bdf", "aeg", "acefg", etc. are subsequences of "abcdefg". The LCS algorithm is used in various applications, including bioinformatics and file comparison programs. It is used to compute a score that represents the degree of similarity between two sequences. The algorithm has two phases: Phase 1: Computes the total score that indicates the degree of similarity by building a score matrix. Phase 2: Identifies the corresponding alignment using the score matrix built in the first phase. The time complexity of the LCS algorithm is $O(mn)$, where m and n are the lengths of the two sequences.

Figure 8 : App

CHAPTER 4

Conclusion

Conclusion :

The PDF Chat Application demonstrates a powerful integration of natural language processing (NLP) and artificial intelligence to enable efficient querying and interaction with PDF documents. By leveraging advanced tools like Streamlit, PyPDF2, Langchain, and Google's Gemini AI, the application effectively processes PDF content, converts it into embeddings, and uses a FAISS vector database for fast and accurate retrieval of relevant information.

Throughout the development of the application, various challenges were encountered, including handling complex PDF formatting, determining the optimal chunk size for text processing, and ensuring the accuracy of AI-generated responses. However, these challenges were overcome through careful design decisions and optimization techniques, resulting in a robust solution that balances performance, scalability, and user experience.

The application is designed to be highly adaptable, capable of scaling to handle large PDF documents and supporting various types of user queries. As a result, it offers a valuable tool for document analysis, research, and information retrieval.

Moving forward, the application can be further enhanced by expanding its capabilities, such as incorporating multi-file uploads, document summarization, and advanced search features. Additionally, continued testing and fine-tuning of the AI model will help improve response accuracy and contextual understanding, ensuring that the application remains a valuable resource for users seeking meaningful insights from their documents.

In conclusion, the PDF Chat Application showcases the power of AI and NLP in transforming how we interact with document-based information, making it an essential tool for efficient knowledge extraction in today's data-driven world.