A

Project Report

on

DocuQuery - AI-Powered PDF Knowledge Assistant

- **Team Name:** Cosmo
- **Team Members:**
    - 1.C.Ajay Kumar
    - 2.P.Manoj Kumar
- **Internship Organization:** [Smart Bridge]
- **Date of Submission:** 10/03/2025

2. Table of Contents

3. Introduction

DocuQuery is an innovative **AI-powered application** designed to revolutionize the way users interact with PDF documents. By leveraging cutting-edge **Natural Language Processing (NLP)** and **Machine Learning (ML)** techniques, DocuQuery enables users to extract, analyze, and query information from PDFs with unparalleled accuracy and efficiency. The application provides a **conversational interface** where users can ask questions in natural language and receive context-aware responses based on the content of their uploaded documents.

Built with **Streamlit** for the front-end, DocuQuery offers a user-friendly and intuitive interface. The backend integrates **Google Generative AI** (using the Gemini-Pro model) and **LangChain** to handle text processing, conversational memory, and retrieval tasks.

DocuQuery is particularly useful for professionals, researchers, and students who need to quickly extract insights from large PDF documents, such as research papers, reports, or manuals. By automating the process of knowledge extraction, DocuQuery saves time and enhances productivity, making it a valuable tool in today’s data-driven world.

4. Objective

The primary objective of DocuQuery is to:

- Enable users to upload PDF documents and extract text from them.
- Provide a conversational interface for users to ask questions and receive answers based on the content of the uploaded PDFs.
- Simplify the process of knowledge extraction from large documents using AI.

5. Features

- ➢**PDF Upload**: Users can upload multiple PDF files for processing.
- ➢**Text Extraction**: The application extracts text from uploaded PDFs.
- ➢**Text Chunking**: Extracted text is split into smaller chunks for efficient processing.
- ➢**Vector Embeddings**: Text chunks are converted into vector embeddings using **Hugging Face Embeddings**.
- ➢**Conversational AI**: Users can interact with the application using natural language queries.
- ➢**Chat History**: The application maintains a chat history for each session.

6. Technologies Used

Streamlit: For building the user interface.

PyPDF2: For extracting text from PDF files.

LangChain: For managing conversational chains and memory.

Google Generative AI: For generating responses using the **Gemini-Pro** model.

FAISS: For creating and managing vector stores.

Hugging Face Embeddings: For generating text embeddings.

Sentence Transformers: For text similarity and embeddings.

7. System Architecture

The system architecture of DocuQuery consists of the following components:

1. **Front-End**: Built using Streamlit, allowing users to upload PDFs and interact with the application.
2. **Back-End**:
   - ✓ Text extraction and chunking using **PyPDF2** and **LangChain**.
   - ✓ Vector embeddings using **Hugging Face Embeddings** and **FAISS**.
   - ✓ Conversational AI using **Google Generative AI** and **LangChain**.
3. **Database**: Vector stores are created and managed in memory for each session.

8. Implementation Details

Key Functions

1. **Text Extraction**:
   - ✓The get_pdf_text function extracts text from uploaded PDFs using **PyPDF2**.
   - ✓Handles cases where text extraction fails (e.g., image-based PDFs).
2. **Text Chunking**:
   - ✓The get_text_chunks function splits the extracted text into smaller chunks using **LangChain's RecursiveCharacterTextSplitter**.
3. **Vector Store Creation**:
   - ✓The get_vector_store function converts text chunks into vector embeddings

using **Hugging Face Embeddings** and stores them in a **FAISS** vector store.

4. **Conversational Chain**:
   - ✓The get_conversational_chain function sets up a conversational retrieval chain using **Google Generative AI** and **LangChain**.
5. **User Interaction**:
   - ✓The user_input function handles user queries and generates responses using the conversational chain.

9. Challenges Faced

- **Text Extraction**: Some PDFs (e.g., image-based or scanned documents) could not be processed due to the lack of extractable text.
- **API Key Management**: Ensuring secure handling of the Google Generative AI API key.
- **Memory Management**: Managing chat history and session state in Streamlit.

10. Results and Outcomes

- Successfully built a functional AI-powered PDF knowledge assistant.
- Users can upload PDFs, extract text, and ask questions in natural language.
- The application provides accurate and context-aware responses based on the content of the uploaded documents.

11. Future Scope

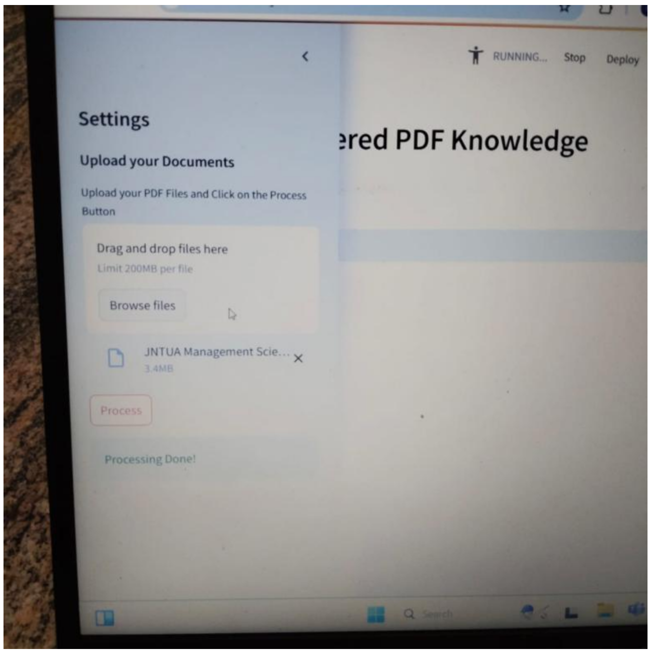
- **OCR Integration**: Add support for image-based PDFs using Optical Character Recognition (OCR).
- **Multi-Language Support**: Extend the application to support multiple languages.
- **Cloud Deployment**: Deploy the application on platforms like **Streamlit Community Cloud** or **Snowflake** for wider accessibility.
- **Enhanced UI**: Improve the user interface with additional features like document preview and search.

12. Conclusion

DocuQuery is a powerful tool for extracting and querying information from PDF documents using AI. It simplifies the process of knowledge extraction and provides a user-friendly interface for interacting with document content. The application demonstrates the potential of AI in document processing and opens up possibilities for future enhancements.

Appendices

Appendix A: Screenshots of the Application

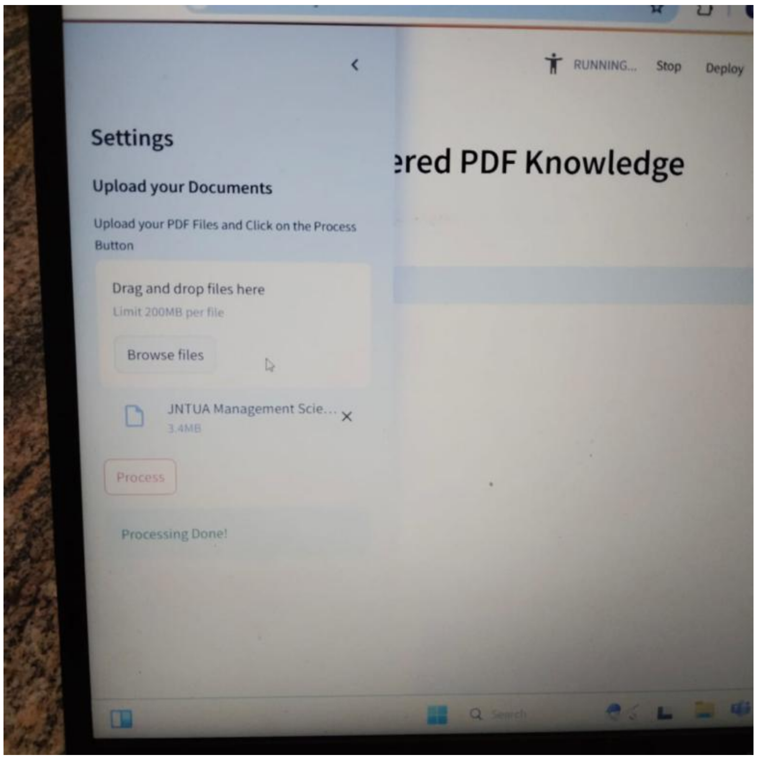


Upload your PDF Files and Click on the Process Button

Drag and drop files here
Limit 200MB per file

JNTUA Management Scie...
3.4MB

Processing Done!



Upload your PDF Files and Click on the Process Button

Drag and drop files here
Limit 200MB per file

JNTUA Management Scie...
3.4MB

Processing Done!

Appendix B: Sample Code

```
import os
import streamlit as st
from PyPDF2 import PdfReader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.embeddings import HuggingFaceEmbeddings  # Corrected Import
from langchain.vectorstores import FAISS
from langchain_google_genai import GoogleGenerativeAI  # Corrected Import
from langchain.chains import ConversationalRetrievalChain
from langchain.memory import ConversationBufferMemory

# Set API Key for Google Generative AI
os.environ["GOOGLE_API_KEY"] = "AIzaSyCeWYwsdLuoue4tqdG-y3TyIKLloug0Q-s"  # Replace with your actual API key

# Function to extract text from PDFs
def get_pdf_text(pdf_docs):
    text = ""
    for pdf in pdf_docs:
        pdf_reader = PdfReader(pdf)
        for page in pdf_reader.pages:
            page_text = page.extract_text()
            if page_text:
                text += page_text
            else:
```

```
            st.warning(f"Text extraction failed for a page in {pdf.name}. The PDF may be image-based or unreadable.")
    return text


# Function to split text into chunks
def get_text_chunks(text):
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
    return text_splitter.split_text(text)


# Function to create vector store from text chunks
def get_vector_store(text_chunks):
    embeddings = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")  # Fixed embedding model
    vector_store = FAISS.from_texts(texts=text_chunks, embedding=embeddings)
    return vector_store


# Function to set up conversational retrieval chain
def get_conversational_chain(vector_store):
    llm = GoogleGenerativeAI(model="gemini-pro")  # Updated model
    memory = ConversationBufferMemory(memory_key="chat_history", return_messages=True)
    conversation_chain = ConversationalRetrievalChain.from_llm(
        llm=llm,
        retriever=vector_store.as_retriever(),
        memory=memory
    )
    return conversation_chain

# Function to handle user input and generate responses
def user_input(user_question):
    response = st.session_state.conversation({'question': user_question})
    st.session_state.chat_history = response['chat_history']
    for i, message in enumerate(st.session_state.chat_history):
        st.write("Human: " if i % 2 == 0 else "Bot: ", message.content)

# Main function to set up Streamlit UI
def main():
    st.set_page_config(page_title="DocuQuery: AI-Powered PDF Knowledge
Assistant")
    st.header("DocuQuery: AI-Powered PDF Knowledge Assistant")

    # Initialize session state variables
    if 'conversation' not in st.session_state:
        st.session_state.conversation = None
    if 'chat_history' not in st.session_state:
        st.session_state.chat_history = []

    # Sidebar for document upload
    with st.sidebar:
        st.title("Settings")
        st.subheader("Upload your Documents")
        pdf_docs = st.file_uploader("Upload your PDF Files and Click on the
Process Button", accept_multiple_files=True)
```

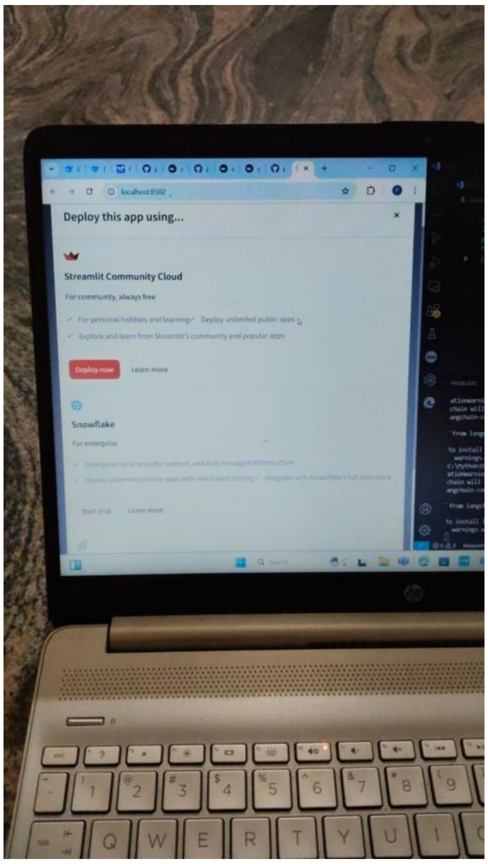
```
        if st.button("Process"):
            with st.spinner("Processing"):
                # Extract text from PDFs
                raw_text = get_pdf_text(pdf_docs)
                if not raw_text:
                    st.error("No text could be extracted from the uploaded PDFs. Please upload valid PDFs.")
                else:
                    # Split text into chunks
                    text_chunks = get_text_chunks(raw_text)
                    # Create vector store
                    vector_store = get_vector_store(text_chunks)
                    # Set up conversational chain
                    st.session_state.conversation = get_conversational_chain(vector_store)
                    st.success("Processing Done!")

    # User input for questions
    user_question = st.text_input("Ask a question from the PDF files:")
    if user_question:
        if 'conversation' not in st.session_state or st.session_state.conversation is None:
            st.warning("Please upload and process documents first.")
        else:
            user_input(user_question)

# Run the application
```

```
if __name__ == "__main__":
    main()  # Removed the unnecessary period
```

Appendix C: Deployment Options



localhost:8502

Deploy this app using...

Streamlit Community Cloud

For community, always free

✓ For personal hobbies and learning✓ Deploy unlimited public apps

✓ Explore and learn from Streamlit's community and popular apps

Deploy now Learn more

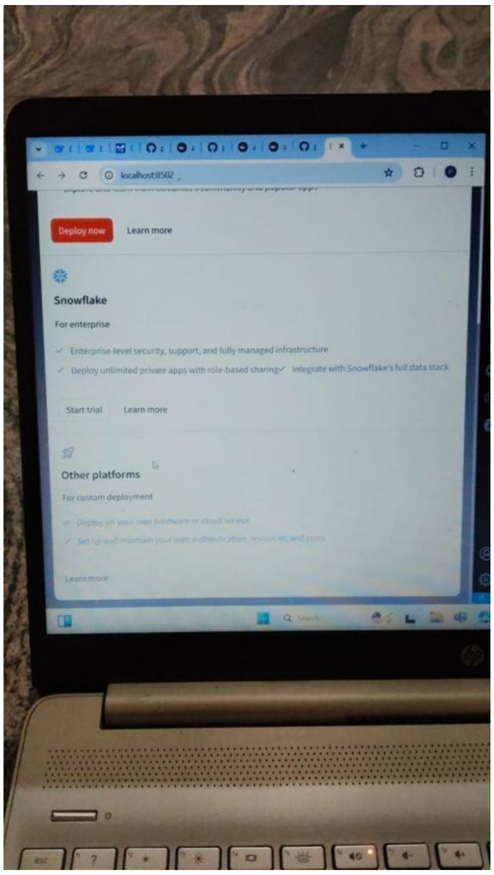
Snowflake

For enterprise

✓ Enterprise-level security, support, and fully managed infrastructure

✓ Deploy unlimited private apps with role-based sharing✓ Integrate with Snowflake's full data stack

Start trial Learn more



localhost:8502

Deploy now Learn more

Snowflake

For enterprise

✓ Enterprise-level security, support, and fully managed infrastructure

✓ Deploy unlimited private apps with role-based sharing✓ Integrate with Snowflake's full data stack

Start trial Learn more

Other platforms

For custom deployment

✓ Deploy on your own hardware or cloud service

✓ Set up and maintain your own authentication, resources, and costs

Learn more