Multi-PDF Content-Based Question Answering System

Your Name

October 2024

Contents

1 Introduction

This documentation describes the implementation of a multi-PDF content-based question-answering system. The system is built using Python libraries, Google Generative AI (Gemini), FAISS for vector search, and Streamlit for a web interface. It allows users to upload multiple PDF documents, extract text, and ask questions related to the content of the PDFs.

2 Libraries and Dependencies

The following Python libraries are used in the project:

- **os, shutil**: For file system operations.
- **Streamlit**: To create a web interface for the application.
- **PyPDF2 (PdfReader)**: To extract text from PDF documents.
- **RecursiveCharacterTextSplitter**: To split large text blocks into smaller chunks for processing.
- **google.generativeai**: To use Google Generative AI (Gemini) for language model interactions.
- **FAISS**: For efficient similarity search and vector storage.
- **LangChain**: For integrating language model functionalities with FAISS and Google Generative AI.
- **sklearn**: To compute text similarity using TF-IDF and cosine similarity.
- **hashlib**: For hashing passwords to securely store them.

3 Environment Setup

The system uses Google Generative AI (Gemini) and relies on an API key stored in environment variables.

```
from dotenv import load_dotenv
import os

load_dotenv()
api_key = os.getenv("GOOGLE_API_KEY")
genai.configure(api_key=api_key)
```

4 Login and Logout System

The system includes a simple login mechanism that verifies users by checking their hashed passwords.

4.1 Password Hashing Function

A function is used to hash the password securely using SHA-256.

```
import hashlib

def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()
```

4.2 Login Function

Users enter their credentials in the Streamlit sidebar. Their password is hashed and checked against stored hashes.

```
def login():
    st.sidebar.title("Login")
    username = st.sidebar.text_input("Username")
    password = st.sidebar.text_input("Password", type="password")
    if st.sidebar.button("Login"):
        hashed_password = hash_password(password)
        if username in users and users[username] == hashed_password:
            st.session_state['logged_in'] = True
            st.session_state['username'] = username
            st.sidebar.success(f"Welcome {username}!")
        else:
            st.sidebar.error("Incorrect username or password")
```

4.3 Logout Function

If a user is logged in, a logout option is displayed.

```
def logout():
    if 'logged_in' in st.session_state and st.session_state['logged_in']:
        st.sidebar.write(f"Logged in as {st.session_state['username']}")
        if st.sidebar.button("Logout"):
            st.session_state['logged_in'] = False
            st.session_state['username'] = None
```

5 PDF Text Extraction and Chunking

The text from PDF files is extracted using the PyPDF2 library. It is then chunked into manageable pieces using RecursiveCharacterTextSplitter.

5.1 PDF Text Extraction

The text is extracted from each page of the PDFs and stored along with the page number and document name.

```
def get_pdf_text_with_pages(pdf_paths):
    text_chunks_with_pages = []
    for pdf_path in pdf_paths:
        pdf_reader = PdfReader(pdf_path)
        doc_name = os.path.basename(pdf_path)
        for page_number, page in enumerate(pdf_reader.pages, start=1):
            text = page.extract_text()
```

```
            if text:
                text_chunks_with_pages.append((text, page_number, doc_name))
    return text_chunks_with_pages
```

5.2 Text Chunking

The extracted text is split into smaller chunks with a chunk size of 400 characters and a 50-character overlap.

```
from langchain.text_splitter import RecursiveCharacterTextSplitter

def get_text_chunks_with_pages(text_chunks_with_pages):
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=400, chunk_overlap
    chunks_with_pages = []
    for text, page_number, doc_name in text_chunks_with_pages:
        chunks = text_splitter.split_text(text)
        for chunk in chunks:
            chunks_with_pages.append((chunk, page_number, doc_name))
    return chunks_with_pages
```

6 Vector Store Creation and Querying

A FAISS vector store is created to store the text chunks along with their embeddings, allowing for efficient similarity search.

6.1 Vector Store Creation

The text chunks are embedded using Google Generative AI embeddings and stored in a FAISS vector store.

```
from langchain_google_genai import GoogleGenerativeAIEmbeddings
from langchain_community.vectorstores import FAISS

def get_vector_store_with_pages(text_chunks_with_pages, index_name):
    embeddings = GoogleGenerativeAIEmbeddings(model="models/text-embedding-004")
    texts, page_numbers, doc_names = zip(*text_chunks_with_pages)
    vector_store = FAISS.from_texts(texts, embedding=embeddings)
    vector_store.save_local(index_name)
    with open("page_numbers_docs.pkl", "wb") as f:
        pickle.dump((page_numbers, doc_names), f)
```

7 Conversational Chain for Question Answering

A conversational chain is set up using LangChain and Google Generative AI to answer user questions based on the document content.

```
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain.prompts import PromptTemplate
from langchain.chains.question_answering import load_qa_chain

def get_conversational_chain():
    prompt_template = """
        Answer the question in a detailed manner using the given context.
        Question: {question}
        Context: {context}
    """
    prompt = PromptTemplate(template=prompt_template, input_variables=["question
    llm = ChatGoogleGenerativeAI(model="models/chat-bison-001")
    return load_qa_chain(llm=llm, prompt=prompt)
```

8 Conclusion

This system integrates multiple technologies, including FAISS, Google Generative AI, and LangChain, to create an interactive question-answering tool based on PDF documents. Users can upload multiple PDFs, and the system extracts text, processes it, and provides answers based on the document content.