Python Code Explanation and Documentation

Your Name

November 4, 2024

1 Introduction

This document provides detailed explanations for each line of the Python code, focusing on the functionality and purpose of every component. The Python code implements a Streamlit-based multi-PDF content-based question-answering system using various libraries such as PyPDF2, Langchain, and Google Generative AI (Gemini). The code also includes a user login system and text-based search using embeddings and TF-IDF techniques.

2 Code Breakdown

2.1 Imports

The code begins by importing necessary libraries and modules.

```
import os
import shutil
import streamlit as st
from PyPDF2 import PdfReader
from langchain.text_splitter import RecursiveCharacterTextSplitter
import google.generativeai as genai
from langchain_community.vectorstores import FAISS
from langchain_google_genai import GoogleGenerativeAIEmbeddings
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain.chains.question_answering import load_qa_chain
from langchain.prompts import PromptTemplate
from dotenv import load_dotenv
import pickle
import re
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import hashlib
```

- `os`, `shutil`: These modules are used for file and directory manipulation.
- `streamlit`: The library is used to create the web interface.
- `PyPDF2`: Provides functionalities to read and extract text from PDF files.

- `RecursiveCharacterTextSplitter`: Splits text into chunks for embedding and search.
- `google.generativeai`: Configures and interacts with Google Generative AI (Gemini model).
- `FAISS`: A vector store for fast similarity search.
- `load_qa_chain`: Loads a question-answering chain using a model and a prompt.
- `dotenv`: Loads environment variables from a `.env` file.
- `hashlib`: Provides secure hashing for password encryption.
- `TfidfVectorizer`, `cosine_similarity`: For text vectorization and calculating similarity scores.

2.2 Loading Environment Variables

```
# Load environment variables
load_dotenv()
api_key = os.getenv("GOOGLE_API_KEY")
```

`load_dotenv()` loads environment variables from a `.env` file, and `os.getenv()` retrieves the Google API key required for the Generative AI service.

2.3 Google Generative AI Configuration

```
# Configure the Google Generative AI
genai.configure(api_key=api_key)
```

This line configures the Google Generative AI (Gemini) by using the API key obtained earlier.

2.4 Hashing Passwords

```
# Function to hash passwords
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()
```

The function `hash_password` takes a plain text password, hashes it using the SHA-256 algorithm, and returns the hashed password for secure storage.

2.5 User Management (Login/Logout)

```
# Dictionary for storing users (in-memory for demo purposes)
users = {
    "user1": hash_password("password1"),
    "user2": hash_password("password2"),
}
```

This dictionary stores users and their corresponding hashed passwords. In production, a more secure database system would be used.

```
# User login system
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")
```

The `login()` function creates a login form in the Streamlit sidebar. The entered password is hashed and compared with the stored value. If the credentials are valid, the login session is established.

```
# Logout system
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
```

The `logout()` function allows users to log out and resets their session state.

2.6 PDF Text Extraction

```
# Function to extract text from multiple PDFs with document names
    and page numbers
def get_pdf_text_with_pages(pdf_paths):
    text_chunks_with_pages = []
    for pdf_path in pdf_paths:
        try:
            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))
        except FileNotFoundError as e:
            print(f"Error: {e}")
    return text_chunks_with_pages
```

This function reads the contents of multiple PDFs using `PyPDF2`. For each page, the extracted text is stored along with the page number and document name in a list of tuples.

2.7 Text Chunking

```python
# Split text into chunks, including document name and page numbers
def get_text_chunks_with_pages(text_chunks_with_pages):
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=400,
    chunk_overlap=50)
    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
```

The `get_text_chunks_with_pages()` function breaks long text into smaller chunks using `RecursiveCharacterTextSplitter`. The chunks maintain the page number and document name information.

2.8 Vector Store Creation

```python
# Create and save vector store with document names and page numbers
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)
```

This function creates a FAISS vector store using text embeddings from Google Generative AI. The vector store is saved locally, and the associated page numbers and document names are also stored in a `.pkl` file for later use.

2.9 Question Answering Chain

```python
# Function to get the conversational 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", "context"])
    llm = ChatGoogleGenerativeAI(model="models/chat-bison-001")
    return load_qa_chain(llm=llm, prompt=prompt)
```

The `get_conversational_chain()` function creates a question-answering chain by loading a conversational prompt template and configuring it with the Google Generative AI large language model.

3 Conclusion

This LaTeX documentation details every aspect of the Python code, focusing on its structure and functionality. The code implements a content-based question-answering system over multiple PDF documents using embeddings and vector search techniques. Each section has been explained to provide a clear understanding of how the system works.