# rag\_pipeline\_project.py

import os

import requests

from bs4 import BeautifulSoup

from PyPDF2 import PdfReader

from langchain.text\_splitter import RecursiveCharacterTextSplitter

from sentence\_transformers import SentenceTransformer

import pinecone

from langchain.prompts import PromptTemplate

from langchain.llms import OpenAI

# --- Configuration ---

PINECONE\_API\_KEY = "YOUR\_PINECONE\_API\_KEY"

PINECONE\_ENV = "gcp-starter" # Adjust based on your Pinecone environment

OPENAI\_API\_KEY = "YOUR\_OPENAI\_API\_KEY"

EMBEDDING\_MODEL\_NAME = "all-MiniLM-L6-v2"

# --- PDF RAG Pipeline ---

def extract\_text\_from\_pdf(pdf\_path):

reader = PdfReader(pdf\_path)

text = ""

for page in reader.pages:

text += page.extract\_text() or ""

return text

def chunk\_text(text, chunk\_size=500, chunk\_overlap=100):

text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=chunk\_size, chunk\_overlap=chunk\_overlap)

return text\_splitter.split\_text(text)

def embed\_and\_store(chunks, index\_name):

pinecone.init(api\_key=PINECONE\_API\_KEY, environment=PINECONE\_ENV)

embedder = SentenceTransformer(EMBEDDING\_MODEL\_NAME)

index = \_get\_or\_create\_index(index\_name, embedder.get\_sentence\_embedding\_dimension())

embeddings = embedder.encode(chunks, show\_progress\_bar=True)

vectors = [{"id": str(i), "values": emb, "metadata": {"text": chunks[i]}} for i, emb in enumerate(embeddings)]

index.upsert(vectors)

print(f"{len(chunks)} chunks uploaded to Pinecone!")

def query\_pdf\_rag(user\_query, index\_name):

embedder = SentenceTransformer(EMBEDDING\_MODEL\_NAME)

pinecone.init(api\_key=PINECONE\_API\_KEY, environment=PINECONE\_ENV)

index = pinecone.Index(index\_name)

query\_embedding = embedder.encode(user\_query).tolist()

results = index.query(vector=query\_embedding, top\_k=5, include\_metadata=True)

contexts = [match['metadata']['text'] for match in results['matches']]

llm = OpenAI(model\_name="gpt-3.5-turbo", api\_key=OPENAI\_API\_KEY)

prompt\_template = PromptTemplate(template="Context: {context}\n\nQuestion: {question}\nAnswer:",

input\_variables=["context", "question"])

return llm(prompt\_template.format(context="\n".join(contexts), question=user\_query))

# --- Website RAG Pipeline ---

def scrape\_website(url):

response = requests.get(url)

if response.status\_code == 200:

soup = BeautifulSoup(response.text, "html.parser")

return " ".join([p.get\_text() for p in soup.find\_all("p")])

else:

print("Failed to fetch URL:", url)

return ""

def process\_and\_store\_web\_content(url, index\_name):

text = scrape\_website(url)

if not text:

return

chunks = chunk\_text(text)

embed\_and\_store(chunks, index\_name)

def query\_web\_rag(user\_query, index\_name):

return query\_pdf\_rag(user\_query, index\_name) # Reuses the same query function

# --- Helper Functions ---

def \_get\_or\_create\_index(index\_name, dimension):

if index\_name not in pinecone.list\_indexes():

pinecone.create\_index(index\_name, dimension=dimension)

return pinecone.Index(index\_name)

# --- Main Execution ---

def main():

print("\n--- Retrieval-Augmented Generation (RAG) Pipeline ---")

print("Choose an option:")

print("1. Process PDF")

print("2. Query PDF")

print("3. Process Website")

print("4. Query Website")

choice = input("Enter the option number: ")

if choice == "1":

pdf\_path = input("Enter the path to the PDF file: ").strip()

index\_name = input("Enter the Pinecone index name: ").strip()

text = extract\_text\_from\_pdf(pdf\_path)

chunks = chunk\_text(text)

embed\_and\_store(chunks, index\_name)

print("PDF processed successfully and stored!")

elif choice == "2":

index\_name = input("Enter the Pinecone index name: ").strip()

query = input("Enter your query: ").strip()

response = query\_pdf\_rag(query, index\_name)

print("\nResponse:", response)

elif choice == "3":

url = input("Enter the website URL: ").strip()

index\_name = input("Enter the Pinecone index name: ").strip()

process\_and\_store\_web\_content(url, index\_name)

print("Website content processed and stored!")

elif choice == "4":

index\_name = input("Enter the Pinecone index name: ").strip()

query = input("Enter your query: ").strip()

response = query\_web\_rag(query, index\_name)

print("\nResponse:", response)

else:

print("Invalid option! Please run the program again.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

## Short Report: How to Use

1. **Save the Code**: Save the provided code as **rag\_pipeline\_project.py**.
2. **Install Dependencies**: Run the following command to install all required libraries:

bash

Copy code

pip install PyPDF2 langchain openai sentence-transformers pinecone-client faiss-cpu requests beautifulsoup4

1. **Setup API Keys**: Replace the placeholders in the code with your actual API keys:
   * YOUR\_PINECONE\_API\_KEY
   * YOUR\_OPENAI\_API\_KEY
2. **Run the Program**: Execute the script:

bash

Copy code

python rag\_pipeline\_project.py

1. **Follow the Prompts**: The program will prompt you to:
   * **Process a PDF**: Extract, chunk, embed, and store in Pinecone.
   * **Query a PDF**: Ask questions about the processed PDF.
   * **Process a Website**: Scrape content, embed, and store.
   * **Query a Website**: Ask questions about processed websites.

**Sample Workflow**

1. **Processing a PDF**:
   * Input: path\_to\_pdf\_file.pdf
   * Index Name: pdf-rag-index
2. **Querying a PDF**:
   * Input a query like:  
     *"What is the unemployment rate for graduates on page 2?"*
3. **Processing a Website**:
   * URL: https://www.stanford.edu
   * Index Name: web-rag-index
4. **Querying a Website**:
   * Input a query like:  
     *"What programs does Stanford University offer?"*