


```
# Install required libraries
!pip install openai==0.28 langchain faiss-cpu sentence-transformers tiktoken pdfplumber
```

 Show hidden output

```
# Import libraries
import openai
import faiss
import tiktoken
import pdfplumber
import random
import numpy as np
import pandas as pd
from langchain.text_splitter import RecursiveCharacterTextSplitter
from sentence_transformers import SentenceTransformer
```

```
# Initialize OpenAI API key
openai.api_key = "YOUR_OPENAI_API_KEY" # Replace with your API key
```

 /usr/local/lib/python3.10/dist-packages/sentence_transformers/cross_encoder/CrossEncoder.py:13: TqdmExperimentalWarning: Usi
from tqdm.autonotebook import tqdm, trange

```
# Function to extract text from a PDF file
def extract_text_from_pdf(file_path):
    text = ""
    with pdfplumber.open(file_path) as pdf:
        for page in pdf.pages:
            text += page.extract_text()
    return text
```

check if the pdf is upload successfully

 Generate


print hello world using rot13



Close

```
# Extract text from the uploaded PDF
document_text = extract_text_from_pdf('/ORD_Use_Agreement.pdf')

# Check if the text extraction was successful
if document_text:
    print("PDF successfully uploaded and text extracted!")
    # Print the first 1000 characters to verify the content
    print("\nSample Extracted Text (First 1000 characters):\n")
    print(document_text[:1000]) # Print only the first 1000 characters
else:
    print("Failed to extract text from the PDF. Please check the file path and try again.")
```

 PDF successfully uploaded and text extracted!

Sample Extracted Text (First 1000 characters):

```
CHICAGO- O'HARE INTERNATIONAL AIRPORT
*****
AMENDED AND RESTATED
AIRPORT USE AGREEMENT
AND TERMINAL FACILITIES LEASE
*****
(As Amended through 2001 - Unofficial Version)
J54154-2 C:\Documents and Settings\OM00022\Local
Settings\Temp\XPgrpwise\ohareuseagreementamendedandrestatedunoff
icialversion.wpdTABLE OF CONTENTS
Page
ARTICLE I DEFINITIONS . . . . . 1
Section 1.01 - Definitions . . . . . 1
Section 1.02 - Interpretation . . . . . 17
Section 1.03 - Incorporation of Exhibits . . . . . 18
ARTICLE II TERM . . . . . 19
Section 2.01 - Term of Agreement . . . . . 19
ARTICLE III
```

text splitting function

```
# Function to split text by tokens using tiktoken
def split_text_by_tokens(text, token_limit):
    tokenizer = tiktoken.get_encoding("cl100k_base")
    tokens = tokenizer.encode(text)
    chunks = []
    for i in range(0, len(tokens), token_limit):
        chunk = tokens[i:i + token_limit]
        chunks.append(tokenizer.decode(chunk))
    return chunks
doc_chunk = split_text_by_tokens(document_text, token_limit=512)
```

test if text split works

```
print("Number of Chunks Created:", len(doc_chunk))
print("\nSample Chunk (First 100 characters of the first chunk):\n")
print(doc_chunk[62][:100]) # Print the first 100 characters of the first chunk
```

```
➦ Number of Chunks Created: 137

Sample Chunk (First 100 characters of the first chunk):

shall the projection of Landing Fees,
Terminal Area Use Charges or Fueling System Fees of any Airli
```

DO NOT RUN THIS! Create the embeddings //do not use this approach! plan changed since open-ai is expensive

```
# expensive
# Initialize OpenAI API key
openai.api_key = "" # Replace with your OpenAI API key

# Function to generate embeddings for text chunks using the new API interface
def get_embeddings(texts):
    response = openai.Embedding.create(
        model="text-embedding-ada-002", # Use the correct embedding model
        input=texts
    )
    # Extract and return the embeddings
    return [embedding["embedding"] for embedding in response["data"]]

# Example: Generate embeddings for your text chunks
embeddings = get_embeddings(doc_chunk)
```

Use open resources embeddings instead of openai

```
# Load a pre-trained model
model = SentenceTransformer('all-MiniLM-L6-v2') # You can choose another model if needed

# Generate embeddings for your text chunks
embeddings = model.encode(doc_chunk)

print("Generated embeddings using free model successfully!")
```

```
➦ /usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_tokenization_
warnings.warn(
Generated embeddings using free model successfully!
```

create vector database with the embeddings + keep the original chunk for the key words matching method

```
# Initialize the FAISS index for vector-based retrieval
dimension = len(embeddings[0]) # The dimension of the embeddings
index = faiss.IndexFlatL2(dimension) # Using L2 distance for similarity search

# Convert embeddings to a NumPy array and add to the index
index.add(np.array(embeddings, dtype=np.float32))
print("Embeddings added to FAISS index successfully!")
```

Embeddings added to FAISS index successfully!

vectorize user query(questions)

```
# Function to vectorize the user query
def vectorize_query(query):
    # Use the same model (sentence-transformers) to create the embedding
    query_embedding = model.encode([query])[0]
    return np.array([query_embedding], dtype=np.float32) # Convert to NumPy array

# Function to retrieve top-k relevant chunks from the vector database
def retrieve_top_k_chunks(query_embedding, top_k=3):
    distances, indices = index.search(query_embedding, top_k) # Search FAISS index
    top_chunks = [doc_chunk[i] for i in indices[0]] # Retrieve the corresponding text chunks
    return top_chunks

combine the top-k related words chunk from the document and give chatgpt a new prompt (QA+related info)

# Function to format the context for ChatGPT
def format_context_for_chatgpt(user_query, retrieved_chunks):
    context = "\n".join(retrieved_chunks) # Combine the chunks into a single string
    prompt = f"Question: {user_query}\n\nContext:\n{context}\n\nAnswer:"
    return prompt

# Function to get an answer from ChatGPT
openai.api_key = "sk-proj-1IM7HGmYFS5ifhZXfDulULKR46BERjEyELI4V4Jl2Accq6pHU7DwqlJ8WwKpEHNwioLWc8oCGIT3BlbkFJAnojrHpJ-zW9nkQBEVJ0"
def get_chatgpt_response(prompt):
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo", # Use the new model
        messages=[
            {"role": "system", "content": "You are a helpful assistant."},
            {"role": "user", "content": prompt}
        ],
        max_tokens=300, # Adjust the max_tokens as needed
        temperature=0.7 # Adjust temperature for more or less creative answers
    )
    return response['choices'][0]['message']['content'].strip()

# Complete RAG workflow function
def rag_chatbot(user_query, top_k=5):
    # Step 1: Vectorize the user query
    query_embedding = vectorize_query(user_query)

    # Step 2: Retrieve top-k relevant chunks
    retrieved_chunks = retrieve_top_k_chunks(query_embedding, top_k)

    # Step 3: Format the context for ChatGPT
    prompt = format_context_for_chatgpt(user_query, retrieved_chunks)

    # Step 4: Get the response from ChatGPT
    answer = get_chatgpt_response(prompt)

    return answer

# Example usage
user_question = "What are the key concepts discussed in the document?"
answer = rag_chatbot(user_question)
print(f"Answer from ChatGPT:\n{answer}")
```

plan. These concepts cover various aspects related to the agreement and operations at Chicago-O'Hare International Airport.

