**Chat with PDF Using RAG Pipeline**

**1. Introduction**

* **Overview**

**2. System Requirements**

* **Hardware Requirements**
* **Software Requirements**
* **Dependencies**

**3. Installation Guide**

* **Installing Dependencies**
* **Setting Up Environment**

**4. Implementation Details**

* **Step-by-Step Process**
  + **Data Extraction**
  + **Text Chunking**
  + **Embedding Text**
  + **Storing Embeddings**
  + **Query Processing**
  + **Response Generation**

**5. Code Explanation**

* **Embedding Model Initialization**
* **OpenAI API Integration**
* **Text Chunking Function**
* **Query Embedding Function**
* **Similarity Search with FAISS**
* **Response Generation with OpenAI**
* **Web Scraping with Scrapy**

**6. Running the Pipeline**

* **How to Run the Script**
* **User Interaction and Query Handling**
* **Viewing Results**

**7. Example Usage**

* **Sample Data and Queries**
* **Expected Output**

**8. Troubleshooting**

* **Common Issues**
* **Solutions and Workarounds**

**9.Actual code:**

* **Code**

**10. Best Practices**

* **Optimizing Performance**
* **Enhancing Efficiency**

**11. Further Reading**

* **Relevant Articles and Papers**
* **Additional Resources**

**12. Conclusion**

* **Summary**
* **Future Work and Improvements**

**Overview**

The goal of this script is to build a pipeline that scrapes text data from university websites, chunks and embeds the text, stores it in a vector database for efficient retrieval, and generates responses based on user queries using OpenAI's API.

**Purpose and Scope:**

The purpose of this project is to develop a system that can effectively manage and utilize text data extracted from university websites. It aims to answer user queries by leveraging advanced NLP techniques and OpenAI's language model.

**2.System Requirements:**

* **Hardware Requirements:**

 CPU: Multi-core processor

 RAM: Minimum 8GB

 Storage: Sufficient space to store scraped data and embeddings

* **Software Requirements:**

 Operating System: Windows, macOS, or Linux

 Python: Version 3.6 or above

**Dependencies:**

Python libraries: scrapy, sentence-transformers, numpy, faiss-cpu, openai, json, os

**3.Installation Guide:**

**Installing Dependencies**

Install the required libraries using pip:

pip install scrapy sentence-transformers numpy faiss-cpu openai

**Setting Up Environment**

Ensure you have Python installed. Create a virtual environment to manage dependencies:

python -m venv myenv

source myenv/bin/activate # On Windows, use myenv\Scripts\activate

**4.Implementation Details:**

**Step-by-Step Process:**

**Data Extraction:**

1. **Scrapy Spider**: Define a spider class to scrape text from specified university websites.
2. **Run Spider**: Execute the spider to collect text data and store it in a JSON file.

**Text Chunking**

1. **Chunk Text**: Break down the extracted text into manageable chunks for embedding.

**Embedding Text**

1. **Sentence Transformer**: Use a pre-trained model to convert text chunks into vector embeddings.

**Storing Embeddings**

1. **FAISS**: Store the embeddings in a FAISS index for efficient similarity searches.

**Query Processing**

1. **Query Embedding**: Convert user queries into embeddings.
2. **Similarity Search**: Perform a similarity search in the FAISS index to retrieve relevant text chunks.

**Response Generation**

1. **OpenAI**: Generate responses based on the retrieved text chunks using OpenAI's API.

**5. Code Explanation**

**Embedding Model Initialization**

from sentence\_transformers import SentenceTransformer

embedding\_model = SentenceTransformer('all-MiniLM-L6-v2')

**OpenAI API Integration**

import openai

openai.api\_key = 'YOUR\_API\_KEY' # Replace with your actual API key

**Text Chunking Function**

def chunk\_text(text, chunk\_size=200):

words = text.split()

return [' '.join(words[i:i + chunk\_size]) for i in range(0, len(words), chunk\_size)]

**Query Embedding Function**

def query\_embedding(query):

return embedding\_model.encode([query])[0]

**Similarity Search with FAISS**

import faiss

def search\_vector\_database(query, k=5):

query\_vec = np.array([query\_embedding(query)])

distances, indices = index.search(query\_vec, k)

return indices, distances

**Response Generation with OpenAI**

def generate\_response\_with\_openai(query, context):

response = openai.ChatCompletion.create(

model="gpt-4o-mini",

messages=[

{"role": "user", "content": f"Answer the following question based on the context: {context}\nQuestion: {query}"}

]

)

return response['choices'][0]['message']['content']

**Web Scraping with Scrapy**

import scrapy

from scrapy.crawler import CrawlerProcess

class UniversitySpider(scrapy.Spider):

name = 'university\_spider'

start\_urls = [

"https://www.uchicago.edu/",

"https://www.washington.edu/",

"https://www.stanford.edu/",

"https://und.edu/"

]

def parse(self, response):

text = ' '.join(response.css('p::text').getall())

yield {'url': response.url, 'content': text}

**6. Running the Pipeline**

**How to Run the Script**

Run the following script to start the pipeline:

if not os.path.exists('scraped\_data.json'):

process = CrawlerProcess(settings={

'FEED\_FORMAT': 'json',

'FEED\_URI': 'scraped\_data.json',

})

process.crawl(UniversitySpider)

process.start()

# Load the scraped data

with open('scraped\_data.json') as f:

scraped\_data = json.load(f)

# Further steps include chunking, embedding, and handling user queries as described above.

**User Interaction and Query Handling**

Prompt the user for a query and process the query to retrieve and generate responses

user\_query = input("Enter your query: ")

retrieved\_indices, \_ = search\_vector\_database(user\_query)

retrieved\_chunks = [metadata[i] for i in retrieved\_indices[0]]

retrieved\_text = "\n".join([chunked\_data[chunk['url']][chunk['chunk']] for chunk in retrieved\_chunks])

response = generate\_response\_with\_openai(user\_query, retrieved\_text)

print("Generated Response:", response)

**7. Example Usage**

**Sample Data and Queries**

* **Sample Data**: University website text data extracted by the Scrapy spider.
* **Sample Query**: "Tell me about the research facilities at Stanford University."

**Expected Output**

* **Retrieved Text**: Extracts relevant text chunks from the stored data.
* **Generated Response**: A coherent response generated by OpenAI's model based on the retrieved text.

**8. Troubleshooting**

**Common Issues**

* **Scrapy Spider Errors**: Ensure correct URLs and CSS selectors are used.
* **API Key Issues**: Verify the OpenAI API key is correctly set.

**Solutions and Workarounds**

* **Debugging**: Use logging to identify and resolve issues.
* **Documentation**: Refer to the official documentation of the libraries used for specific error handling.

**9.Actual code:**

**pip install openai**

**pip install llama-index transformers sentence-transformers faiss-cpu requests beautifulsoup4**

**pip install scrapy**

**pip install openai==0.28**

**import scrapy**

**from scrapy.crawler import CrawlerProcess**

**from sentence\_transformers import SentenceTransformer**

**import numpy as np**

**import faiss**

**import openai**

**import json**

**import os  # Import os for file checking**

**# Initialize the embedding model**

**embedding\_model = SentenceTransformer('all-MiniLM-L6-v2')**

**# Set your OpenAI API key**

**openai.api\_key = 'sk-proj-MpUbUqS3TJJlbPowZLfx5j5AgG5ogXa00ETqbUtF2Dn-9wF0dZtNmNZjC1jMtibr21qZ3Mqx0kT3BlbkFJefQ5G2lcj1r3e6TM9RHe3HQsY2X72Fvk2nokWzhM4fzhslo50\_hvWypN-oU\_2eCtvlOSrnWIAA'  # Replace with your actual API key**

**# Function to chunk text**

**def chunk\_text(text, chunk\_size=200):**

**words = text.split()**

**return [' '.join(words[i:i + chunk\_size]) for i in range(0, len(words), chunk\_size)]**

**# Function to query embedding**

**def query\_embedding(query):**

**return embedding\_model.encode([query])[0]**

**# Function to perform similarity search in FAISS**

**def search\_vector\_database(query, k=5):**

**query\_vec = np.array([query\_embedding(query)])**

**distances, indices = index.search(query\_vec, k)**

**return indices, distances**

**# Function to generate response using OpenAI API**

**def generate\_response\_with\_openai(query, context):**

**response = openai.ChatCompletion.create(**

**model="gpt-4o-mini",**

**messages=[**

**{"role": "user", "content": f"Answer the following question based on the context: {context}\nQuestion: {query}"}**

**]**

**)**

**return response['choices'][0]['message']['content']**

**class UniversitySpider(scrapy.Spider):**

**name = 'university\_spider'**

**start\_urls = [**

**"https://www.uchicago.edu/",**

**"https://www.washington.edu/",**

**"https://www.stanford.edu/",**

**"https://und.edu/"**

**]**

**def parse(self, response):**

**# Extract text from paragraphs**

**text = ' '.join(response.css('p::text').getall())**

**yield {'url': response.url, 'content': text}**

**# Run the Scrapy spider only if scraped\_data.json doesn't exist**

**if not os.path.exists('scraped\_data.json'):**

**process = CrawlerProcess(settings={**

**'FEED\_FORMAT': 'json',**

**'FEED\_URI': 'scraped\_data.json',**

**})**

**process.crawl(UniversitySpider)**

**process.start()  # The script will block here until the crawling is finished**

**# Load the scraped data**

**with open('scraped\_data.json') as f:**

**scraped\_data = json.load(f)**

**# Chunk and embed data**

**chunked\_data = {}**

**embeddings = []**

**metadata = []**

**for item in scraped\_data:**

**url = item['url']**

**content = item['content']**

**if content:**

**chunks = chunk\_text(content)**

**chunk\_embeddings = embedding\_model.encode(chunks)**

**embeddings.extend(chunk\_embeddings)**

**metadata.extend([{"url": url, "chunk": i} for i in range(len(chunks))])**

**chunked\_data[url] = chunks**

**embeddings = np.array(embeddings)**

**# Store embeddings in FAISS**

**dimension = embeddings.shape[1]**

**index = faiss.IndexFlatL2(dimension)**

**index.add(embeddings)**

**# User query**

**user\_query = input("Enter your query: ")**

**retrieved\_indices, \_ = search\_vector\_database(user\_query)**

**# Retrieve relevant chunks**

**retrieved\_chunks = [metadata[i] for i in retrieved\_indices[0]]**

**retrieved\_text = "\n ".join([chunked\_data[chunk['url']][chunk['chunk']] for chunk in retrieved\_chunks])**

**# Generate response using OpenAI**

**response = generate\_response\_with\_openai(user\_query, retrieved\_text)**

**# Print results**

**print("Retrieved Text:")**

**print(retrieved\_text)**

**print("\nGenerated Response:")**

**print(response)**

**# Optionally, print both responses together**

**print("\n--- Summary ---")**

**print(f"General Response: {retrieved\_text}")**

**print(f"Generated Response: {response}")**

**10. Best Practices**

**Optimizing Performance**

* **Parallel Processing**: Utilize multi-threading or multi-processing for faster data extraction and processing.
* **Efficient Storage**: Use FAISS's compression techniques to store embeddings more efficiently.

**Enhancing Efficiency**

* **Caching**: Implement caching mechanisms to store frequently used results.
* **Batch Processing**: Process data in batches to minimize computational overhead.

**11. Further Reading**

**Relevant Articles and Papers**

* **Retrieval-Augmented Generation (RAG)**: Research papers and articles on the concept and implementation of RAG.
* **FAISS Documentation**: Official documentation for the FAISS library.

**Additional Resources**

* **Sentence Transformers**: Documentation and examples for the Sentence Transformers library.
* **OpenAI API**: Official API documentation for integrating OpenAI's language models.

**12. Conclusion**

**Summary**

This documentation provides a comprehensive guide to implementing a Retrieval-Augmented Generation pipeline, from setting up the environment and dependencies to running the script and interacting with the system.

**Future Work and Improvements**

* **Enhance Accuracy**: Explore fine-tuning models on specific datasets for improved accuracy.
* **User Interface**: Develop a user-friendly interface for easier interaction with the system.