**Lab Exercise 9- Retrieval-Augmented Generation (RAG) model using Azure OpenAI**

**Objective:**

In this lab exercise, we will implement a **Retrieval-Augmented Generation (RAG)** model using **Azure OpenAI** and a simple retrieval mechanism on **Google Colab**. The idea behind **RAG** is to combine information retrieval with generative models to answer complex questions by retrieving relevant documents and generating responses based on that information.

**Lab Objective:**

To create a **Retrieval-Augmented Generation (RAG)** pipeline that:

1. Retrieves relevant documents or information from a knowledge base.
2. Uses the retrieved information to generate a response using **Azure OpenAI**.

**Prerequisites:**

1. **Azure OpenAI Access** with an API key.
2. **Google Colab** to run the code.
3. Install necessary Python packages.

**Step 1: Setup the Google Colab Environment**

You'll need to install the requests library for making API calls to Azure OpenAI and faiss-cpu for building the simple document retrieval system.

bash

Copy code

!pip install requests faiss-cpu

**Step 2: Create a Simple Document Store**

We'll use a small set of documents as a knowledge base for the retrieval part of the RAG pipeline. You can replace this with any other document database or even a vector store later on.

python

Copy code

import faiss

import numpy as np

from sklearn.feature\_extraction.text import TfidfVectorizer

# Sample documents (small knowledge base)

documents = [

"Python is a programming language.",

"The capital of France is Paris.",

"Azure is a cloud computing platform by Microsoft.",

"OpenAI provides powerful language models for various tasks.",

"The Great Wall of China is a historic landmark."

]

# Convert documents into TF-IDF vectors

vectorizer = TfidfVectorizer()

document\_vectors = vectorizer.fit\_transform(documents).toarray()

# Index the document vectors using FAISS (for fast retrieval)

index = faiss.IndexFlatL2(document\_vectors.shape[1])

index.add(np.array(document\_vectors).astype(np.float32))

# Function to retrieve the most relevant document

def retrieve(query, top\_n=1):

query\_vector = vectorizer.transform([query]).toarray()

distances, indices = index.search(query\_vector.astype(np.float32), top\_n)

return [documents[i] for i in indices[0]]

This code uses **TF-IDF** for encoding documents and **FAISS** for efficient similarity search.

**Step 3: Setup Azure OpenAI Client**

You'll need your **Azure OpenAI API Key** and **Endpoint** from your Azure portal. Replace the placeholders '<Your\_Azure\_API\_Key>' and '<Your\_Azure\_Endpoint>' with your actual credentials.

python

Copy code

import os

import requests

# Set up Azure OpenAI credentials

api\_key = "<Your\_Azure\_API\_Key>"

endpoint = "<Your\_Azure\_Endpoint>"

deployment\_name = "text-davinci-003" # Replace with your deployment name

# Create the full endpoint URL

url = f"{endpoint}/openai/deployments/{deployment\_name}/completions?api-version=2023-05-15"

# Set up headers

headers = {

"Content-Type": "application/json",

"api-key": api\_key

}

**Step 4: Build the RAG Pipeline**

Now, we'll build a simple RAG pipeline where a query is first passed through the document retrieval system, and then the retrieved document is used as context for Azure OpenAI to generate a final response.

python

Copy code

def rag\_pipeline(query):

# Step 1: Retrieve the most relevant document

retrieved\_docs = retrieve(query, top\_n=1)

context = " ".join(retrieved\_docs)

print("Retrieved Document:", context)

# Step 2: Use Azure OpenAI to generate an answer using the retrieved document

prompt = f"Based on the following context, answer the question: {query}\n\nContext: {context}"

# Create the payload

payload = {

"prompt": prompt,

"max\_tokens": 100,

"temperature": 0.7

}

# Send the request to Azure OpenAI API

response = requests.post(url, headers=headers, json=payload)

if response.status\_code == 200:

result = response.json()

return result['choices'][0]['text'].strip()

else:

return f"Error: {response.status\_code}, {response.text}"

**Step 5: Testing the RAG Pipeline**

You can now test the RAG pipeline by providing queries, and the agent will retrieve the relevant document and generate a response.

python

Copy code

query = "What is Azure?"

response = rag\_pipeline(query)

print("Final Response:", response)

**Example Output:**

* **Query**: "What is Azure?"
* **Retrieved Document**: "Azure is a cloud computing platform by Microsoft."
* **Final Response**: "Azure is a cloud computing platform by Microsoft that provides various services such as computing, analytics, storage, and networking."

**Summary of Steps:**

1. **Document Store**: We created a small knowledge base using a set of documents and encoded them using TF-IDF vectors.
2. **Retrieval**: Used **FAISS** for retrieving the most relevant document based on the query.
3. **Azure OpenAI Integration**: We called the Azure OpenAI API to generate a final response using the retrieved document as context.
4. **Testing**: We tested the pipeline with example queries.

**Further Enhancements:**

* You could replace the document store with a more sophisticated vector store such as **Azure Cognitive Search** or **Pinecone**.
* Improve the retrieval method by using dense embeddings like BERT or OpenAI’s own embeddings.
* Add multi-document retrieval and let the model generate responses based on more than one source.

This simple lab exercise demonstrates how **RAG** can be implemented in **Google Colab** with **Azure OpenAI** to enhance the accuracy and relevance of responses by leveraging retrieval-based information.