**Practical – 2**

**Retrieval-Augmented Generation (RAG) Application Using PDF READ+LANGCHAIN+ BEDROCK+LLAMA**

**Retrieval-Augmented Generation (RAG)** is a technique that enhances large language models (LLMs) by retrieving relevant information from external knowledge sources before generating a response. Instead of relying solely on pre-trained knowledge, RAG enables the model to fetch real-time data from a vector database, improving accuracy and relevance, especially for domain-specific tasks.

In our project, we implement a **RAG-based system** using AWS Bedrock and LangChain. The code processes a **PDF document**, extracts its text using **PyPDF**, and converts it into vector embeddings using **Amazon Titan embeddings**. These embeddings are stored in a **FAISS vector database** for efficient similarity search. When a user asks a question, the system retrieves the most relevant document sections from the FAISS index and passes them to **Llama 3 (via AWS Bedrock)** to generate an informed response. This ensures that the model provides contextually relevant answers based on the uploaded PDF content.

**Requirements.txt**

boto3

botocore

awscli

pypdf

langchain

faiss-cpu

langchain-community

langchain-aws

**Technologies and Libraries Used**

* **AWS Services:** Amazon S3, AWS Bedrock, IAM Permissions
* **LangChain**
  + LangChain is a framework designed to help build applications powered by large language models (LLMs).
  + It simplifies working with AI models, vector stores (like FAISS), and retrieval-augmented generation (RAG) workflows.
  + In your project, LangChain is used to manage embeddings, query processing, and interactions with AWS Bedrock.
* **PyPDF**
  + PyPDF is a Python library for reading, modifying, and extracting text from PDF files.
  + In your project, it is used to parse the uploaded PDF and convert it into text that can be embedded into a vector store.
* **FAISS (Facebook AI Similarity Search)**
  + FAISS is an open-source library for efficient similarity search and clustering of dense vectors.
  + It is used to store and retrieve document embeddings quickly based on user queries.
* **Boto3 (AWS SDK for Python)**
  + Boto3 is the official Amazon Web Services (AWS) SDK for Python.
  + It enables communication with AWS services, such as S3 (for storing PDFs) and Bedrock (for querying Llama 3).

**Steps Followed**

**1. Setting Up the AWS Environment**

* Created an **S3 bucket** to store the PDF documents.
* Configured an **AWS IAM role** with appropriate permissions for Bedrock and S3.
* Initialized AWS credentials using boto3.

**2. Installing Required Libraries**

* Installed langchain, langchain\_community, langchain\_aws, faiss-cpu, and boto3.
* Activated a **Conda virtual environment** and ensured dependencies were installed.

**3. Loading and Processing the PDF Document**

* Used pypdf to extract text from a local PDF file.
* Processed the extracted text into manageable chunks for embedding.

**4. Generating Vector Embeddings with AWS Bedrock**

* Used BedrockEmbeddings from langchain\_aws to generate vector embeddings.
* Stored the vectors in **FAISS index** for efficient retrieval.

**5. Querying the Document with Llama 3**

* Loaded the FAISS index and retrieved relevant document chunks.
* Passed the retrieved text as context to the **Llama 3 8B Instruct v1** model.
* Displayed the response in the **VS Code terminal**.

**2. Issues Encountered**

**1. ModuleNotFoundError for langchain\_community**

* The import statement for BedrockEmbeddings was outdated.
* Fixed by replacing from langchain.llms import Bedrock with from langchain\_community.llms import Bedrock.

**2. FAISS Index Not Found Error**

* The script attempted to load a FAISS index that did not exist.
* Fixed by ensuring that the FAISS index was created before loading.

**3. allow\_dangerous\_deserialization Warning in FAISS**

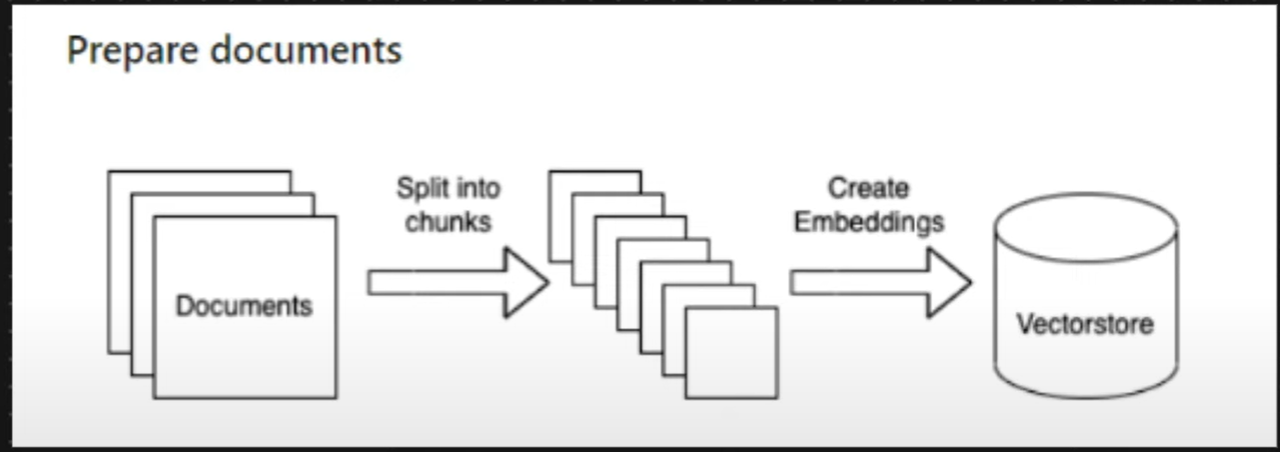
* FAISS requires explicit permission to deserialize pickle files.
* Fixed by setting allow\_dangerous\_deserialization=True in FAISS.load\_local().

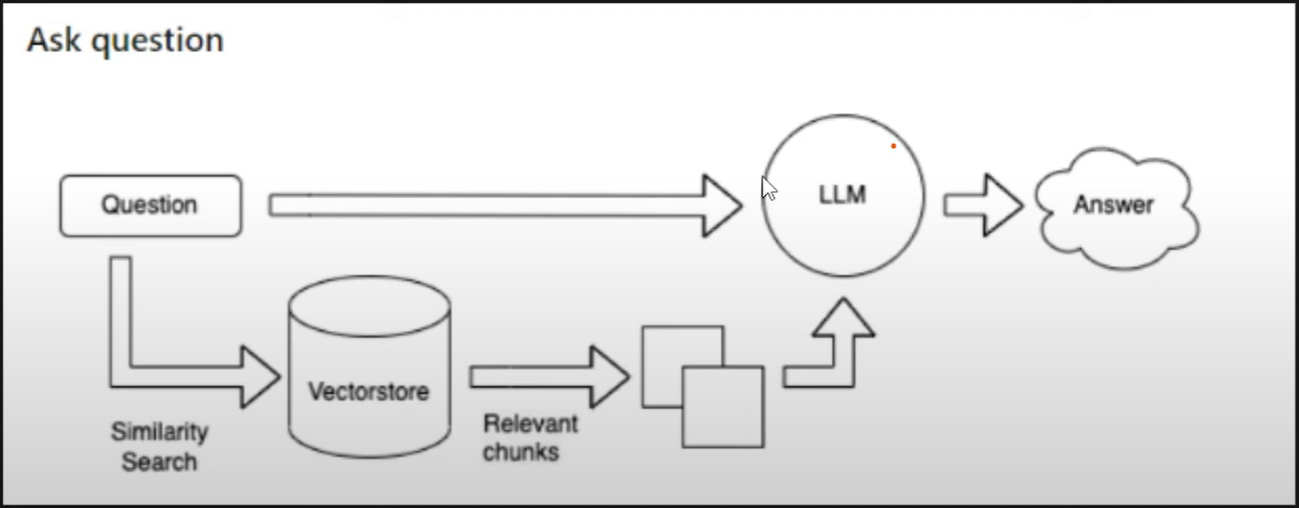
**4. AccessDeniedException When Using AWS Bedrock**

* The IAM role did not have permission to invoke the Llama 3 model.
* Fixed by adding bedrock:InvokeModel permissions in the IAM policy.

**5. No PDF Found in Local Directory**

* The script could not find the PDF file to process.
* Fixed by placing the PDF file in the correct directory before running the script.





**Explanation of Data Flow**

1. **User Uploads PDF → The document is uploaded to an S3 bucket for cloud storage.**
2. **PDF is Read → The text is extracted using PyPDF or similar libraries.**
3. **Convert Text to Embeddings → The text is converted into numerical representations using Amazon Titan Embeddings.**
4. **Store in FAISS → The embeddings are stored in a FAISS (Facebook AI Similarity Search) index for fast retrieval.**
5. **User Asks a Question → The user enters a query, which is processed by Llama 3.**
6. **Retrieve Similar Chunks → The system searches the FAISS index for the most relevant information.**
7. **Generate Response → The Llama 3 model uses the retrieved information to generate a final answer.**
8. **Response Sent to User → The generated answer is displayed to the user.**

**Embedding and Vectorization Explained**

**Embedding** and **vectorization** are techniques used to convert text (or other types of data) into a numerical format that machine learning models can understand. They are essential in **Natural Language Processing (NLP)**, **information retrieval**, and **AI-powered applications** like your RAG system.

**1️⃣ What is Embedding?**

* **Embedding** is the process of converting words, sentences, or entire documents into **dense numerical representations (vectors)** in a multi-dimensional space.
* These embeddings **capture semantic meaning**, so similar words or concepts are mapped **closer** together in this space.

**Example of Word Embeddings:**

| **Word** | **3D Embedding (Example)** |
| --- | --- |
| Apple | [0.9, 0.1, 0.3] |
| Banana | [0.8, 0.2, 0.4] |
| Car | [0.1, 0.9, 0.7] |

🔹 Here, **"Apple" and "Banana"** are closer in the vector space because they are both fruits, while **"Car"** is farther away.

🛠️ **Embedding Models Used in Your Project:**

* **Amazon Titan Embeddings** → Converts PDF text into embeddings.
* **FAISS (Facebook AI Similarity Search)** → Stores and retrieves these embeddings efficiently.

**2️⃣ What is Vectorization?**

* **Vectorization** is a broader term that refers to transforming any type of data (text, images, audio) into numerical vectors.
* It is the **step before embedding** in some cases.

**Types of Vectorization in NLP:**

1. **TF-IDF (Term Frequency-Inverse Document Frequency)** – Basic technique, weights words based on importance.
2. **Word2Vec / FastText** – Creates word embeddings based on context.
3. **Transformer-based Embeddings (like BERT, Titan, Llama 3)** – More advanced, understands entire sentences and context.

**Difference Between Embedding and Vectorization**

| **Feature** | **Embedding** | **Vectorization** |
| --- | --- | --- |
| **Purpose** | Capture meaning | Convert text into numbers |
| **Output** | Dense vector | Sparse/dense vector |
| **Example Methods** | Word2Vec, Titan, BERT | TF-IDF, One-hot encoding |
| **Used in** | NLP, AI, Search engines | Basic text processing |

**How It Works in Your Project**

1. **PDF text is extracted and converted into embeddings** using **Amazon Titan**.
2. **FAISS stores these embeddings** and retrieves relevant vectors when a user asks a question.
3. **Llama 3 generates a response** using the retrieved embeddings, making the system efficient in answering queries based on PDF content.

🔹 **Conclusion:** Embeddings **understand meaning**, while vectorization is the **process of representing data numerically**. Your RAG system **relies on embeddings** to find relevant information before generating a response. 🚀

**Python code :-**

import json

import os

import boto3

from langchain\_aws.embeddings import BedrockEmbeddings  # Requires `pip install langchain-aws`

from langchain\_community.llms import Bedrock

from langchain.text\_splitter import RecursiveCharacterTextSplitter

from langchain\_community.document\_loaders import PyPDFDirectoryLoader

from langchain\_community.vectorstores import FAISS

from langchain.prompts import PromptTemplate

from langchain.chains import RetrievalQA

# Initialize AWS Bedrock client

bedrock = boto3.client(service\_name="bedrock-runtime", region\_name="us-east-1")

# Titan Embedding Model

bedrock\_embeddings = BedrockEmbeddings(model\_id="amazon.titan-embed-text-v2:0", client=bedrock)

# Data ingestion function

def data\_ingestion():

    print("Loading PDF documents...")

    loader = PyPDFDirectoryLoader("data")  # Ensure the PDF files are inside the 'data' folder

    documents = loader.load()

    text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=10000, chunk\_overlap=1000)

    docs = text\_splitter.split\_documents(documents)

    return docs

# Create and save FAISS vector store

def get\_vector\_store(docs):

    print("Creating vector embeddings...")

    vectorstore\_faiss = FAISS.from\_documents(docs, bedrock\_embeddings)

    vectorstore\_faiss.save\_local("faiss\_index")

    print("Vector store created successfully!")

# Initialize Llama 3 model

def get\_llama3\_llm():

    llm = Bedrock(

        model\_id="meta.llama3-8b-instruct-v1:0",

        client=bedrock,

        model\_kwargs={"max\_gen\_len": 512}

    )

    return llm

# Define prompt template

prompt\_template = """

Human: Use the following context to answer the question.

Provide a concise yet detailed response of at least 250 words.

If you don't know the answer, say that you don't know.

<context>

{context}

</context>

Question: {question}

Assistant:

"""

PROMPT = PromptTemplate(template=prompt\_template, input\_variables=["context", "question"])

# Retrieve response from Llama 3

def get\_response\_llm(llm, vectorstore\_faiss, query):

    qa = RetrievalQA.from\_chain\_type(

        llm=llm,

        chain\_type="stuff",

        retriever=vectorstore\_faiss.as\_retriever(search\_type="similarity", search\_kwargs={"k": 3}),

        return\_source\_documents=True,

        chain\_type\_kwargs={"prompt": PROMPT}

    )

    answer = qa({"query": query})

    return answer['result']

def main():

    print("Chat with PDF using AWS Bedrock (Llama 3)")

    # Create vector store if needed

    if input("Do you want to create/update the vector store? (yes/no): ").strip().lower() == "yes":

        docs = data\_ingestion()

        get\_vector\_store(docs)

    # Load FAISS index

    print("Loading vector store...")

    # error 1faiss\_index = FAISS.load\_local("faiss\_index", bedrock\_embeddings)

    # error 2faiss\_index = FAISS.load\_local("faiss\_index", bedrock\_embeddings, allow\_dangerous\_deserialization=True)

    if not os.path.exists("faiss\_index/index.faiss"):

        print("FAISS index not found! Creating new vector store...")

        docs = data\_ingestion()  # Ingest data again

        get\_vector\_store(docs)  # Generate FAISS index

    # Now, load the FAISS index safely

    faiss\_index = FAISS.load\_local("faiss\_index", bedrock\_embeddings, allow\_dangerous\_deserialization=True)

    # Initialize Llama 3

    llm = get\_llama3\_llm()

    while True:

        user\_question = input("\nEnter your question (or type 'exit' to quit): ")

        if user\_question.lower() == "exit":

            print("Exiting...")

            break

        print("\nProcessing your query...")

        response = get\_response\_llm(llm, faiss\_index, user\_question)

        print("\nLlama 3 Response:\n")

        print(response)

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

    main()

