**Set Up FastAPI**

First, create a new directory for your project and initialize a Python environment.

mkdir fastapi-rag-classification

cd fastapi-rag-classification

python -m venv venv

source venv/bin/activate

pip install fastapi uvicorn transformers scikit-learn

**Create the FastAPI Application**

Create a file named main.py to define the FastAPI server.

from fastapi import FastAPI, HTTPException

from pydantic import BaseModel

from transformers import pipeline, AutoModelForSeq2SeqLM, AutoTokenizer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

import numpy as np

import json

app = FastAPI()

class Prompt(BaseModel):

text: str

class TextData(BaseModel):

text: str

model\_name = "facebook/bart-large-cnn"

tokenizer = AutoTokenizer.from\_pretrained(model\_name)

model = AutoModelForSeq2SeqLM.from\_pretrained(model\_name)

summarization\_pipeline = pipeline("summarization", model=model, tokenizer=tokenizer)

dataset = [

{"text": "Anxiety is a normal part of life, but anxiety disorders are different.", "category": "anxiety"},

{"text": "Depression is a common and serious medical illness that negatively affects how you feel.", "category": "depression"},

{“text”:”the study of the structure of body parts closely related to physiology , which focuses on other body parts related to each other”.”category”:”physiology”.

]

vectorizer = TfidfVectorizer()

X = vectorizer.fit\_transform([item["text"] for item in dataset])

y = np.array([item["category"] for item in dataset])

classifier = LogisticRegression()

classifier.fit(X, y)

@app.post("/rag")

async def rag(prompt: Prompt):

# Generate a response using the summarization pipeline

try:

inputs = tokenizer.encode(prompt.text, return\_tensors="pt", max\_length=512, truncation=True)

outputs = model.generate(inputs, max\_length=150, min\_length=40, length\_penalty=2.0, num\_beams=4, early\_stopping=True)

summary = tokenizer.decode(outputs[0])

relevant\_articles = [item for item in dataset if prompt.text.lower() in item["text"].lower()]

return {"summary": summary, "articles": relevant\_articles}

except Exception as e:

raise HTTPException(status\_code=500, detail=str(e))

@app.post("/classification")

async def classify(text\_data: TextData):

try:

X\_new = vectorizer.transform([text\_data.text])

prediction = classifier.predict(X\_new)[0]

return {"category": prediction}

except Exception as e:

raise HTTPException(status\_code=500, detail=str(e))

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

import uvicorn

uvicorn.run(app, host="0.0.0.0", port=8000)

**Dockerize the Application**

Create a Dockerfile to containerize the application.

FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt requirements.txt

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

Create requirements.txt

**Specify the required Python packages.**

fastapi

uvicorn

transformers

scikit-learn

**Build and Run the Docker Container**

docker build -t fastapi-rag-classification .

docker run -p 8000:8000 fastapi-rag-classification

**Create a Hugging Face Space:**

To deploy on Hugging Face Spaces

Create a new space on Hugging Face.

Upload your project files, including main.py, requirements.txt, and Dockerfile.

Configure the space to use Docker by creating a README.md with the following content:

---

title: FastAPI RAG and Classification

emoji: 🚀

colorFrom: pink

colorTo: spicejet

sdk: docker

pinned: false

---

# FastAPI RAG and Classification

This space demonstrates a FastAPI application with RAG and text classification endpoints.

**Optimize the Response Time**

**For optimization, consider the following:**

Use pre-trained model weights efficiently by caching them.

Optimize your Docker image by using a smaller base image.

Limit the input size for the LLM to reduce processing time.

Use asynchronous processing for heavy tasks.

To build a Retrieval-Augmented Generation (RAG) application for a mental health chatbot,

We need to combine two main components:

Retrieval: Finding relevant articles or blog posts related to the user's prompt.

Generation: Generating a response using a large language model (LLM) to provide helpful insights based on the user's prompt and the retrieved articles.

**Steps for Optimization and Productionization**

Optimize the TF-IDF Vectorizer:

Use a more sophisticated retrieval method such as BM25, or a semantic search model like Sentence-BERT.

Caching:

Cache model and vectorizer loads if the server restarts frequently.

Asynchronous Processing:

Ensure FastAPI endpoints use asynchronous processing to handle concurrent requests efficiently

.Deploying on Cloud: Deploy the Docker container on cloud services like AWS, GCP, or Azure for better scalability.

Testing the Application:

To test the application, send a POST request to the /rag endpoint with a JSON body containing a prompt.{

"text": "I feel very anxious and stressed."

}