**RAG-Based Semantic Quote Retrieval and Structured QA**

**Overview**

This project builds a **semantic quote search engine** using **RAG (Retrieval-Augmented Generation)**. It takes a natural language query (e.g., "quotes about courage by women authors") and returns a structured JSON with relevant quote, author, tags, and summary.

**Technologies Used**

* **Python** for all core scripting
* **Sentence-Transformers** for generating dense embeddings
* **FAISS** for fast similarity-based retrieval
* **OpenRouter** for using Mistral or LLaMA2 open LLMs
* **Gradio** for interactive UI (chosen over Streamlit for Colab compatibility)
* **RAGAS** for evaluating the quality and relevance of RAG-generated answers

**Code Walkthrough & Explanation**

**1. Data Loading and Preprocessing (data\_preparation.py)**

We load the dataset using HuggingFace's datasets library and then clean and format it for our embedding model.

import pandas as pd

# Load the dataset

# This dataset contains quote, author, and tags

df = pd.read\_json("hf://datasets/Abirate/english\_quotes/quotes.jsonl", lines=True)

# Drop nulls and duplicates to ensure clean input

df.dropna(inplace=True)

df.drop\_duplicates(subset=["quote"], inplace=True)

# Combine fields to form a single string for embeddings

# This gives our model context including author and tags

df["combined"] = df.apply(lambda x: f"{x['quote']} - {x['author']} | Tags: {' '.join(x['tags'])}", axis=1)

**2. Sentence Embedding & Vector Indexing (embed\_and\_index.py)**

We use sentence-transformers to encode the cleaned data into dense vector format and store them in a FAISS index for fast retrieval.

from sentence\_transformers import SentenceTransformer

import faiss

# Load a general-purpose embedding model

model = SentenceTransformer('all-MiniLM-L6-v2')

# Generate embeddings from the combined text fields

embeddings = model.encode(df['combined'].tolist(), show\_progress\_bar=True)

# Build FAISS index for L2 distance

faiss\_index = faiss.IndexFlatL2(384)

faiss\_index.add(embeddings)

# Save index and model for later use

faiss.write\_index(faiss\_index, "quotes\_index.faiss")

model.save("models/fine-tuned-model")

**3. RAG Pipeline (app.py)**

We create a full RAG loop by:

1. Accepting a query
2. Retrieving top-k similar quotes
3. Passing those as context to an LLM (via OpenRouter)
4. Returning a structured answer using JSON format

import openai

import gradio as gr

import faiss

from sentence\_transformers import SentenceTransformer

from dotenv import load\_dotenv

import os

# Load API key from environment

load\_dotenv()

openai.api\_key = os.getenv("OPENROUTER\_API\_KEY")

# Load model and FAISS index

model = SentenceTransformer("models/fine-tuned-model")

index = faiss.read\_index("quotes\_index.faiss")

# Inference function

# Takes natural language query -> FAISS match -> LLM structured output

def query\_rag(user\_input):

user\_vec = model.encode([user\_input])

D, I = index.search(user\_vec, 1) # Top-1 result

# Get the matched quote details

match = df.iloc[I[0][0]]

context = f"Quote: {match['quote']}. Author: {match['author']}. Tags: {', '.join(match['tags'])}"

# Prompt LLM to convert into JSON format

prompt = f"Using the quote context below, generate a structured JSON with keys quote, author, tags, summary.\nContext: {context}"

response = openai.ChatCompletion.create(

model="mistral",

messages=[{"role": "user", "content": prompt}]

)

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

# Gradio Interface (for Colab compatibility)

gr.Interface(fn=query\_rag, inputs="text", outputs="json").launch()

**4. RAG Evaluation with RAGAS (evaluate\_rag.py)**

We use the RAGAS framework to evaluate:

* **Faithfulness**: Whether the LLM output sticks to retrieved context
* **Answer Relevancy**: Is the generated answer relevant to the question?
* **Context Recall**: Does the context used contain all relevant facts?

from ragas.metrics import faithfulness, answer\_relevancy, context\_recall

from ragas.evaluation import evaluate

# Simulated test set - create your own based on known truth

rag\_dataset = [

{

"question": "Quotes about courage",

"answer": "Life shrinks or expands in proportion to one's courage.",

"contexts": ["Life shrinks or expands in proportion to one's courage. - Anais Nin"]

},

# Add more entries for comprehensive scoring

]

# Evaluate RAG performance

result = evaluate(rag\_dataset, metrics=[faithfulness, answer\_relevancy, context\_recall])

print(result)

**Final Output (Example)**

**User Input:**

"Show me quotes about courage by women authors"

**Returned JSON:**

{

"quote": "Life shrinks or expands in proportion to one's courage.",

"author": "Anaïs Nin",

"tags": ["life", "courage"],

"summary": "Anaïs Nin emphasizes that courage determines the scope of life."

}