Project Title: Building a Customized Question-Answering System using RAG (Implementation Video Link: Link)

Name: Mrinal Kanti Mukherjee

Role: Data Science Intern

Team Lead: Manjesh Kumar Mishra

Product Manager: Devyani Vidhate

1. Introduction

This project implements a Retrieval-Augmented Generation (RAG) pipeline to develop a customized question-answering system that retrieves and processes city-related information from various sources. The system integrates semantic search, external data retrieval, and large language model (LLM)-based response generation to provide fact-based, contextually relevant answers to user queries.

2. Workflow of the RAG Pipeline (RAG.ipynb)

The RAG pipeline follows a structured approach to process user queries and generate responses. The workflow consists of the following steps:

1. Data Preprocessing: Prepare and index city-related data.
2. Query Processing: Convert user queries into embeddings and retrieve relevant cities.
3. Pass Retrieved Data to LLM: Use Wikipedia & OpenStreetMap data to enrich the response.
4. Display Final Response: Format and return a well-structured answer to the user.

2.1 Data Preprocessing

- Load city data from major_cities.json.
- Generate vector embeddings using Sentence Transformers.
- Store embeddings in FAISS for fast semantic retrieval.

2.2 Query Processing

- Convert user query into vector embeddings.
- Use FAISS to find the top-3 most relevant city matches.
- Retrieve supporting documents from Wikipedia & OpenStreetMap.

2.3 Pass Retrieved Data to LLM

- Combine retrieved information into a structured prompt.
- Call Google Gemini AI API to generate a response.

2.4 Display Final Response

- Format response for text display.
- Send response to the frontend UI for user display.

3. Backend Implementation (app.py)

The backend is implemented using Flask and serves as an API layer for handling requests from the frontend.

3.1 Features of app.py

- Handles user queries via a /query API endpoint.
- Uses FAISS for efficient similarity-based retrieval.
- Fetches real-world data from Wikipedia & OpenStreetMap.
- Calls the LLM API (Google Gemini AI) for answer generation.
- Returns JSON responses to the frontend.

3.2 API Workflow

- The user submits a query via the frontend.
- The backend converts it into a vector embedding.
- FAISS retrieves the most relevant city.
- The system fetches Wikipedia & OpenStreetMap data.
- The LLM API generates a structured response.
- The final response is sent back to the frontend.

4. Frontend Implementation (index.html)

The frontend provides an interactive user interface for submitting queries and displaying responses.

4.1 Features of index.html

- Search Box: Allows users to input queries.
- AJAX API Calls: Sends requests to app.py for processing.
- Dynamic Response Display: Shows answers in a structured format.

4.2 User Interaction Flow

1. The user enters a query in the search bar.
2. The request is sent to the Flask backend via AJAX.
3. The system processes the query and returns a JSON response.
4. The UI displays the formatted answer dynamically.

Conclusion :

This project successfully implements a customized RAG-based question-answering system for city-related queries. By integrating semantic retrieval (FAISS), knowledge retrieval (Wikipedia & OpenStreetMap), and response generation (LLM APIs), it provides fact-based, contextually accurate answers in real time.