**LANGCHAIN-BASED ASSISTANT FOR MOVIE KNOWLEDGE RETRIEVAL**

***INTRODUCTION:***

In the evolving landscape of intelligent systems, the ability to extract meaningful insights from unstructured and semi-structured data is becoming essential. This project aims to bridge the gap between raw movie metadata and human-like conversational interfaces.

Built upon the sample\_mflix dataset from MongoDB Atlas, this system transforms NoSQL collections into a well-structured SQL relational database and applies modern Natural Language Processing (NLP) techniques to create a seamless, intelligent user experience. By leveraging OpenAI embeddings, LangChain’s RAG architecture, and a FAISS-based vector store, the system empowers users to ask natural language questions about movies and receive context-aware answers in real-time.

This end-to-end pipeline demonstrates critical skills in data engineering, embedding generation, retriever-augmented generation (RAG), and LLM-based response generation, all deployed through a user-friendly interface built using Streamlit.

***PROBLEM STATEMENT:***

With the exponential growth of multimedia content and user interactions on streaming platforms, efficiently retrieving relevant information from vast movie databases has become a complex challenge. Traditional keyword-based search systems often fail to deliver accurate, context-aware responses to user queries, especially when the data is scattered across unstructured formats like JSON documents in NoSQL databases.

This project addresses the challenge of transforming unstructured movie metadata into a structured, searchable, and intelligent question-answering system. By integrating a Retrieval-Augmented Generation (RAG) framework, the system can provide accurate and conversational responses to natural language queries using information derived from the **sample\_mflix** MongoDB dataset. The solution also involves migrating data into a relational SQL format, generating semantic embeddings, and building a responsive interface that mimics human-like interaction.

***OBJECTIVES:***

The primary objective of this project is to develop an intelligent movie-based question answering system by combining structured SQL data and unstructured natural language processing techniques using a Retrieval-Augmented Generation (RAG) architecture.

The specific goals include:

1. **Migrate Data from MongoDB to SQL:**  
   Transform unstructured JSON documents from the sample\_mflix MongoDB dataset into a relational SQL format with proper normalization and foreign key relationships.
2. **Design SQL Schema & Model Relationships:**  
   Create a robust relational schema to manage entities like movies, users, comments, sessions, and theaters efficiently.
3. **Preprocess and Embed Textual Content:**  
   Combine relevant fields (movie plot, cast, reviews) into coherent documents and generate semantic vector embeddings using OpenAI or Hugging Face models.
4. **Build a FAISS Vector Store:**  
   Index the vectorized movie documents to enable efficient semantic search and retrieval.
5. **Implement RAG using LangChain:**  
   Integrate LangChain to enable retrieval of relevant movie chunks and generate context-aware responses using a large language model.
6. **Develop a Streamlit-Based QA Interface:**  
   Provide an interactive user interface where users can input natural language questions and receive real-time intelligent answers.
7. **Evaluate System Performance:**  
   Assess the system based on data accuracy, relevance of responses, interface usability, and retrieval latency.

***DATASET DESCRIPTION:***

The dataset used in this project is the **sample\_mflix** dataset, a publicly available movie dataset hosted on **MongoDB Atlas**. It simulates the backend of a movie streaming platform, containing comprehensive data across multiple collections related to movies, users, comments, and session details.

**Collections Used:**

* **movies** – Contains metadata such as title, plot, genre, cast, directors, languages, runtime, and release information.
* **comments** – User-submitted movie reviews linked to specific movies.
* **embedded\_movies** – Additional movie entries embedded within documents.
* **users** – User profiles, including login and identification data.
* **sessions** – Session details to simulate login behavior and activity.
* **theaters** – Geolocation and details of theaters, useful for linking screening locations.

**Dataset Characteristics:**

* **Source Format:** JSON documents (MongoDB)
* **Target Format:** Relational Tables (MySQL)
* **Total Collections Extracted:** 6
* **Size:** Medium-scale, real-world simulation
* **Domain Coverage:** Movies, reviews, user interactions, streaming platform behavior

**Preprocessing Done:**

* Flattened nested fields (e.g., cast, genres)
* Converted arrays into comma-separated strings
* Renamed columns to match SQL schema
* Handled nulls and missing fields
* Ensured data types were compatible with relational storage

***ENVIRONMENT CONFIGURATION (.env FILE):***

To ensure security, flexibility, and cleaner code management, all sensitive credentials and configuration variables are stored in a .env file. This file is loaded using the python-dotenv library and is excluded from version control to protect private information.

**Environment Variables Used**

| **Variable Name** | **Description** |
| --- | --- |
| mongodb\_uri | MongoDB Atlas connection string for extracting collections |
| db\_user | Username for the SQL (MySQL/PostgreSQL) database |
| db\_password | Password for the SQL database |
| db\_host | Host address for the SQL server (e.g., localhost or remote endpoint) |
| db\_name | Name of the target SQL database |
| api\_key\_1 to api\_key\_5 | API keys used for accessing OpenAI or other embedding services; helps rotate keys if token limit is reached |

***METHODOLOGY:***

The project follows a modular pipeline involving data migration, transformation, semantic embedding, and intelligent response generation using RAG architecture. The complete workflow is divided into the following phases:

**1. Data Extraction & Transformation**

* Extracted raw collections (movies, comments, users, sessions, theaters, embedded\_movies) from MongoDB Atlas using Python’s pymongo and custom extraction scripts.
* Flattened nested fields such as cast, genres, and languages into comma-separated strings.
* Transformed JSON documents into clean Pandas DataFrames with renamed, typed, and normalized fields.
* Designed a **relational SQL schema** that defines primary and foreign key relationships between tables.
* Loaded the cleaned data into a **MySQL** database using SQLAlchemy.

**2. Preprocessing & Embedding Generation**

* Merged key textual fields (title, plot, reviews, cast) into single document strings for each movie.
* Removed short or irrelevant entries and normalized text (case, symbols, etc.).
* Generated **semantic embeddings** using **OpenAI Embedding API**, representing each movie document in vector space.
* Stored the embeddings efficiently in a **FAISS vector store** for fast similarity-based retrieval.

**3. RAG Architecture Implementation**

* Utilized **LangChain** to implement Retrieval-Augmented Generation (RAG).
* Set up a **Retriever** using FAISS to fetch relevant text chunks based on user queries.
* Connected the retriever output to an LLM (OpenAI GPT model) to generate natural language answers using both the query and the retrieved context.
* Ensured that the architecture supports real-time query resolution with context relevance.

**4. QA Interface Development**

* Developed a simple and user-friendly front end using **Streamlit**.
* Built an input field for user queries and an output box to show real-time responses.
* Integrated backend logic to accept prompts, trigger embedding retrieval, pass context to the LLM, and display generated answers.
* Ensured proper UI layout, latency handling, and clarity of responses for an engaging user experience.

***EVALUATION METRICS:***

The system’s performance and effectiveness were assessed based on the following key metrics:

| **Metric** | **Description** |
| --- | --- |
| **Data migration completeness** | All collections from MongoDB were properly mapped, cleaned, and pushed into the SQL database. |
| **Embedding accuracy** | Ensures that semantically similar queries return contextually similar results from the vector store. |
| **Response relevance** | Evaluated based on human judgment for how precise and relevant the generated answers are to user queries. |
| **System latency** | Measures the time taken from the user’s prompt submission to the delivery of the final response. |
| **Usability of the interface** | Assessed based on user feedback regarding the ease of use, clarity, and responsiveness of the QA interface. |

***CONCLUSION:***

This project successfully demonstrates the integration of data engineering and natural language processing to build an intelligent, real-time movie question-answering system. Starting from unstructured MongoDB collections, the data was transformed into a structured SQL format, semantically vectorized, and queried using a Retrieval-Augmented Generation (RAG) approach with LangChain and OpenAI.

The system effectively retrieves relevant movie information and responds to natural language queries through an interactive interface. This end-to-end solution not only showcases the practical use of LLMs in media applications but also lays the foundation for more scalable AI-powered assistants across domains.

***TECHNICAL STACK & TOOLS USED:***

| **Category** | **Tools/Technologies** |
| --- | --- |
| **Database** | MongoDB (NoSQL), MySQL |
| **Data Migration** | Python (pymongo, SQLAlchemy, Pandas) |
| **Data Transformation** | Python (Data cleaning, flattening, schema mapping) |
| **Embeddings** | OpenAI Embedding API |
| **Vector Store** | FAISS (Facebook AI Similarity Search) |
| **RAG Framework** | LangChain |
| **Language Model** | OpenAI GPT (LLM for generating responses) |
| **Frontend UI** | Streamlit (Web-based QA interface) |
| **Environment Tools** | .env, dotenv, Git, |

***PROJECT FOLDER STRUCTURE:***

Project\_Final\_RAG\_QA/

│

├── main.py # Master script to trigger the full pipeline

├── chatbot.py # Final chatbot interface (optional if used)

├── .env # Environment variables for API keys & DB access

├── requirements.txt # Required Python packages

├── ER\_diagram.png # SQL schema visualization

├── project\_document.docx # Project report/documentation

├── project\_rag\_qa\_db.sql # SQL dump (optional, for DB setup)

│

├── source/

│ ├── etl/

│ │ ├── mongodb\_extract.py # MongoDB data extraction

│ │ ├── transform\_extracted\_data.py # Clean raw extracted data

│ │ └── load\_transformed\_data\_to\_sql.py # Load cleaned data into SQL

│ │

│ ├── processing/

│ │ ├── embeddings.py # Embedding generation using OpenAI

│ │ ├── load\_data\_from\_sql.py # Load back SQL data for embeddings

│ │ ├── process\_data.py # Preprocessing and combination

│ │ └── merge\_tables.py # Merging multiple datasets

│ │

│ ├── rag/

│ │ └── rag\_pipeline.py # LangChain-based RAG logic

│ │

│ ├── sql/

│ │ ├── create\_connection.py # Connect to SQL DB

│ │ ├── close\_connection.py # Safely close SQL connection

│ │ ├── create\_table\_in\_sql.py # Table creation logic

│ │ ├── insert\_data.py # Data insertion scripts

│ │ ├── insert\_query.py # Insert queries

│ │ ├── table\_creation.py # Full schema definitions

│ │ └── read\_data\_from\_sql.py # Data fetch operations

│ │

│ └── utils/

│ ├── transform\_comments\_data.py

│ ├── transform\_movies\_data.py

│ ├── transform\_users\_data.py

│ ├── transform\_sessions\_data.py

│ ├── transform\_theaters\_data.py

│ └── transform\_embedded\_movies\_data.py