**🎬 RAG-Powered TV Show Insights with LangChain & Generative AI**

**📌 Project Overview**

This project is a hands-on implementation of **Generative AI** and **LangChain** applied to real-world datasets (Netflix & TMDB).  
It combines **Retrieval-Augmented Generation (RAG)** with multiple **Large Language Models (LLMs)** to let users query TV/movie data in plain English.

Instead of reading raw CSVs or writing SQL, the system allows questions like:

*“Which popular Indian TV shows released after 2020 are still running?”*  
*“List long-running Netflix dramas with high IMDb ratings.”*

The app uses **LangChain to orchestrate retrieval + LLM calls**, and **Generative AI models** (local or cloud) to produce natural-language answers.

**⚙️ Key Components**

**1. Data Processing**

* Datasets from **Kaggle** (Netflix titles & TMDB ~150k shows).
* Cleaned with **Pandas**, labeled (is\_popular, is\_long\_running).
* Saved as **Parquet** for efficient downstream use.

**2. Vector Store with FAISS**

* Text chunks (title, genres, ratings, seasons, etc.) embedded with **HuggingFace embeddings** (all-MiniLM-L6-v2).
* Stored in a **FAISS vector database** for fast similarity search.

**3. Generative AI via LLMs**

Multiple backends supported, making the app **pluggable**:

* **Local (via Ollama)** → LLaMA 3, Mistral
* **Cloud (OpenAI)** → GPT-4o mini, GPT-3.5
* **Cloud (Anthropic)** → Claude 3 Sonnet
* **Hugging Face Hub** → Falcon, other open-source models

Each is a **Generative AI model**, producing answers in natural language.

**4. LangChain Integration**

* **RetrievalQA Chain** from LangChain orchestrates the pipeline:
  1. Embed user query
  2. Retrieve top-k relevant chunks from FAISS
  3. Inject context into the chosen LLM
  4. Generate a fluent, grounded answer

LangChain makes it easy to swap between local and cloud models seamlessly.

**5. Streamlit Web App**

* Sidebar: pick LLM backend (Ollama, GPT, Claude, HF).
* Input box: type your question.
* Output: answer + retrieved sources.

**🛠 Frameworks & Tools Used**

* **LangChain** → RAG pipeline orchestration
* **Generative AI LLMs** → Ollama (local), OpenAI, Anthropic Claude, HuggingFace
* **Pandas** → dataset cleaning & labeling
* **HuggingFace Transformers** → embeddings (all-MiniLM-L6-v2)
* **FAISS** → vector database for similarity search
* **Streamlit** → interactive web UI
* **KaggleHub** → dataset download automation

**📊 Architecture**

**LLM Backend Selection**

👉 (Insert Diagram 1: Ollama vs GPT vs Claude vs HF)

**RAG Pipeline Flow**

👉 (Insert Diagram 2: query → embeddings → FAISS → retriever → LLM → answer)

**🎯 Why This Matters**

* Demonstrates **Generative AI in action**: producing answers from raw structured data.
* Uses **LangChain’s RetrievalQA** to connect search + generation seamlessly.
* Bridges **data engineering, AI model integration, and frontend development**.
* Shows awareness of **trade-offs** between **local/offline vs. cloud/online LLMs**.

⚡ This is a strong **Generative AI + LangChain portfolio project**:

* 📊 Works on real datasets
* 🤖 Combines retrieval + generation
* 💻 Runs locally (Ollama) or in the cloud (GPT/Claude)
* 🎨 Interactive UI with Streamlit

flowchart LR

A[User Query] --> B[Embed Query with HuggingFace Embeddings]

B --> C[Search FAISS Index]

C --> D[Retrieve Top-K Relevant Chunks]

D --> E[Pass Context + Query to LLM]

E --> F[LLM Generates Final Answer]

F --> G[Answer Shown in Streamlit UI]

style A fill:#2c3e50,stroke:#fff,stroke-width:2px,color:#fff

style B fill:#8e44ad,stroke:#fff,stroke-width:1px,color:#fff

style C fill:#16a085,stroke:#fff,stroke-width:1px,color:#fff

style D fill:#27ae60,stroke:#fff,stroke-width:1px,color:#fff

style E fill:#2980b9,stroke:#fff,stroke-width:1px,color:#fff

style F fill:#f39c12,stroke:#fff,stroke-width:1px,color:#000

style G fill:#e74c3c,stroke:#fff,stroke-width:1px,color:#fff