IMDB Chatbot with LangGraph and Streamlit Implementation Report

1. Executive Summary

The IMDB Chatbot is designed to provide natural language interaction with a structured movie dataset. It leverages a hybrid search strategy combining structured SQL and semantic retrieval via FAISS. A ReAct Agent implemented with LangGraph powers the chatbot, which enables dynamic decision-making, reasoning, conversational memory, and stepwise execution. The Streamlit front end provides an intuitive and interactive user experience with transparency regarding the ReAct Agent's thought process (Chain-of-Thought). The LLM of Choice powering the application is 'gemini-2.0-flash'.
This report details the solution architecture, AI methodologies, data preparation techniques, and future improvements, explaining why this implementation is optimal for querying the IMDB dataset.

2. Solution Architecture and Design Choices

2.1 Hybrid Search Approach: SQL + FAISS

- Structured Search via SQLite: This method handles queries related to numeric and categorical data such as year, rating, genre, director, and gross earnings.
- Semantic Search via FAISS: Enables similarity-based retrieval of movies using movie overviews stored as dense vector embeddings.

This approach ensures SQL ensures precise metadata queries, while FAISS allows contextual movie discovery.

2.2 Agent-Based Execution Flow

The chatbot operates as a LangGraph-powered ReAct Agent with the following features:

- Conversational Memory: Tracks user queries for contextual responses.
- Query Expansion: Queries are rephrased or expanded dynamically if no relevant data is found.
- Stepwise Execution: Determines the best retrieval approach per query and executes accordingly.

2.3 Frontend and User Interaction

- Built with Streamlit for an intuitive, web-based chat interface.
- Users input natural language queries, which the Agent interprets and executes.
- Responses include user queries and a chain-of-thought log for transparency regarding the Agent's workflow.

3. Data Preparation and Handling

3.1 Data Cleaning and Transformation

Before constructing the structured and semantic search tools, the raw dataset underwent significant preparation:

- Handling Missing Values: Empty overview fields were replaced with empty strings, missing IMDB_Rating, and other numerical values were assigned defaults so the Agent would understand that the actual value was missing in these fields and respond appropriately to the user.
- Data Type Normalization: Converted fields like Released_Year, Gross Revenue, and Runtime were cleansed and transformed to suitable datatypes for populating into structured formats.

3.2 SQLite Database Construction

- A structured SQLite database (movies.db) was created to store metadata for structured queries.
- Indexes on Series_Title, Genre, and IMDB_Rating improved query efficiency.

3.3 FAISS Index Creation for Semantic Search

- Overview embeddings were generated using '*sentence-transformers/all-MiniLM-L6-v2*'. The SentenceTransformer class handled all of the tokenization, mean pooling, etc., that go with embedding.
- 'IndexFlatL2' was chosen for FAISS due to its effectiveness in short-text retrieval.

4. AI Techniques and Methodologies

4.1 LangGraph React Agent for Query Execution

- React Agents iteratively decide and execute the best tool for a query, reducing unnecessary operations.
- The tools available were the semantic search and structured query tools.

4.2 Conversational Memory for Context Awareness

- Implemented using LangChain's 'Memory' to retain user context.
- Utilized Streamlit session for the application memory handling

4.3 Query Expansion for Improved Retrieval

- If a search returns no results, the query is expanded with synonyms or rewordings (currently a very basic static list; see the future improvements section for ideas to enhance this)

5. Future Improvements

5.1 Self-Reflective Layer for Retry Mechanism

- Adding self-reflection would allow the Agent to retry up to n times if results are weak.
- The value of n will be determined via iterative testing on a robust curated dataset.
- Measures like Perplexity, ROUGE/BLEU score, etc, could be utilized for a programmatic evaluation.

5.2 Guardrails for Safer Interactions

- Implement content filtering and query validation rules to prevent misuse.
- Currently, the app can respond to Gemini's existing knowledge. However, this behavior might need modification in an industry setting while building a domain-specific conversational system.

5.3 Enhanced Query Expansion with LLMs

- Use another LLM call to dynamically rewrite user queries for better semantic search if the result quality is not up to par. Currently, this is just a hard-cast list of expansions that happen in the code.
- If this system is for a specific domain, Other word vector models (specifically trained on domain data) could also be used for semantic query expansion.
- Techniques to dynamically expand the structured query could also be implemented for better results.
- A Self-Reflective Agent could implement result quality checks for iterative query expansion.

6. Alternative Approaches and Justification

6.1 Plan & Execute vs. React Agent

- Plan & Execute pre-plans all steps before execution, while React Agents dynamically adjust per step.
- Plan & Execute usually works better for complex queries. However, they are prone to mid-execution failures, which are typically handled by dynamic replanning (keep the successful steps constant and replan starting from the failure step)

6.2 LLM-Powered Single-Pass Search

- Could use LLMs to generate SQL queries directly but lacks precision compared to structured search tools.
- To my knowledge, the landscape for the Single-Pass choice step for semantic search vs. structured SQL search is not that stable and would require interactive testing.

7. Conclusion

This chatbot implementation leverages a hybrid retrieval strategy, balancing structured and semantic queries with a React Agent framework. With future improvements like self-reflection and guardrails, it can become even more robust, ensuring a reliable and efficient natural language search experience for movie data.