Conversational Movie Discovery System

Technical Implementation Report

---

1. Executive Summary

This project presents a Conversational Movie Discovery System that enables users to search IMDb movies using natural language queries. Built with LangChain, Large Language Models (LLMs), and a vector store, the system interprets vague or partial queries to return relevant movie results with human-like responses.

The solution addresses the limitations of traditional keyword-based search by implementing semantic search powered by HuggingFace embeddings and ChromaDB, combined with LLM-based filter extraction and response generation via ChatGroq's LLaMA 3. The IMDb dataset is efficiently processed using Dask for scalable data handling, with director and writer identifiers mapped to readable names for enhanced user experience.

Key Features:

- Natural language query interpretation
- Intelligent filter extraction (genre, rating, year, actors, language, etc.)
- Context-aware follow-up handling
- Streaming, conversational responses
- Scalable architecture for large datasets
- Improved UI using Streamlit for a richer interaction experience

---

2. System Architecture

2.1 Architecture Overview

The system follows a modular, LLM-powered architecture that integrates LangChain components, vector storage, and conversational memory to ensure semantic accuracy and contextual understanding. The pipeline processes user queries through multiple stages to deliver contextual, natural language responses.

User Query (Natural Language)

↓

Query Processor (LLM-based filter extraction)

↓

Retriever (Vector search with semantic + structured filtering)

↓

Vector Store (ChromaDB)

↓

Retrieved Documents (Structured text + metadata)

↓

LLM (Generates contextual response)

↓

Follow-up Query Handler (Context manager)

↓

Final Response (Natural Language + Recommendations)

2.2 Core Components

Query Processor

- Uses a language model (LLaMA 3 via ChatGroq) to extract structured filters from user queries
- Handles vague, ambiguous, or multi-intent inputs
- Supports follow-up recognition and context continuation
- Converts extracted values into Chroma-compatible filter queries using operators like $gte, $lte, $and, $or, $contains

Large Language Model (LLM)

- Model: LLaMA 3 (8B variant) via ChatGroq
- Configuration:
    - Filter Extraction: temperature = 0.1
    - Response Generation: temperature = 0.7
    - Max tokens: 512
- Primary Functions:
    - Natural language understanding and intent extraction
    - Conversational formatting of movie results
    - Maintains coherence in multi-turn interactions

Vector Store

- Database: ChromaDB for persistent vector storage
- Document Structure: LangChain Document objects containing:
    - page_content: structured text (title, genres, ratings, crew)
    - metadata: structured fields (e.g., tconst, year, rating, genres, director, actors)
- Embeddings generated via HuggingFace model (all-MiniLM-L6-v2)
- Enables filtered and semantically meaningful retrieval

Semantic Search & Retrieval

- Hybrid Approach:
    - Embedding similarity (semantic search)
    - Metadata filtering using structured LLM output
- Uses top-k document retrieval (configurable)
- Filters applied using logical chaining ($and, $or, etc.) within Chroma

Follow-up Query Handler

- Maintains session-specific memory
- Accumulates filter context for follow-up queries
- Ensures relevance across multi-turn interactions

User Interface

- Developed using **Streamlit**
- Features:
    - Sidebar to display live filters and movie count
    - Styled chat bubbles for user/assistant
    - Button-based example queries for guidance
    - Expander sections to inspect filters and retrieved documents per turn
    - Stateless rerun mechanism for interaction refresh
    - Theme-friendly interface with minimal styling

---

3. Data Processing Pipeline

3.1 Raw Data Sources

The system processes IMDb TSV datasets:

- title.akas.tsv – Alternate titles
- title.basics.tsv – Core movie metadata
- title.crew.tsv – Directors and writers
- title.ratings.tsv – Ratings and vote count
- name.basics.tsv – Names of directors/writers

3.2 Data Processing Workflow

Step 1: Data Loading and Filtering

- Technology: Dask with PyArrow backend
- Filter: retain rows with isOriginalTitle == '1' from title.akas.tsv

Step 2: Dataset Merging

- Sequential left joins:
    1. title.basics.tsv ←→ title.ratings.tsv
    2. ←→ title.crew.tsv
    3. ←→ filtered title.akas.tsv

Step 3: Content Filtering

- Retain only feature films (titleType == 'movie')

Step 4: Name Resolution

- Map director/writer IDs to names using name.basics.tsv
- Create readable director_names, writer_names columns

Step 5: Document Creation

- Create LangChain documents:
    - page_content: formatted summary (title, year, rating, genres, etc.)
    - metadata: structured fields used for filtering

---

4. Query Processing and Intelligence

4.1 Natural Language Understanding

Filter Extraction Logic

- LLM (LLaMA 3) extracts filters into a structured JSON format
- Prompt is strict: always return all fields with null/defaults if absent
- Fields include: language, genres, rating_min, rating_max, year_exact, year_after, year_before, year_between, actor, director, keywords, is_greeting, is_continuation

Example Outputs:

- Query: "I want a tamil movie by director Shankar"
  → {"language": "Tamil", "director": "Shankar", ...}
- Query: "Just English" (after context)
  → {"language": "English", "is_continuation": true, ...}

Chroma Filter Conversion

- Converts extracted filters into Chroma-compatible structure
- Supports:
    - $contains for genres, actors, language, etc.
    - $gte, $lte, $eq for numerical filtering
    - $or, $and chaining for complex logic

- Fall-back mechanism for incomplete or malformed outputs

4.2 LLM Integration Strategy

Prompting

- LLM role: friendly, knowledgeable movie assistant
- Style: conversational and human-like
- Injects filters and retrieved document info into context

Response Generation

- Generates recommendations using retrieved documents only
- Refers to movie metadata (title, year, genre, etc.)
- Prevents hallucination by using strict prompt rules
- Supports context accumulation and multi-turn memory

---

5. Technology Stack

Layer	Technologies
Data Processing	Dask, Pandas, PyArrow
Data Storage	ChromaDB, CSVs
Embedding Model	HuggingFace Transformers (all-MiniLM-L6-v2)
LLM	ChatGroq (LLaMA 3 8B 8192)
Vector Search Framework	LangChain + Chroma Vector Store
Prompting & Chaining	LangChain Prompts, RunnableLambda, RunnablePassthrough
Memory Management	Custom context tracking in Streamlit session state
User Interface	Streamlit
Additional Libraries	PyTorch, dotenv, logging, pydantic

---

6. Implementation Challenges and Solutions

6.1 Large Dataset Processing Challenge

Problem:
Processing millions of IMDb records using pandas led to “Out of Array Memory” errors.

Solution:
Used Dask for distributed processing:

- Processes in parallel and in chunks
- Handles joins, filtering, and preprocessing efficiently

- Enables ingestion to ChromaDB without memory issues

6.2 LLM Infrastructure Challenge

Problem:
Running open-source LLMs like LLaMA 3 locally was infeasible due to hardware constraints.

Solution:
Used Groq's hosted LLM inference:

- Low-latency cloud access to LLaMA 3 (8B)
- No local GPU required
- Free-tier supports prototyping and testing
- Integrated via LangChain for both filter and response generation

---

7. Testing and Evaluation

7.1 Test Coverage

The system was tested across a variety of input types:

Query Type	Example
Simple Queries	"Show me action movies"
Filtered Queries	"Find comedies from the 80s with rating above 8"
Follow-up Queries	"Who directed it?"
Ambiguous Inputs	"Find a good movie"
Multi-genre Queries	"Romantic comedies that are also horror films"
Contextual Queries	"Just English" (following a prior Hindi query)

7.2 Performance Assessment

Strengths:

- Accurate LLM-based filter extraction
- Supports nuanced and ambiguous phrasing
- Strong context handling over multi-turn chats
- Efficient hybrid retrieval (semantic + structured)
- Streamlit-based UX offers clarity and interaction depth

Limitations:

- Response accuracy depends on quality of retrieved documents
- No advanced ranking or post-filtering on retrieval results

- Cloud LLM dependency limits full offline deployment
- Genre conflicts (e.g., "sci-fi horror romance") can reduce relevance

---

8. Usage and Deployment

8.1 Installation Requirements

```
# Clone repository
git clone https://github.com/GreenBladePk/Movie-Search-Bot

# Create virtual environment
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt
```

8.2 Configuration

Set up environment variables:

- GROQ_API_KEY – for LLaMA 3 access
- HUGGINGFACE_API_KEY – (optional) for embedding model access

8.3 System Initialization

```
# Preprocess IMDb data and build Chroma vector store
python ingest_data.py

# Launch the Streamlit application
streamlit run app.py
```

8.4 Example Interactions

- "What are some good sci-fi movies from the 1980s?"
- "Show me action movies with rating above 8"
- "Find romantic comedies from the 2000s"
- "Just English" (after previous Hindi query)

---

9. Conclusion

This project successfully delivers a conversational movie search system powered by LLMs and vector search. The migration to Streamlit enhanced interactivity and usability, while LLM-powered filter extraction replaced fragile regex-based logic with a more adaptable and accurate solution.

Real-world limitations like dataset size and model hosting were addressed using Dask and Groq respectively. The system demonstrates strong integration skills across modern AI stacks (LangChain, ChromaDB, Groq, HuggingFace) and delivers a usable, scalable, and intelligent movie assistant interface.

The project showcases end-to-end competence—from data ingestion and vector storage to prompt engineering and real-time interaction design. It effectively solves a real-world challenge of natural language movie recommendation.
