Query Generation for Multi-Modal Film Databases using LLMs

Yeva Galstyan
Global Software Development
Hochschule Fulda
Fulda, Germany
yeva.galstyan@informatik.hs-fulda.de

I. Introduction and Motivation

Research in these areas requires querying large datasets that combine structured metadata, such as release year, genre, and rating, with unstructured content as synopses or reviews. It can be very time-consuming and requires a solid understanding of the schemas to construct queries that retrieve meaningful information. The task becomes more difficult due to incomplete data, particularly in the case of low-budget or independent film productions that lack standardized records. Currently, there is a growing interest in tools that enable natural language querying over complex databases using Large Language Models (LLMs)[1].

This research investigates how LLMs can enable the creation of detailed search queries across various heterogeneous movie datasets. The focus is firstly to assess whether such systems can produce technically correct queries and secondly to estimate whether the system can accurately identify which elements of a question relate to structured data and which refer to plot summaries or taglines, and address the correct parts of the dataset accordingly.

II. Methodology: Data and LLM Architecture

A. Data Consolidation and Heterogeneous Schema Creation

The study utilizes a database compiled from two distinct sources: The MovieLens Dataset (structured metadata) and TMDb (unstructured content). The data processing pipeline consists of several Python scripts.

1) **Initial Merge and Processing:** The raw MovieLens `movies.csv` and `links.csv` files were merged using the movieId. This step also involved extracting the year from the movie title via regular expressions and preserving the external tmdbId for augmentation.
2) **Tag Aggregation:** User-submitted tags from `tags.csv` were filtered only to include tags for the movies present in the final merged dataset. The tags for each movie were then aggregated into a single, delimited `tags` text column.
3) **TMDb Augmentation:** This step performed asynchronous API calls to TMDb, using the extracted `tmdbId` to fetch the essential descriptive texts: `tagline` and `overview`. This step transformed the dataset into a multimodal resource.
4) **Final Upload:** The final consolidated CSVs were imported into the PostgreSQL database, creating the target `movies_full` table.

The final schema provided to the LLM for context is:

Column Name	Data Type	Source
movieId	INT PRIMARY KEY	MovieLens
tmdbId	INT	MovieLens
title	TEXT	MovieLens
genres	TEXT	MovieLens
year	INT	MovieLens
tags	TEXT	MovieLens
tagline	TEXT	TMDb API
overview	TEXT	TMDb API

Database Schema

B. LLM Query Architecture (Gemini Integration)

The Streamlit front-end uses an asynchronous, three-step chain of thought, managed by specific function calls to the Gemini API:

1) **Intent Classification:** The LLM is prompted to classify the user query into a boolean JSON output {`"is_movie_question": true/false`}. This step acts as a guardrail to ensure computational resources (SQL generation) are only expended for relevant tasks.
2) **SQL Generation:** If classified as a movie question, a detailed system prompt, including the full database schema, instructs the LLM to generate a valid PostgreSQL query. The prompt includes strict JSON output constraints and rules for handling text fields.
3) **Execution and Fallback:** The application executes the generated SQL. If the LLM is not able to generate a valid SQL query due to database limitations, the LLM is prompted to create a user-friendly explanation.

III. Experimental Feature: Semantic Vector Search

A. Vector Embedding Methodology

An experimental module was developed to explore vector-based semantic search as an alternative retrieval mechanism, designed to find movies based on meaning or themes rather

than exact keywords, thus bypassing a core limitation of structured SQL.

This was implemented by first generating a 384-dimensional vector embedding for the combined `tagline` and `overview` of each movie using the `all-MiniLM-L6-v2` Sentence Transformer model.

These high-dimensional vectors were stored in a dedicated `VECTOR(384)` column within the database table, utilizing the `pgvector` PostgreSQL extension. Querying involved embedding the user's question and performing a similarity search using the Euclidean distance operator.

```
SELECT title, overview  FROM movies_full
ORDER BY embedding <-> $1 LIMIT 10;
```

B. Status and Evaluation

While this implementation successfully integrated vector technology with the relational database, the feature was ultimately disabled in the main application flow. Preliminary testing showed that, for the structured benchmark queries, the results from the semantic similarity search were often too broad or factually imprecise compared to the exact filtering achieved by the LLM-generated SQL.

IV. Results and Discussion: LLM Performance Analysis

The LLM's performance was evaluated against a set of natural language queries, focusing on its ability to correctly map language to the appropriate database fields and logic.

A. Query Translation

The LLM demonstrated a high command of both structured and unstructured fields:

- **Structured Fields (Year, Genres):** The model correctly used exact numerical matches (`WHERE year = 1995`) and utilized the `LIKE` operator to search within the pipe-delimited genres field.
- **Unstructured Fields (Overview, Tagline):** The LLM successfully parsed user intent that spanned multiple text fields. The LLM generated a highly specific and functional query demonstrating strong linguistic to SQL mapping.
- **Multi-Criteria Complexity:** The model correctly handled combined numerical, text, and boolean logic

B. Constraint Handling

The LLM was directly prompted not to generate queries if the requested information could not be answered by the database schema. The results confirm that the LLM sticks to this explicit constraint, showing a high degree of database introspection.

Query	Output Type	Output
List all movies produced by Steven Spielberg.	Text Response	I am sorry, but the database schema does not contain information about movie producers or directors.
Find romantic movies starring Meryl Streep.	Text Response	I cannot fulfill this request. The database schema does not include information about actors or cast members. I can only search for movies by title, genre, year, tags, tagline, or overview.

Test result: Queries outside the database schema

V. Conclusion

This work explored the use of Large Language Models for query generation across heterogeneous, multi-modal film databases. A unified schema was created by combining structured metadata from MovieLens with unstructured descriptive content from TMDb. This allowed the LLM to reason over both numerical and free-text attributes. Test evaluation demonstrated that the LLM was able to accurately map natural language queries to SQL, correctly distinguish between structured and unstructured fields, and respect the schema constraints when information was unavailable.

An experimental vector-based semantic search module was also integrated, highlighting potential for meaning-oriented retrieval, though its precision was limited compared to the LLM's schema-driven SQL generation.

Future work will focus on combining the strengths of both methods: using LLMs to generate accurate SQL for structured fields, while relying on vector search to handle meaning-based queries over text.

References

[1] X. Liu et al., "A Survey of NL2SQL with Large Language Models: Where are we, and where are we going?," arXiv.org, 2024. https://arxiv.org/abs/2408.05109.