Movie Recommendation System - Solution Overview

Data Preprocessing

1. Bollywood Dataset:I started with the Bollywood movie dataset from Kaggle and performed data preprocessing to clean the data by removing missing values.
2. Genre Extraction: I added an extra column for movie genres to help with filtering. I used IMDb IDs from the dataset to gather additional movie metadata via web scraping, such as actors, directors, and genres, which were then added to the dataset.

Neo4j Database

1. Graph Creation: The cleaned dataset was fed into the Neo4j graph database. I created nodes for each movie and added relationships between these nodes, such as ACTED_IN, DIRECTED_BY, and BELONGS_TO_GENRE. This step allowed you to represent movies, actors, directors, and genres as entities in a graph structure.
2. Querying the Graph: Using Cypher queries, you built logic to find movie similarities based on shared attributes like actors, directors, or genres. For example:

```
MATCH (m:Movie {name: "3 Idiots"})-[:ACTED_IN|DIRECTED_BY|BELONGS_TO_GENRE]-(similarMovies)
RETURN similarMovies.name
LIMIT 3
```

Recommendation Engine

1. GraphRAG-based Logic: I utilized the GraphRAG architecture to generate movie recommendations by leveraging the graph structure. By traversing the graph and analyzing movie connections (e.g., similar actors, directors, and genres), you were able to recommend movies similar to the one a user watches.
2. Collaborative Filtering: I optionally added collaborative filtering, where the behavior of other users who watched similar movies was factored in to recommend movies.
3. Content-based Filtering: The recommendation system was enhanced with content-based filtering, where the model used metadata like genres, actors, and directors for filtering recommendations.

Machine Learning Model (KNN)

1. K-Nearest Neighbors: I implemented a KNN model to further train and predict movie

recommendations. This helped in generating recommendations based on the features (genres, actors, etc.) of the movies.

UI Development (Streamlit)

1. Streamlit Frontend: I developed a simple and interactive user interface using Streamlit. The UI allows users to input a movie they have watched, and the system returns the top 3 recommended similar movies based on the underlying logic.
2. Local Deployment: The application was deployed locally, making it accessible for users to interact with and receive movie recommendations.

Steps Taken

1. Data Collection:
  - Downloaded the Bollywood dataset from Kaggle.
  - Scraped additional metadata from IMDb using IMDb IDs.

2. Database Design:
  - Fed the cleaned dataset into a Neo4j database.
  - Created nodes for Movies, Actors, Directors, and Genres, with appropriate relationships.

3. Recommendation System:
  - Built a GraphRAG-based recommendation engine using graph traversal to recommend similar movies.
  - Incorporated collaborative filtering and content-based filtering techniques to enhance recommendations.

4. Model Training:
  - Trained a KNN model to provide more accurate movie recommendations based on user preferences and movie metadata.

5. Frontend Development:
  - Developed a user interface with Streamlit to display movie recommendations.

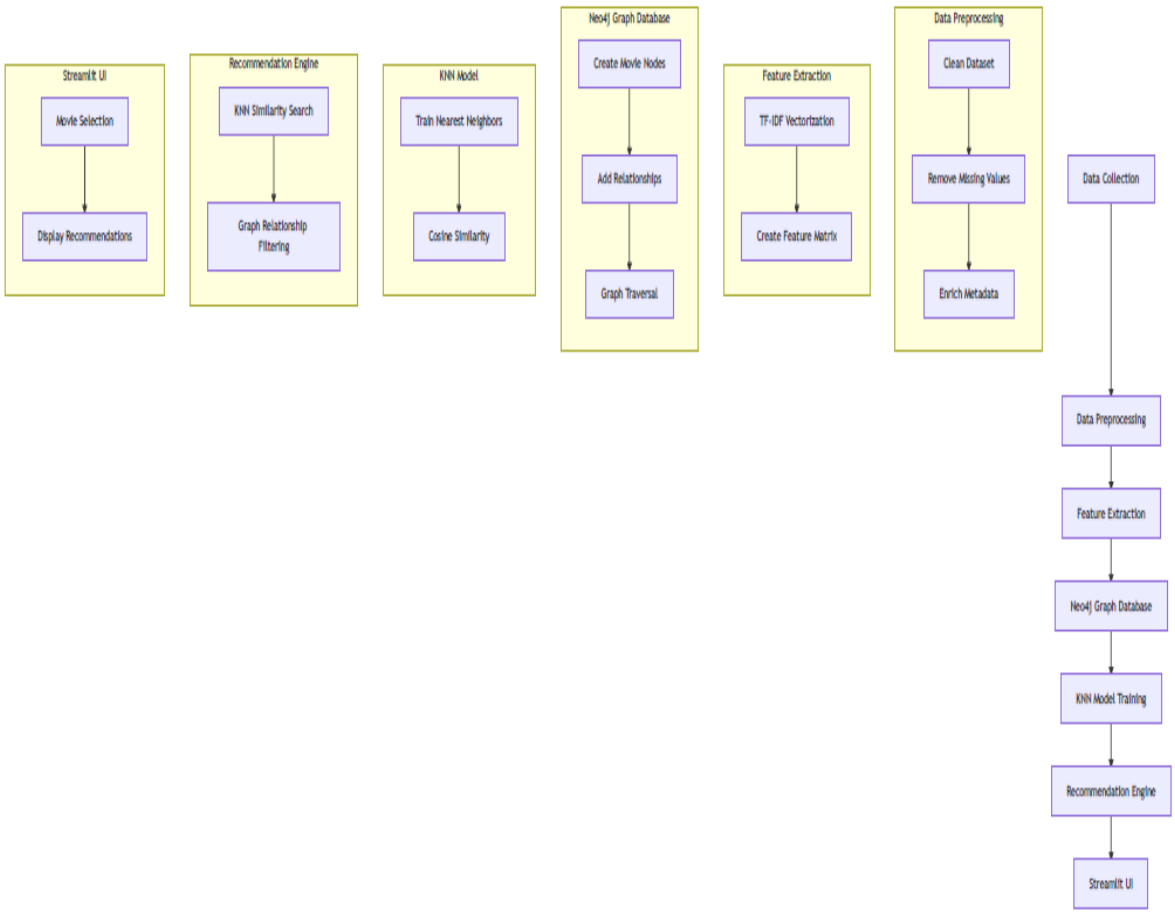
6. Deployment:
  - Deployed the system locally to allow users to interact with the movie recommendation engine.

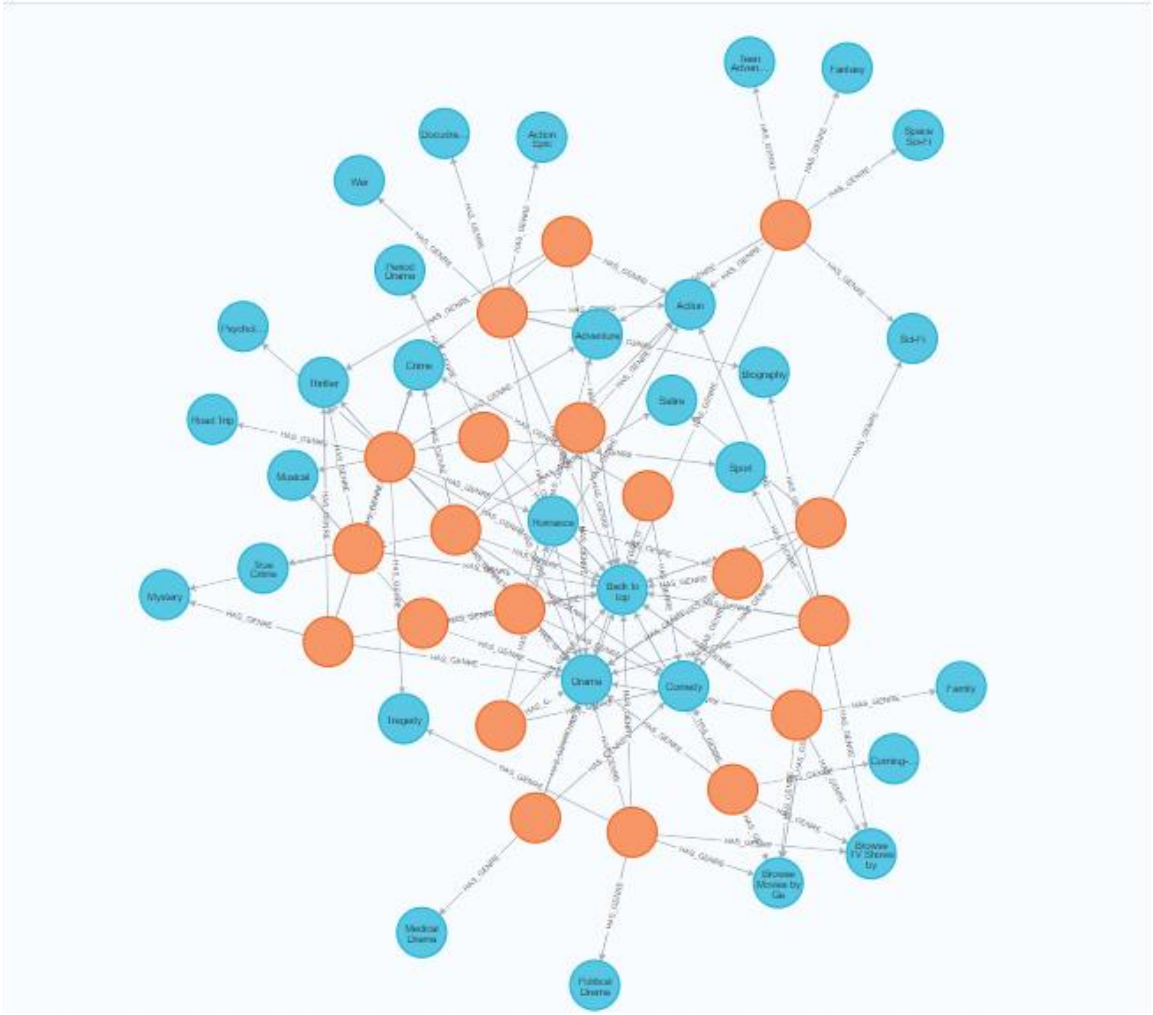
Challenges Faced in the Project

1. **Limited IMDb API Calls**:
    - **Problem**: IMDb API restricts to 1,000 calls per day; dataset had 9,998 movies.
    - **Impact**: Limited ability to fetch additional metadata (genres, actors, directors) for all movies.
    - **Solution**:
        - Use batch WEB scraping.
        - Cache fetched data locally to reduce repeated calls.
2. **Neo4j Docker Connectivity**:
    - **Problem**: Issues connecting Neo4j database in Docker to the local machine.
    - **Impact**: Hindered graph database setup and querying.
    - **Solution**:
        - Correctly configure Docker for network access.
        - Use appropriate ports and --network host for seamless communication.
3. **Cloud Service for Hosting**:
    - **Problem**: Need for cloud services like Google Cloud or AWS, but premium services required credit card details.
    - **Impact**: Cloud hosting was not feasible due to high costs.
    - **Solution**:
        - Deploy using streamlit (locally)
4. **Parallel Processing for Data Enrichment**:
    - **Problem**: Difficulty processing and enriching large datasets due to limited API calls.
    - **Impact**: Time-consuming metadata extraction.
    - **Solution**:
        - Implement parallel processing (multithreading or multiprocessing).

        - Use tools like Scrapy or BeautifulSoup for more efficient scraping.

5. **Deployment and Public Access**:
    - **Problem**: High cost of cloud deployment and need for credit card details.
    - **Impact**: Limited deployment options for public access.
    - **Solution**:
        - Deploy on streamlit.


Streamlit UI
Movie Selection
Display Recommendations
Recommendation Engine
KNN Similarity Search
Graph Relationship Filtering
KNN Model
Train Nearest Neighbors
Cosine Similarity
Neo4j Graph Database
Create Movie Nodes
Add Relationships
Graph Traversal
Feature Extraction
TF-IDF Vectorization
Create Feature Matrix
Data Preprocessing
Clean Dataset
Remove Missing Values
Enrich Metadata
Data Collection
Data Preprocessing
Feature Extraction
Neo4j Graph Database
KNN Model Training
Recommendation Engine
Streamlit UI


Docume...
Action Epic
War
Period Drama
Psychol...
Thriller
Crime
Road Trip
Musical
Mystery
True Crime
Tragedy
Adventure
Action
Biography
Sitcom
Sport
Romance
Back to top!
Drama
Comedy
Family
Browse TV Shows by
Browse Movies by Ge
Medical Drama
Political Drama