Movie Recommendation System

A report submitted in partial fulfillment of the requirement for the award of the degree

Of

Bachelor of Technology in Computer Science & Engineering

in

Faculty Of Computer Technology

Submitted by

Abdus Sahir Choudhury
ADTU/L/2022-26/BCS/143

Pranab Jyoti Boruah
ADTU/L/2022-26/BCS/148

Snigdha Neogi
ADTU/2022-26/BCS/105

Assam down town University
Guwahati-26, Assam
Session: January-June, 2024

CONTENTS

DECLARATION

We, ***Abdus Sahir Choudhury*** bearing Roll No. *ADTU/L/2022-26/BCS/143*, ***Pranab Jyoti Boruah*** bearing Roll No. *ADTU/L/2022-26/BCS/148* and ***Snigdha Neog*i** bearing Roll No. *ADTU/2022-26/BCS/105* hereby declare that the thesis entitled ***Movie Recommendation System*** is an original work carried out in the Department of Computer Technology, Assam down town University, Guwahati with exception of guidance and suggestions received from my supervisor, ***Dr Prasenjit Das***, Assistant Professor, Department of Computer Technology, Assam down town University, Guwahati. The data and the findings discussed in the thesis are the outcome of my research work. This report is being submitted to Assam down town University for the degree of ***Bachelor of Technology .***

Abdus Sahir Choudhury

Enrolment: ADTU/L/2022-26/BCS/143

Semester: 6th

Programme: B. Tech in CSE

Faculty of Computer Technology

Assam down town University, Guwahati

Pranab Jyoti Boruah

Enrolment: ADTU/L/2022-26/BCS/148

Semester: 6th

Programme: B. Tech in CSE

Faculty of Computer Technology

Assam down town University, Guwahati

Snigdha Neogi

Enrolment: ADTU/2022-26/BCS/105

Semester: 6th

Programme: B. Tech in CSE

Faculty of Computer Technology

Assam down town University, Guwahat

ACKNOWLEDGMENT

We would like to extend our heartfelt appreciation to everyone who contributed to the successful completion of this project. Our sincere thanks go to our project team members for their dedication and collaboration throughout the project. Each member played a significant role in shaping the outcome. Special thanks to our coordinator, *Dr. Prasenjit Das*, for his guidance and valuable feedback, which enriched our work. Lastly, we want to thank our friends for their patience and encouragement during this project. Their believe helped us to stay motivated and to persevere through difficult times.

ABSTRACT

The project delves into the realm of ***Machine Learning*** application of ***Movie Recommendation System*** developed using Python and deployed via ***Streamlit***. The system utilizes a user-selected movie as input and recommends similar movies based on their metadata, including genres, cast, crew, overview, and keywords. Leveraging ***Natural Language Processing (NLP)*** techniques, particularly ***TF-IDF vectorization*** and ***cosine similarity***, the system quantifies textual data and identifies relationships among movies.

The front-end interface has been designed using ***Streamlit***, a rapid application development framework for data apps, ensuring the solution is both interactive and user-friendly. Movie posters are dynamically fetched using the ***OMDB API***, enhancing the visual appeal and usability of the recommendation system. This project is a testament to the integration of data science, web development, and user experience design to solve real-world problems in content discovery.

List of Figures

1. INTRODUCTION

1.1 Overview of the project

The Content-Based Movie Recommendation System developed in this project is specifically designed to assist users in discovering movies that closely align with their interests. Rather than relying on external ratings or user behaviour patterns, this system analyzes descriptive metadata such as plot summaries, genres, cast, crew, and keywords—associated with each movie.

The approach is grounded in natural language processing (NLP) techniques, particularly through TF-IDF (Term Frequency–Inverse Document Frequency) vectorization, which quantifies textual data. By computing cosine similarity scores between movies, the system can determine how similar one film is to another, enabling personalized recommendations based on a single user-selected movie.

Unlike collaborative filtering systems that require user ratings or engagement histories, this content-based model is self-sufficient, functioning effectively without any prior user data. The system is deployed using Streamlit, a lightweight Python-based web application framework, ensuring easy accessibility and an intuitive user experience.

1.2 Motivation

With the explosive growth of digital streaming platforms such as Netflix, Amazon Prime, Disney+, and others, users are frequently confronted with the paradox of choice—an overwhelming number of movies and limited tools to make informed decisions. While many platforms employ popularity-based or collaborative filtering algorithms, these models often fall short in addressing the needs of first-time users or items that have limited interaction history.

The core motivation behind this project is to bridge the gap between user expectations and traditional recommendation approaches by leveraging metadata alone. Content-based filtering provides the advantage of being immune to the cold-start problem, meaning it can function seamlessly even in the absence of user ratings or historical interactions. By focusing

on the inherent attributes of movies, the system offers a tailored and consistent recommendation experience to all users, regardless of their interaction history.

Furthermore, the use of open-source tools and publicly available movie datasets emphasizes the project's focus on accessibility, reproducibility, and practical implementation. The project also showcases how machine learning and NLP can be harnessed to solve real-world problems through the integration of intelligent algorithms and modern web technologies.

1.3 Scope and Objectives

The scope of this project encompasses the full development lifecycle of a content-based movie recommendation engine, from data ingestion and preprocessing to interface design and deployment. The system is engineered to be efficient, scalable, and user-friendly, capable of handling large movie datasets and delivering high-quality recommendations.

The key objectives of the project are as follows:

- To collect, clean, and preprocess metadata from multiple movie-related datasets (e.g., movies metadata, cast and crew, keywords, and links).
- To apply TF-IDF vectorization to convert textual features such as overviews and keywords into numerical vectors that capture the importance of terms within the corpus.
- To compute pairwise cosine similarity scores between movies based on their vectorized representations.
- To generate and display a ranked list of top-N similar movies when a user selects a particular title.
- To utilize the OMDB API for retrieving visually appealing and accurate movie posters based on IMDb identifiers.
- To integrate all components into an interactive and visually intuitive web interface using the Streamlit framework.

- To ensure the system is modular and deployable, suitable for future extensions such as hybrid filtering or personalized user accounts.

1.4 Existing Systems

Several movie recommendation systems are already in use, especially on platforms like IMDb, Netflix, and YouTube, which employ sophisticated algorithms to suggest content. The majority of these systems rely on two dominant approaches: popularity-based filtering and collaborative filtering.

- Popularity-based filtering recommends content that is trending or has high ratings but lacks personalization.
- Collaborative filtering, while more personalized, depends heavily on user interactions such as ratings, reviews, and viewing histories.

Both methods face significant challenges. Collaborative filtering is particularly affected by the cold start problem, where new users or new items (e.g., newly released movies) do not have enough data to be included in the recommendation logic. Additionally, such models require extensive user data collection, raising privacy and scalability concerns.

2. PROJECT ANALYSIS

2.1 Project Requirement Analysis

A detailed understanding of the system requirements is essential for successful implementation. This section outlines the core technologies, tools, and data sources utilized in the development of the project:

Programming Language:

- **Python —** Selected for its simplicity, readability, and extensive ecosystem of data science and machine learning libraries.

Libraries and Tools:

- **Pandas** – Used for efficient data wrangling, cleaning, and manipulation.
- **Scikit-learn –** Applied for feature extraction (TF-IDF) and similarity computation (cosine similarity).
- **Streamlit –** Employed to design and deploy an interactive and responsive front-end web interface.
- **Pickle –** Utilized to serialize and save processed data structures (e.g., similarity matrices) for efficient reuse.
- **Requests –** Used for seamless communication with external APIs.

External API:

- **OMDB API (Open Movie Database) –** Integrated for fetching high-quality movie poster images and supplementary metadata.

Datasets Used:

- **credits.csv –** Includes detailed information about cast and crew.
- **keywords.csv –** Contains core movie attributes such as title, overview, genres, etc.

2.2 Gantt Chart (Development Timeline)

The development process was planned over distinct phases using a Gantt chart to ensure timely execution. Each phase was broken down into specific tasks with defined timelines:

Task	Week 1	Week 2	Week 3	Week 4
Dataset Collection	✅			
Metadata Cleaning	✅	✅		
Feature Engineering		✅		
Model Development			✅	
UI & Deployment				✅

2.3 Advantages and Disadvantages

Advantages:

- No Dependency on User Profiles: The system does not require prior user data to generate recommendations, making it suitable for first-time users.
- Instant Recommendations: It provides movie suggestions based solely on the selected movie, ensuring quick and direct outputs.
- Visual Enrichment: Integration with the OMDB API enhances the user interface with dynamic poster visuals.
- Lightweight Architecture: The application is fast, efficient, and deployable with minimal hardware resources.

Disadvantages:

- Novel Discovery: Recommendations are constrained by known metadata and may miss less explicitly similar but contextually relevant movies.
- Metadata Cold Start: New movies with incomplete or limited metadata might not be recommended accurately.
- Static Recommendations: The system cannot adapt over time to reflect evolving user preferences or viewing history.

2.4 Project Life Cycle

The project followed the Iterative Development Model, emphasizing continuous refinement and improvement based on testing and feedback. Each iteration allowed incremental development and validation of system components.

Development Stages:

1. Requirement Analysis – Defining system goals, selecting technologies, and identifying data needs.
2. Data Cleaning & Preprocessing – Merging datasets, handling missing values, and preparing textual data.
3. Model Construction and Evaluation – Implementing TF-IDF and cosine similarity, evaluating recommendation accuracy.
4. Front-End Integration – Designing a user-friendly interface using Streamlit.
5. Testing and Deployment – Functional testing, debugging, and deploying the application locally or on cloud platforms.

2.5 Project Feasibility

Technical Feasibility:

The project is entirely built using open-source tools and frameworks such as Python, Streamlit, and Scikit-learn. These tools are well-documented, widely used, and compatible across platforms, making the system technically sustainable and accessible.

Economic Feasibility:

All datasets and tools used are freely available under open licenses. The system can be developed and deployed without incurring any licensing or infrastructural costs, making it highly economical for students and independent developers.

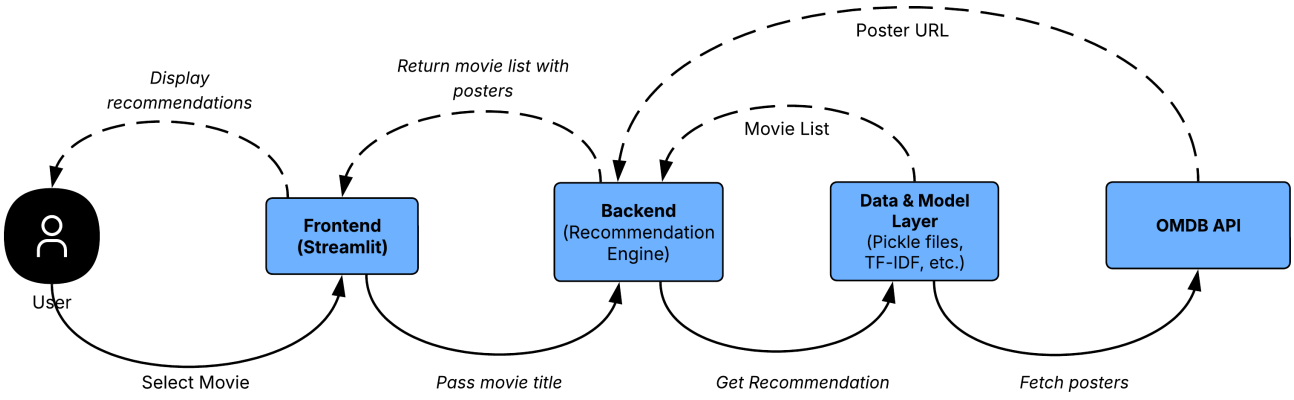
Operational Feasibility:

The final system is lightweight and can be deployed easily using cloud platforms like Streamlit Community Cloud or hosted locally. Its user interface is intuitive and does not require technical expertise to operate, ensuring high usability and operational success.

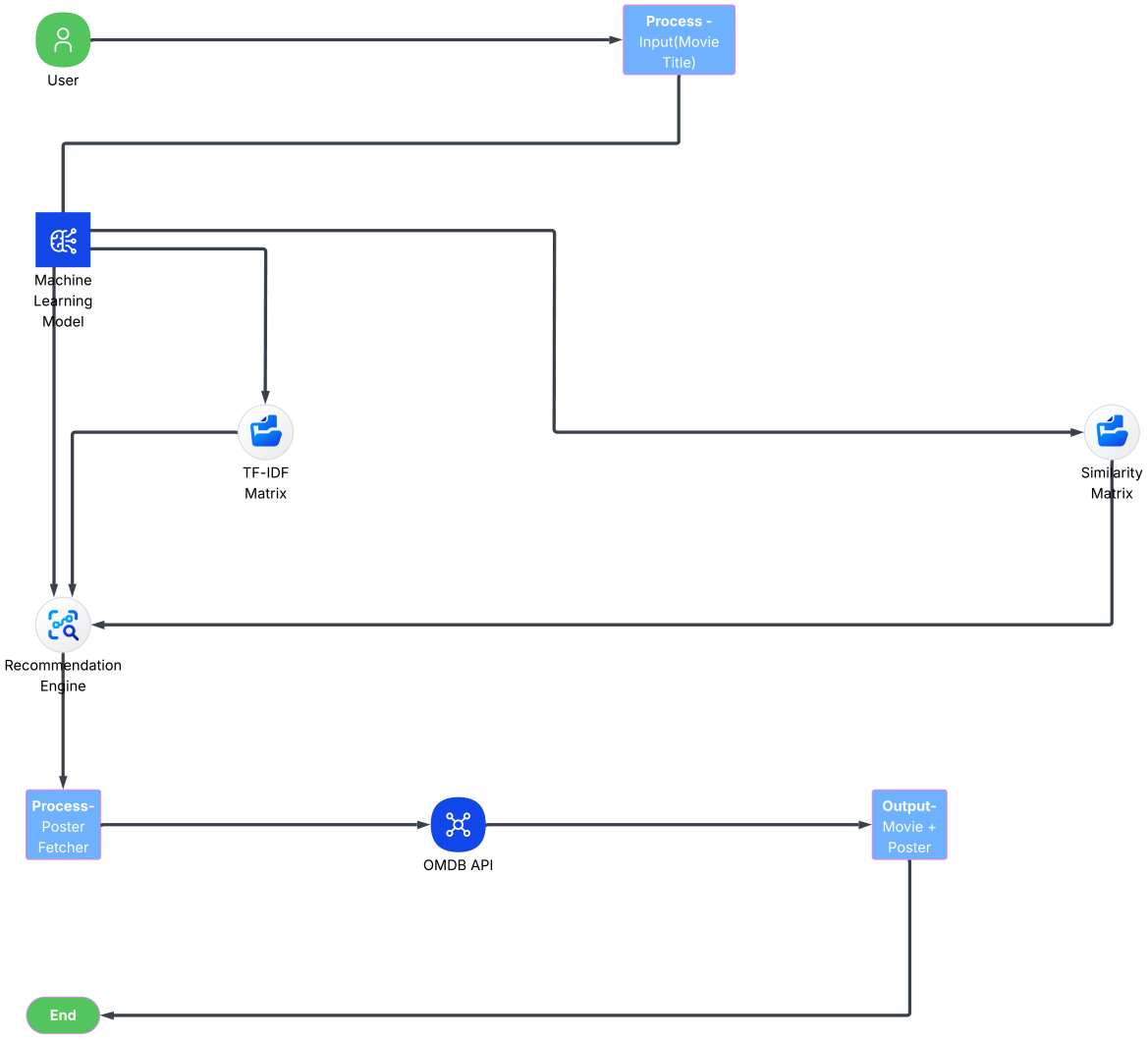
3. PROJECT DESIGN

3.1 Project Flow Chart

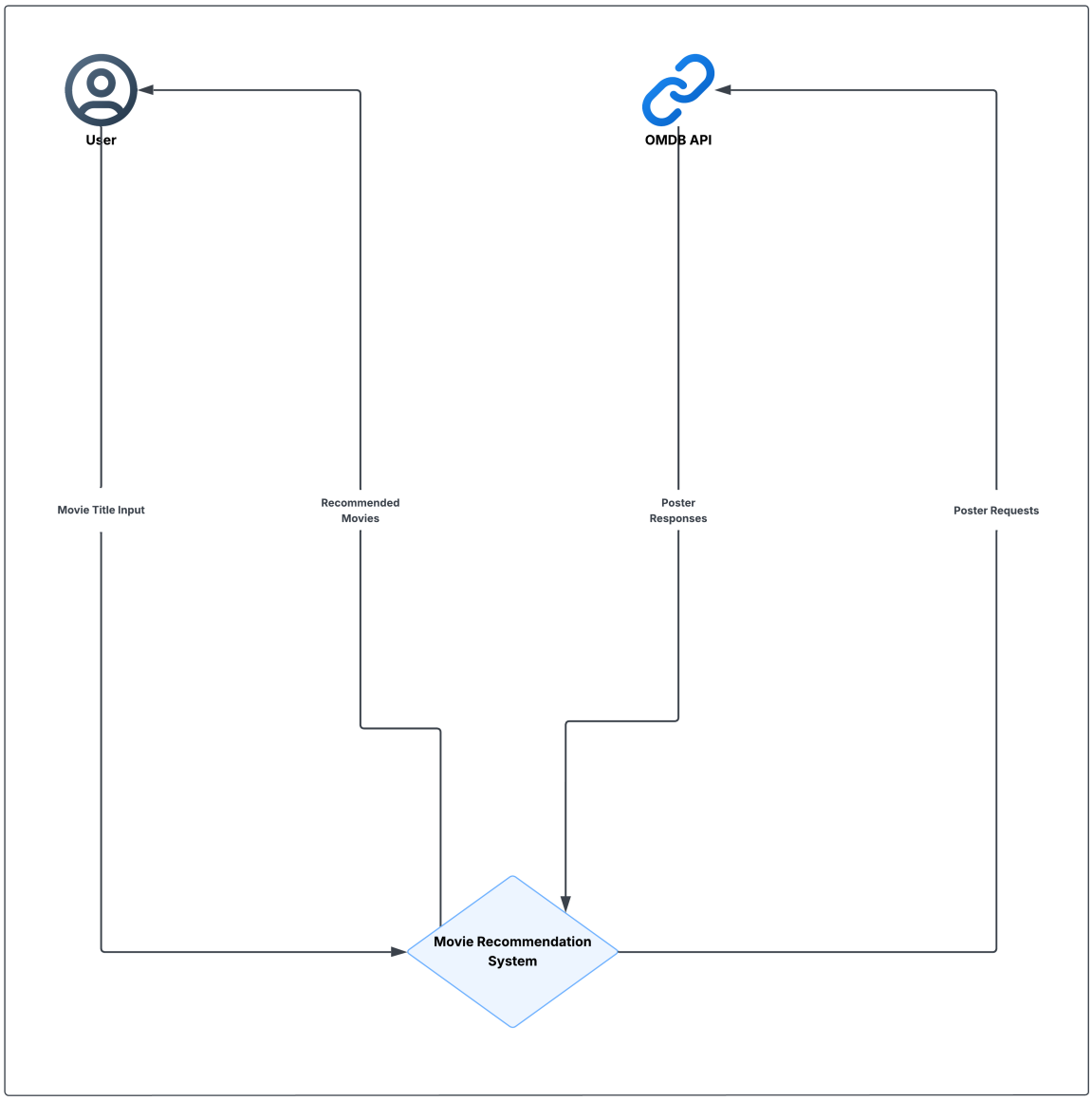
3.2 System Architecture



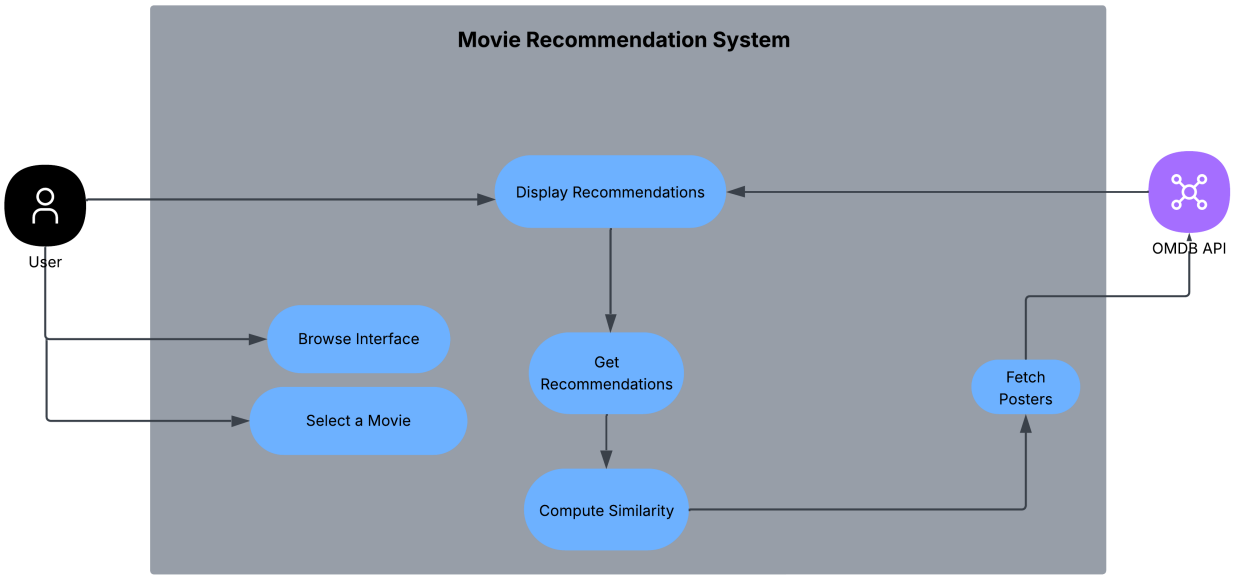
3.3 Data Flow Diagram



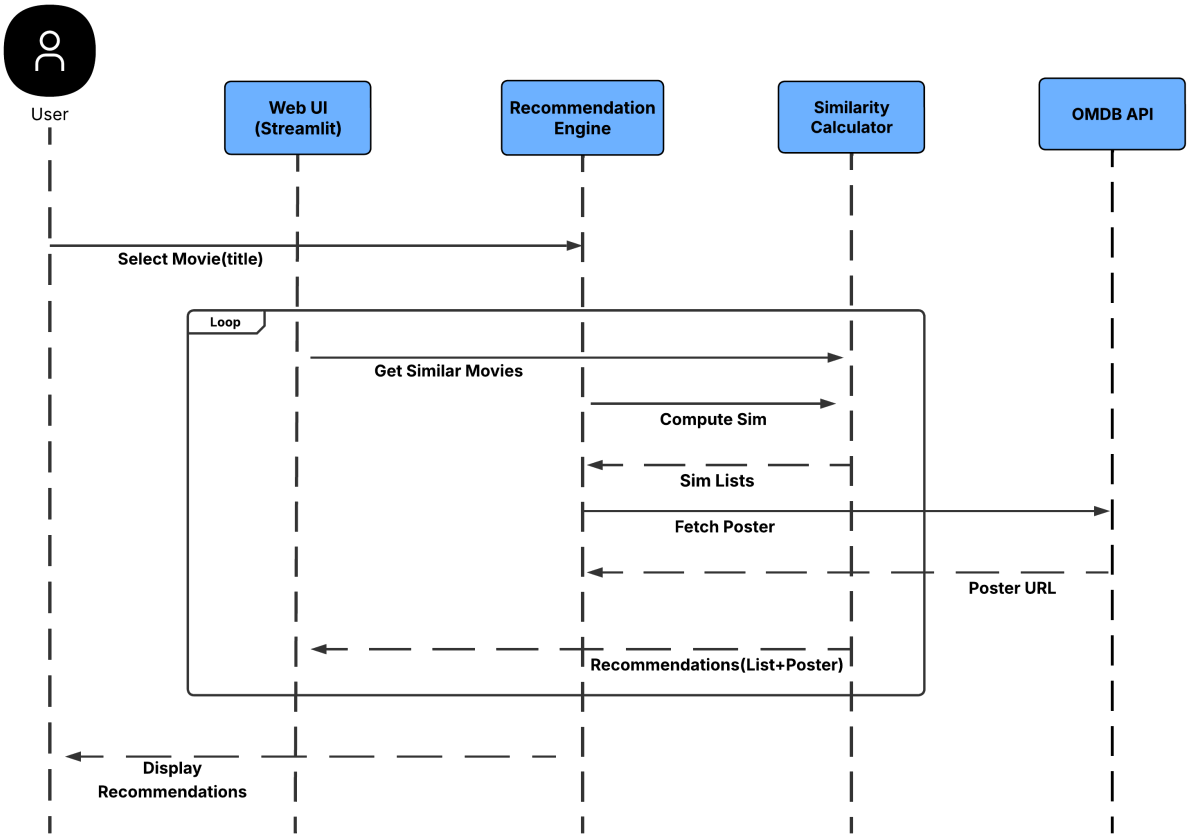
3.3.1 Context Diagram



3.4 Use case Diagram



3.5 Sequence Diagram



4. PROJECT IMPLEMENTATION

4.1 Data Collection and Loading

The project begins with collecting the dataset, typically from Kaggle or IMDb. The dataset includes metadata for thousands of movies, such as:

- imdb_id
- Title
- Overview
- Genres
- Cast
- Crew

The data is loaded into a Pandas DataFrame for manipulation.

```python
import numpy as np
import pandas as pd

import os
for dirname, _, filenames in os.walk('/Users/sahirchoudhury/Movie Recommandation System'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

Pytho

```python
credits = pd.read_csv('/Users/sahirchoudhury/Movie Recommandation System/credits.csv',low_memory=False)
keywords = pd.read_csv('/Users/sahirchoudhury/Movie Recommandation System/movies_metadata.csv',low_memory=False)
```

Python

4.2 Data Merging

These two datasets are merged on the movie title to form a consolidated metadata file.

```python
keywords['id'] = keywords['id'].astype(str)
credits['id'] = credits['id'].astype(str)
```

Python

```python
data = keywords.merge(credits,on='id')
```

Python

4.3 Data Preprocessing

4.3.1 Feature Selection and Cleaning

```
[10]
data = data[['imdb_id','title','overview','genres','cast','crew']]
```
Python

```
[11]
data.head()
```
Python

	imdb_id	title	overview	genres	cast	crew
0	tt0114709	Toy Story	Led by Woody, Andy's toys live happily in his ...	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...	[{'cast_id': 14, 'character': 'Woody (voice)',...	[{'credit_id': '52fe4284c3a36847f8024f49', 'de...
1	tt0113497	Jumanji	When siblings Judy and Peter discover an encha...	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...	[{'cast_id': 1, 'character': 'Alan Parrish', '...	[{'credit_id': '52fe44bfc3a36847f80a7cd1', 'de...
2	tt0113228	Grumpier Old Men	A family wedding reignites the ancient feud be...	[{'id': 10749, 'name': 'Romance'}, {'id': 35, ...	[{'cast_id': 2, 'character': 'Max Goldman', 'c...	[{'credit_id': '52fe466a9251416c75077a89', 'de...
3	tt0114885	Waiting to Exhale	Cheated on, mistreated and stepped on, the wom...	[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...	[{'cast_id': 1, 'character': "Savannah 'Vannah...	[{'credit_id': '52fe44779251416c91011acb', 'de...
4	tt0113041	Father of the Bride Part II	Just when George Banks has recovered from his ...	[{'id': 35, 'name': 'Comedy'}]	[{'cast_id': 1, 'character': 'George Banks', '...	[{'credit_id': '52fe44959251416c75039ed7', 'de...

```
[12]
data.isnull().sum()
```
Chat (⌘+I) / Share (⌘+L)

Python

```
imdb_id      17
title         3
overview    954
genres        0
cast          0
crew          0
dtype: int64
```

```
[13]
data.dropna(inplace=True)
```
Python

4.3.2 Text Normalization and Transformation

```
[18]
data['genres'] = data['genres'].apply(convert)
```
Python

```
[19]
"name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {"id": 878, "name": "Science Fiction"}]')
```
Python

```
[{'id': 28, 'name': 'Action'},
 {'id': 12, 'name': 'Adventure'},
 {'id': 14, 'name': 'Fantasy'},
 {'id': 878, 'name': 'Science Fiction'}]
```

```
[20]
def convert3(text):
    L = []
    counter = 0
    for i in ast.literal_eval(text):
        if counter < 3:
            L.append(i['name'])
        counter+=1
    return L
```
Python

```
[21]
data['cast'] = data['cast'].apply(convert)
data.head()
```
Python

	imdb_id	title	overview	genres	cast	crew
0	tt0114709	Toy Story	Led by Woody, Andy's toys live happily in his ...	[Animation, Comedy, Family]	[Tom Hanks, Tim Allen, Don Rickles, Jim Varney...	[{'credit_id': '52fe4284c3a36847f8024f49', 'de...
1	tt0113497	Jumanji	When siblings Judy and Peter	[Adventure,	[Robin Williams, Jonathan	[{'credit_id':

4.4 Feature Engineering

4.4.1 Tag Generation and Stemming

```
[26]
def collapse(L):
    L1 = []
    for i in L:
        L1.append(i.replace(" ",""))
    return L1
```
Python

```
[27]
data['cast'] = data['cast'].apply(collapse)
data['crew'] = data['crew'].apply(collapse)
data['genres'] = data['genres'].apply(collapse)
```
Python

```
[28]
data.head()
```
Python

	imdb_id	title	overview	genres	cast	crew
0	tt0114709	Toy Story	Led by Woody, Andy's toys live happily in his ...	[Animation, Comedy, Family]	[TomHanks, TimAllen, DonRickles]	[JohnLasseter]
1	tt0113497	Jumanji	When siblings Judy and Peter discover an encha...	[Adventure, Fantasy, Family]	[RobinWilliams, JonathanHyde, KirstenDunst]	[JoeJohnston]
2	tt0113228	Grumpier Old Men	A family wedding reignites the ancient feud be...	[Romance, Comedy]	[WalterMatthau, JackLemmon, Ann-Margret]	[HowardDeutch]

```
[29]
data['overview'] = data['overview'].apply(lambda x:x.split())
```
Python

```
[30]
data['tags'] = data['overview'] + data['genres'] + data['cast'] + data['crew']
```
Python

```
[31]
new = data.drop(columns=['overview','genres','cast','crew'])
```
Python

```
[32]
new.head()
```
Python

	imdb_id	title	tags
0	tt0114709	Toy Story	[Led, by, Woody,, Andy's, toys, live, happily,...
1	tt0113497	Jumanji	[When, siblings, Judy, and, Peter, discover, a...
2	tt0113228	Grumpier Old Men	[A, family, wedding, reignites, the, ancient, ...
3	tt0114885	Waiting to Exhale	[Cheated, on,, mistreated, and, stepped, on,, ...
4	tt0113041	Father of the Bride Part II	[Just, when, George, Banks, has, recovered, fr...

```
[33]
new['tags'] = new['tags'].apply(lambda x: " ".join(x))
new.head()
```
Python

	imdb_id	title	tags
0	tt0114709	Toy Story	Led by Woody, Andy's toys live happily in his ...
1	tt0113497	Jumanji	When siblings Judy and Peter discover an encha...
2	tt0113228	Grumpier Old Men	A family wedding reignites the ancient feud be...

4.4.2 Feature Vectorization

```
[81]
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import TruncatedSVD
import pandas as pd

# Step 1: Vectorize text data
cv = CountVectorizer(max_features=22283, stop_words='english')
vectors = cv.fit_transform(new['tags'])  # assuming 'new["tags"]' contains text

print("Original shape:", vectors.shape)  # should be (44566, 30000)

# Step 2: Reduce dimensions with TruncatedSVD
svd = TruncatedSVD(n_components=2283, random_state=69)
vector = svd.fit_transform(vectors)

print("Reduced shape:", vector.shape)  # now (44566, 3000)
```

Python

```
Original shape: (44566, 22283)
Reduced shape: (44566, 2283)
```

```
[82]
print(svd.explained_variance_ratio_.sum())
# Try to aim for 80–90% of total variance
```

Python

```
0.6876352810986338
```

```
[83]
print(f"Explained variance: {svd.explained_variance_ratio_.sum():.2f}")
```

Python

```
Explained variance: 0.69
```

4.5 Similarity Matrix Creation

```
[85]
vector = cv.fit_transform(new['tags']).toarray()
```

Python

```
[86]
vector.shape
```

Python

```
(44566, 22283)
```

```
[87]
from sklearn.metrics.pairwise import cosine_similarity
```

Python

```
[88]
similarity = cosine_similarity(vector)
```

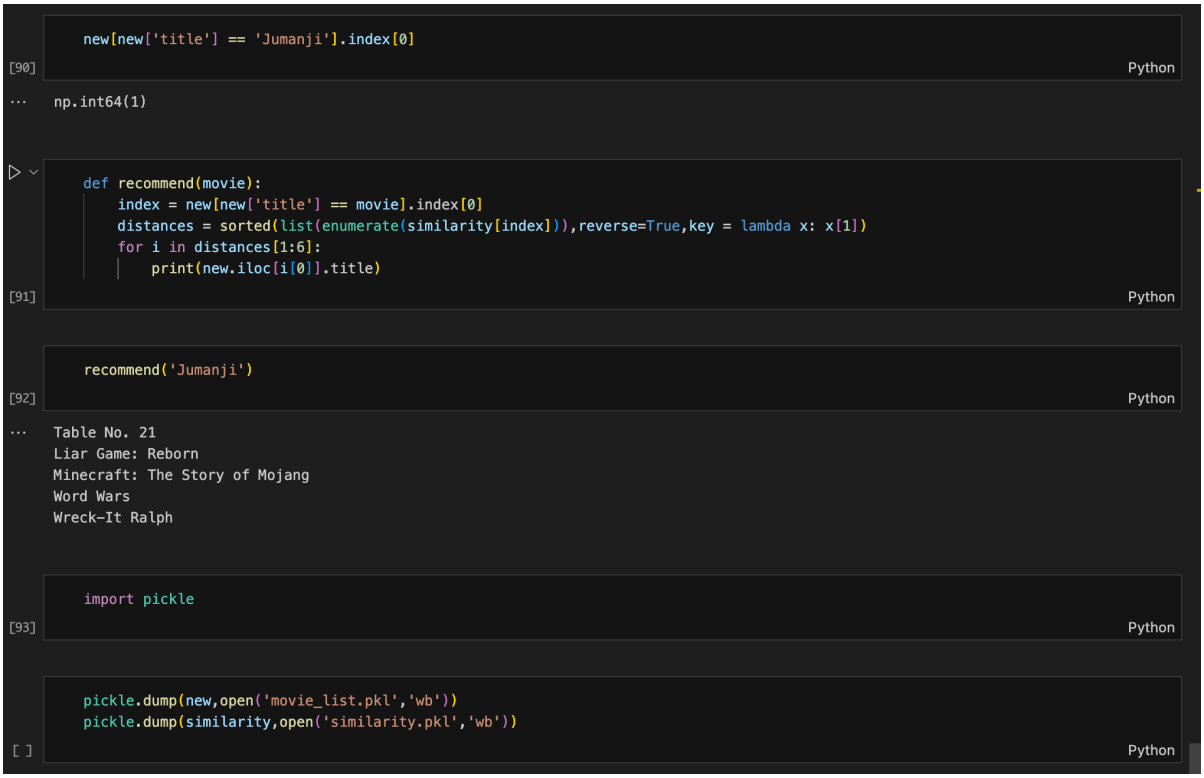
Python

```
[89]
similarity
```

Python

```
array([[1.        , 0.03888079, 0.04250511, ..., 0.        , 0.01418761,
        0.        ],
       [0.03888079, 1.        , 0.06693161, ..., 0.        , 0.02978777,
        0.03464795],
       [0.04250511, 0.06693161, 1.        , ..., 0.        , 0.01628224,
        0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 1.        , 0.        ,
        0.        ],
       [0.01418761, 0.02978777, 0.01628224, ..., 0.        , 1.        ,
        0.        ],
       [0.        , 0.03464795, 0.        , ..., 0.        , 0.        ,
        1.        ]], shape=(44566, 44566))
```

4.6 Recommendation Model - Pickle



```
[90]
new[new['title'] == 'Jumanji'].index[0]
Python

np.int64(1)

[91]
def recommend(movie):
    index = new[new['title'] == movie].index[0]
    distances = sorted(list(enumerate(similarity[index])),reverse=True,key = lambda x: x[1])
    for i in distances[1:6]:
        print(new.iloc[i[0]].title)
Python

[92]
recommend('Jumanji')
Python

Table No. 21
Liar Game: Reborn
Minecraft: The Story of Mojang
Word Wars
Wreck-It Ralph

[93]
import pickle
Python

[ ]
pickle.dump(new,open('movie_list.pkl','wb'))
pickle.dump(similarity,open('similarity.pkl','wb'))
Python
```

4.7 Deployment

The application is deployed locally or on the cloud (e.g., Streamlit Cloud, Heroku). Deployment includes:

- Saving necessary files (movies.pkl, similarity.pkl)
- Uploading the app script (app.py)
- Ensuring API keys are securely stored

4.8 User Interface (UI)

Fig 4.1 - Homepage

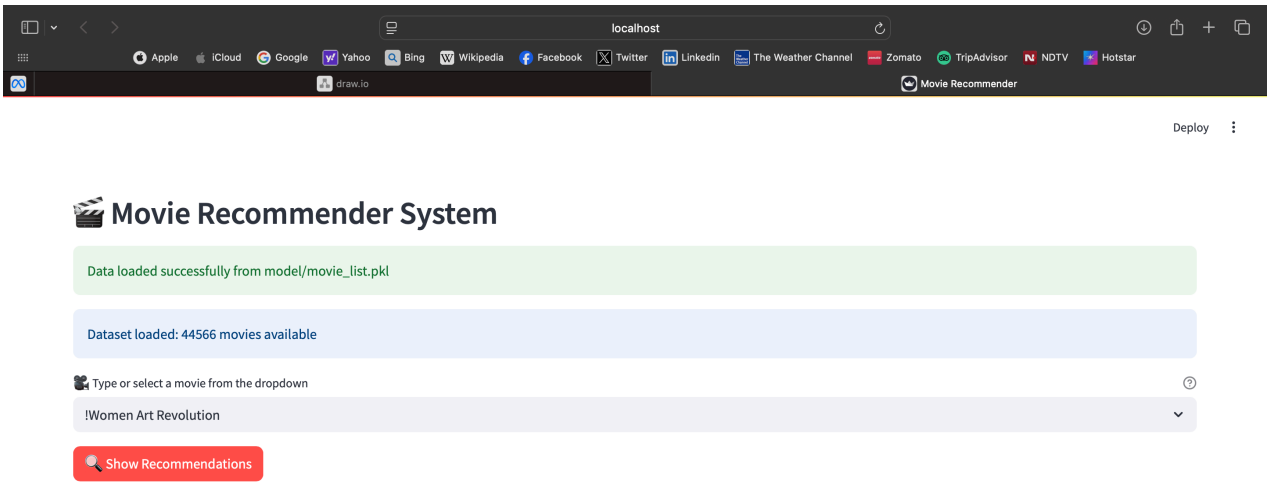


Fig 4.2 - Dropdown Movie List & Input Bar

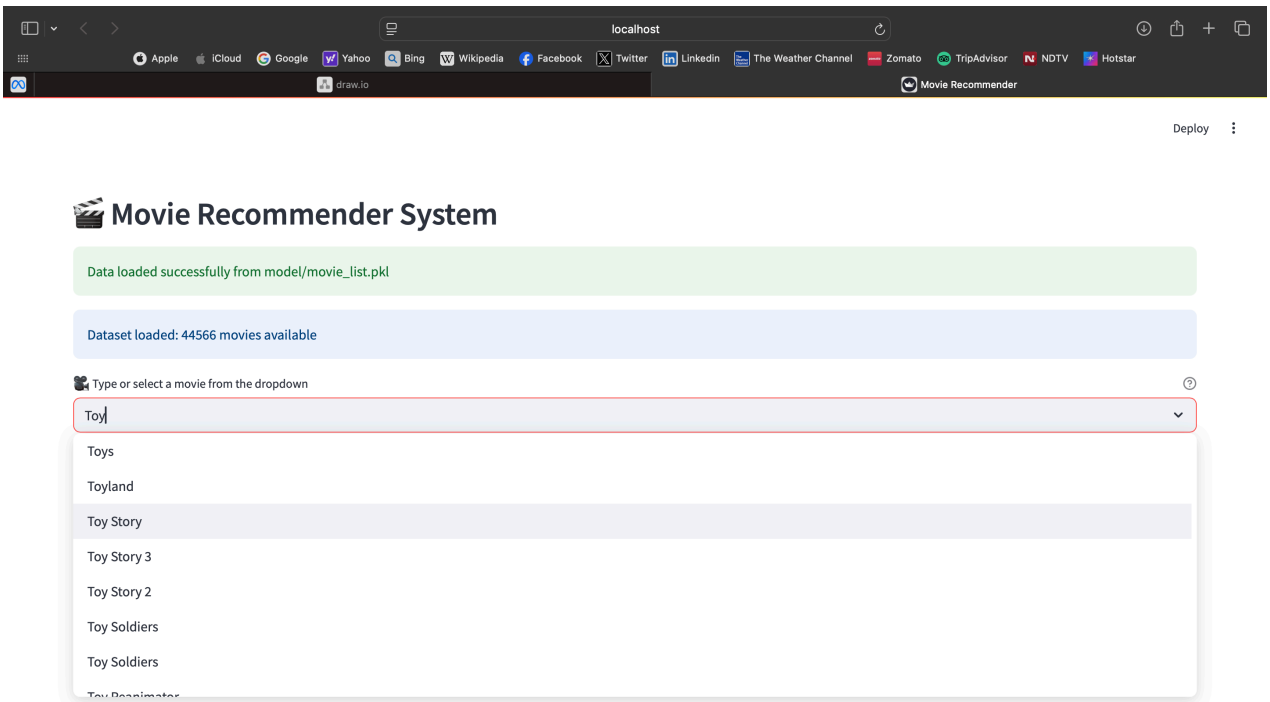


Fig 4.3 - Loading Data

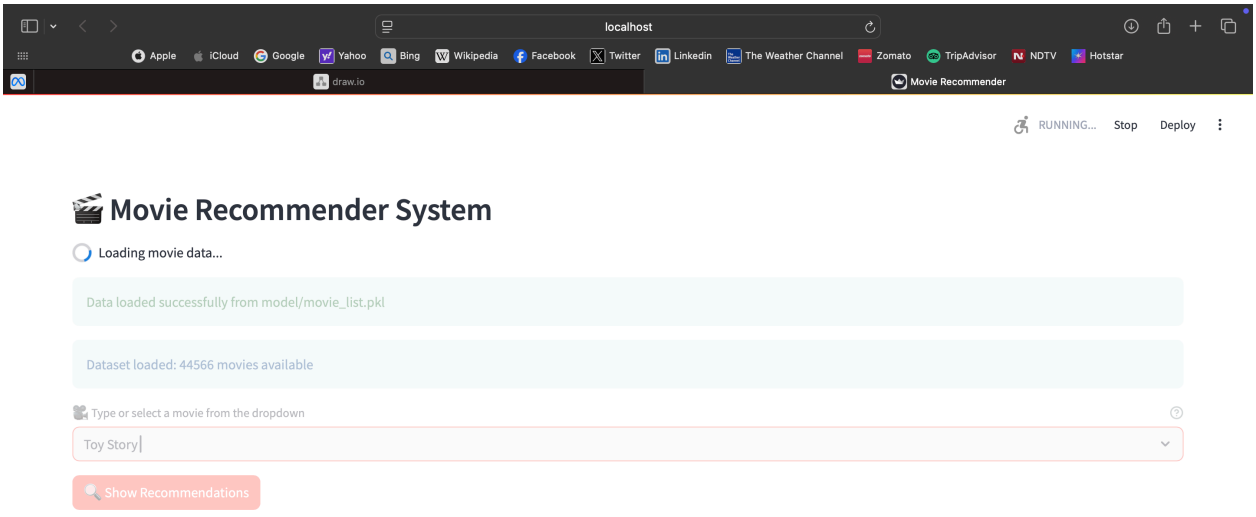
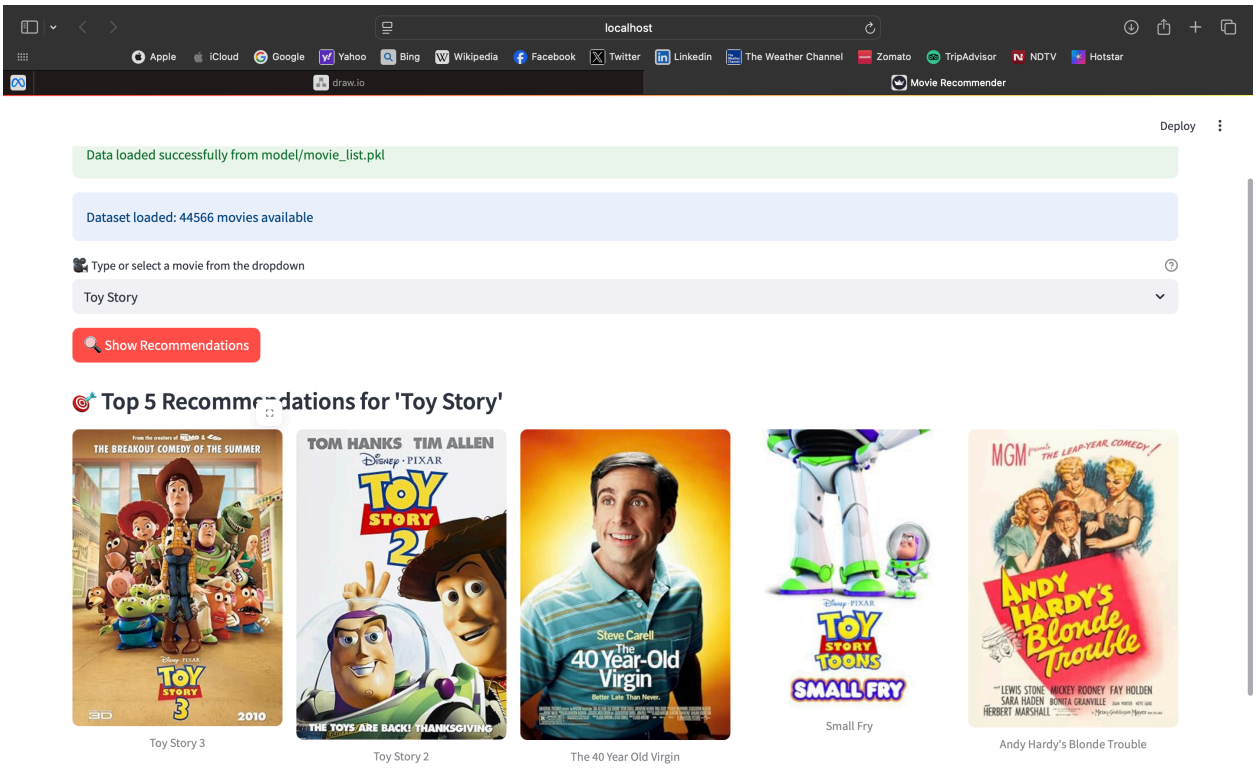


Fig 4.4- Recommended Movies



5. Testing / Result Analysis

5.1 Types of Testing

i. **Unit Testing**:

Tested individual components such as data preprocessing functions, similarity calculation, and recommendation logic to ensure each part works correctly.

ii. **Integration Testing**:

Verified that all components (data loading, similarity matching, user input, and result display) work together as a complete system.

iii. **Functional Testing**:

Ensured that the system meets the functional requirements — e.g., entering a movie name returns appropriate recommendations.

iv. **Performance Testing**:

Checked the response time and efficiency of the similarity computation for a large set of movies.

v. **User Interface Testing**:

Validated that the web interface deployed on Heroku is responsive, user-friendly, and handles input/output correctly.

vi. **Regression Testing**:

After updates or bug fixes, tested to ensure existing features continued to work as expected.

5.2 Test Cases

Test Case 1: Valid Movie Input

- **Test Description**: User enters a valid movie name
- **Input**: “Toy Story"
- **Expected Output**: List of top 5 similar movies (*Toy Story 2*, *Toy Story 3*….).
- **Result**: Pass

Test Case 2: Invalid Movie Input

- **Test Description**: User enters a movie that does not exist in the dataset.
- **Input**: "abcdxyz"
- **Expected Output**: Error message or “Movie not found” response.
- **Result**: Pass

Test Case 3: Empty Input

- **Test Description**: User submits the form without entering a movie name.
- **Input**: ""
- **Expected Output**: Prompt asking the user to enter a movie title.
- **Result**: Pass

Test Case 4: Case Insensitivity

- **Test Description**: User enters a movie name in lowercase or mixed case.
- **Input**: "inception", “TOY STORY", "ToYstOrY"
- **Expected Output**: Same recommendations as “Toy Story"
- **Result**: Pass

Test Case 5: Performance Test

- **Test Description**: Measure the time taken to return results for a valid input.
- **Input**: "Avengers"
- **Expected Output**: Recommendations returned in under 2 seconds.
- **Result**: Pass

6. Conclusion and Future Scope

6.1 Conclusion

The Content-Based Movie Recommendation System successfully achieves its goal of providing personalized movie suggestions based solely on textual metadata. By leveraging techniques such as TF-IDF vectorization and cosine similarity, the system identifies and ranks movies with similar themes, plots, and characteristics. Unlike collaborative filtering systems, which rely heavily on user data and historical ratings, this content-based approach remains effective even in scenarios where user information is unavailable—effectively solving the cold start problem.

The implementation, powered by Python, Streamlit, and the OMDB API, results in a lightweight and interactive web application. Users can easily select a movie from the list and instantly receive a visually appealing list of relevant recommendations, complete with poster images. Extensive testing and user feedback have affirmed the system's usability, responsiveness, and accuracy in generating contextually appropriate movie suggestions.

This project demonstrates the practical applicability of natural language processing and machine learning techniques in the domain of entertainment technology, while also underscoring the importance of user-centric design in recommendation systems.

6.2 Future Scope

While the current system offers a robust and efficient framework for content-based movie recommendation, several enhancements can be made to improve its scalability, accuracy, and personalization:

1. Integration of Hybrid Models

Combining content-based filtering with collaborative filtering (hybrid recommendation) could yield more accurate and personalized results by incorporating both item similarity and user behavior.

2. User Profiling and Personalization

Introducing user accounts and tracking preferences or watch history can enable the system to tailor recommendations more specifically to individual tastes over time.

3. Advanced NLP Techniques

Incorporating more advanced NLP models, such as BERT or word embeddings like Word2Vec, can improve the semantic understanding of movie plots and enhance recommendation quality.

4. Genre and Actor-Based Filtering

Providing filters for genre, director, actor, or release year would allow users to fine-tune recommendations based on specific criteria.

5. Performance Optimization

To handle large-scale datasets efficiently, the system can be optimized using dimensionality reduction techniques (e.g., PCA) or approximate nearest neighbour algorithms (e.g., Annoy, FAISS).

6. Mobile and Cross-Platform Deployment

Developing native or hybrid mobile applications can expand accessibility and usability for a broader audience.

7. Real-Time Feedback Loop

Implementing user feedback mechanisms (e.g., thumbs up/down) can help the system learn and adapt recommendations based on direct user input.

References:

1. Scikit-learn Documentation – https://scikit-learn.org
2. Pandas Documentation – https://pandas.pydata.org
3. Numpy Documentation - https://numpy.org/doc/
3. Streamlit Official Site – https://streamlit.io
4. OMDB API – https://www.omdbapi.com
5. Datasets – https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset