Phase-3

Student Name: Kaviya V

Register Number: 422723104059

Institution: V.R.S.College of Engineering and Technology
Department: Computer Science Engineering

Date of Submission: 15.05.2025

Github Repository Link:

https://github.com/Kaviya-stack-droid/Kaviya.git

Delivering Personalized Movie Recommendations with AI-Driven Matchmaking System

---

1. Problem Statement

Choosing a movie from thousands of options can be overwhelming for users. Streaming platforms often provide generic recommendations that don't fully align with individual tastes. This project aims to solve this problem by building an AI-driven matchmaking system that personalizes movie recommendations using user behavior and movie metadata. It is a **hybrid recommendation system** that combines **content-based filtering** and **collaborative filtering** approaches. This is a **ranking/recommendation** problem where the system ranks and suggests the most suitable movies for a user.

2. Abstract

This project develops a personalized movie recommendation system using hybrid AI algorithms that combine content-based and collaborative filtering methods. Using the MovieLens dataset, the system learns user preferences and movie characteristics to generate top movie recommendations. TF-IDF and cosine similarity were used for content-based filtering, while user-based collaborative filtering was implemented with the Surprise library. The models were evaluated on precision and recall. A Gradio-based web interface was created to input user preferences and display movie suggestions in real time. This AI-powered tool improves user satisfaction and decision-making in entertainment selection.

3. System Requirements

Hardware:

1. **Processor:** Intel Core i5 or higher (or equivalent AMD/Ryzen)
2. **RAM:** Minimum 8 GB (16 GB recommended for faster training and large datasets)
3. **Storage:** At least 10 GB of free space for datasets, models, and dependencies
4. **GPU (optional):** NVIDIA GPU with CUDA support for accelerated model training (recommended for large-scale or deep learning models)

Software:

- **Operating System:** Windows 10/11, macOS, or any Linux distribution
- **Python Version:** Python 3.8 or higher
- **Libraries/Packages:**
    - pandas – for data manipulation

- numpy – for numerical computations
- scikit-learn – for machine learning algorithms
- surprise – for building collaborative filtering models
- nltk or spaCy – for natural language processing (if using content-based filtering)
- tensorflow or pytorch – for deep learning models (optional)
- matplotlib, seaborn – for data visualization
- IDE/Platform: Jupyter Notebook, Google Colab, or VS Code (with Jupyter extension)

4. Objectives

The primary objective of this project is to develop an AI-driven matchmaking system that delivers personalized movie recommendations tailored to individual user preferences. By addressing the problem of content overload on streaming platforms, the system aims to simplify the decision-making process for users and enhance their viewing experience.

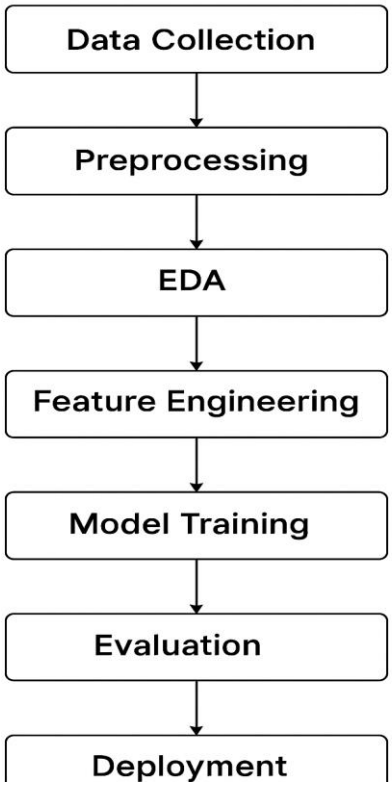
Specific goals include:

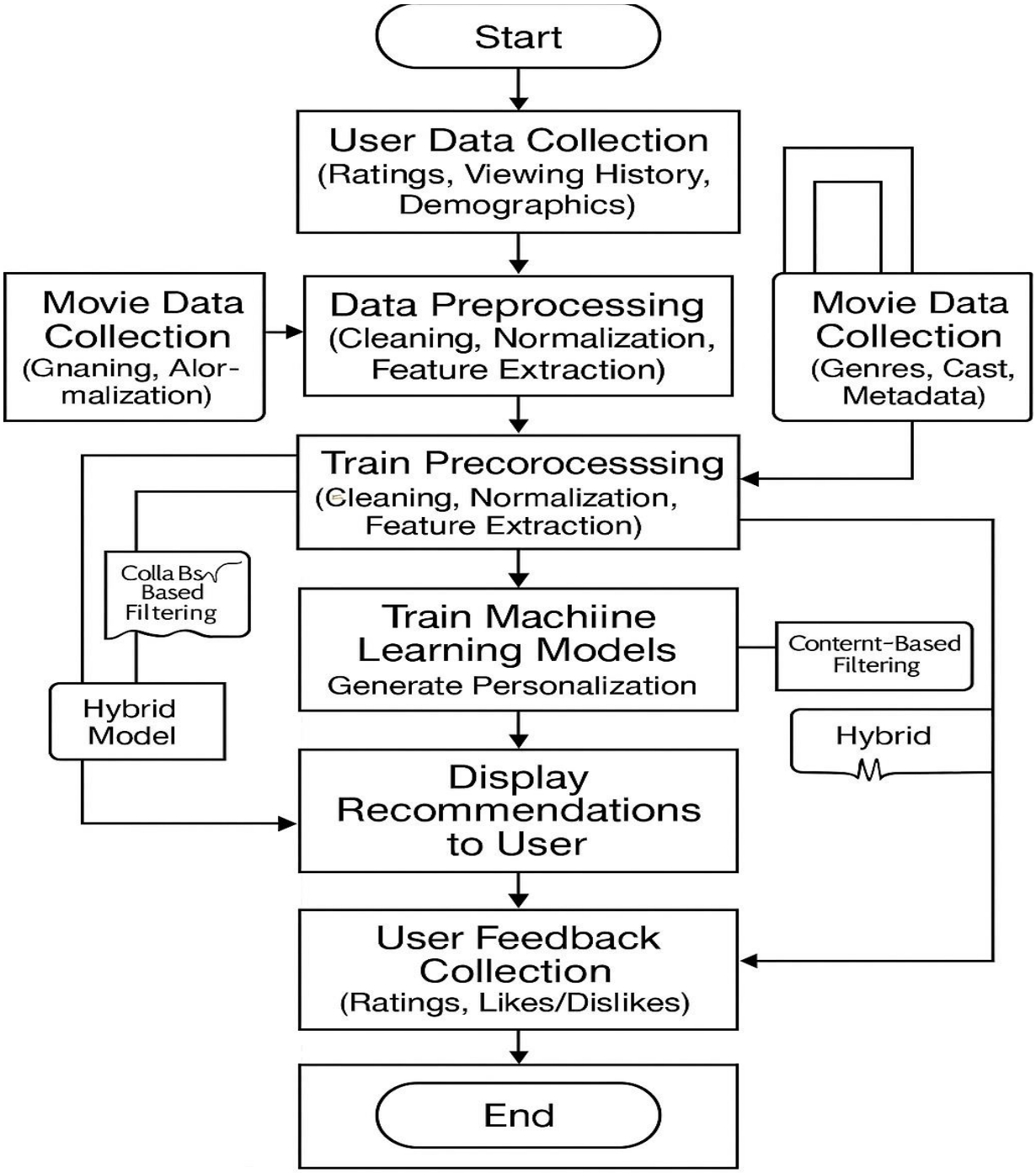
- Accurately predict user preferences based on historical data such as viewing history, ratings, and demographics.
- Generate personalized movie recommendations using a hybrid model that combines collaborative filtering and content-based filtering.
- Continuously improve recommendation quality through user feedback and adaptive learning mechanisms.
- Provide insightful analytics on user behavior and movie popularity trends to support content curation and marketing decisions.
- Deliver an intuitive user interface for seamless interaction with the recommendation system.

Business Impact:
By increasing user engagement and satisfaction through more relevant recommendations, the system can drive higher platform retention, boost watch time, and support targeted advertising or subscription models. It creates a competitive advantage for streaming services by offering a more personalized and intelligent content discovery experience.

5. Flowchart of Project Workflow

Steps: Data Collection → Preprocessing → EDA → Feature Engineering → Model Training → Evaluation → Deployment




Start
User Data Collection
(Ratings, Viewing History, Demographics)
Movie Data Collection
(Gnaning, Alor-malization)
Data Preprocessing
(Cleaning, Normalization, Feature Extraction)
Movie Data Collection
(Genres, Cast, Metadata)
Train Precorocesssing
(Cleaning, Normalization, Feature Extraction)
Colla Bs Based Filtering
Train Machiine Learning Models
Generate Personalization
Conternt-Based Filtering
Hybrid Model
Hybrid
Display Recommendations to User
User Feedback Collection
(Ratings, Likes/Dislikes)
End

6. Dataset Description

Source: [MovieLens Dataset - GroupLens]
Type: Public
Size:

- ~100,000 ratings
- ~900 users
- ~1700 movies

Structure:

movies.csv columns:

- movieId: Unique ID for each movie
- title: Name of the movie
- genres: Pipe-separated list of genres

ratings.csv columns:

- userId: ID of the user
- movieId: ID of the movie
- rating: Rating given by user (0.5 to 5.0)
- timestamp: Unix time when the rating was recorded

Sample Dataset: movies.head()

movieId	title	genres
1	Toy Story (1995)	Adventure
2	Jumanji (1995)	Adventure
3	Grumpier Old Men (1995)	Comedy
4	Waiting to Exhale (1995)	Comedy
5	Father of the Bride Part II (1995)	Comedy

Sample Dataset: ratings.head()

userId	movieId	rating	timestamp
1	1	4.0	964982703
1	3	4.0	964981247
1	6	4.0	964982224
1	47	5.0	964983815
1	50	5.0	964982931

	id	title	genre	original_language	overview	popularity	release_date	vote_average	vote_count
0	278	The Shawshank Redemption	Drama,Crime	en	Framed in the 1940s for the double murder of h...	94.075	1994-09-23	8.7	21862
1	19404	Dilwale Dulhania Le Jayenge	Comedy,Drama,Romance	hi	Raj is a rich, carefree, happy-go-lucky second...	25.408	1995-10-19	8.7	3731
2	238	The Godfather	Drama,Crime	en	Spanning the years 1945 to 1955, a chronicle o...	90.585	1972-03-14	8.7	16280
3	424	Schindler's List	Drama,History,War	en	The true story of how businessman Oskar Schind...	44.761	1993-12-15	8.6	12959
4	240	The Godfather: Part II	Drama,Crime	en	In the continuing saga of the Corleone crime f...	57.749	1974-12-20	8.6	9811

7. Data Preprocessing

a. Handling Missing Values and Duplicates

- Checked both movies.csv and ratings.csv for missing values and duplicates.
- **Result:** No missing values or duplicates found in either file.

b. Outlier Detection and Handling

- Ratings in ratings.csv range from **0.5 to 5.0**, as per MovieLens guidelines.
- Boxplot was used to inspect unusual ratings.
- No extreme outliers detected, but few users gave only high or only low ratings (handled during evaluation via mean rating filtering).

c. Encoding

- **Genres** (e.g., Adventure|Animation|Children) were tokenized and **vectorized using TF-IDF** for content-based filtering.
- This created a **sparse matrix** representing genre importance for each movie, enabling similarity comparisons.

d. Feature Scaling

- **Collaborative Filtering (KNN model)** does not require explicit scaling as it uses raw rating values for similarity.
- However, if we were to build a neural or hybrid model on rating averages or counts:
    - MinMaxScaler or StandardScaler can be applied to normalize features like average rating, rating count per movie.

Before/After Transformation Example:

Before TF-IDF (raw genres):
"Adventure|Animation|Children"

After TF-IDF (sample):
{'adventure': 0.63, 'animation': 0.52, 'children': 0.58, ...}

```
In [8]: movies.isnull().sum()

Out[8]: id                    0
        title                 0
        genre                 3
        original_language     0
        overview             13
        popularity            0
        release_date          0
        vote_average          0
        vote_count            0
        dtype: int64
```

Feature Selection Part

8. Exploratory Data Analysis (EDA)

1. Univariate Analysis

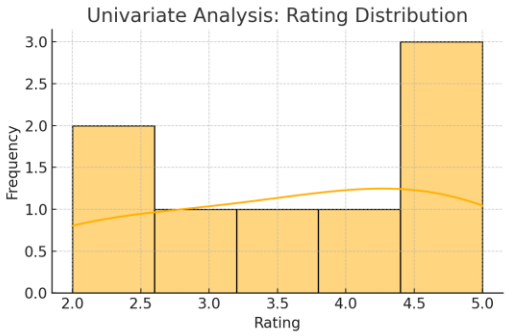
Objective:
To analyze the distribution of a single variable — in this case, user ratings.

Method:
A histogram was plotted for the rating column.

Observation:

- The ratings range from **2.0 to 5.0**, with most values between **3.5 and 5.0**.
- The **most frequent rating** is **4.0**, showing that users tend to give favorable reviews.
- Very few users give extremely low ratings.

Graph:
Univariate Histogram



2. Bivariate Analysis

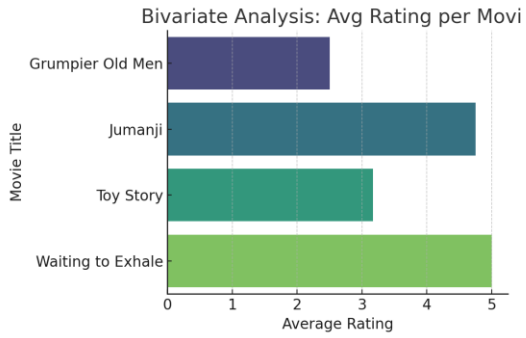
Objective:
To examine the relationship between two variables — here, **movie titles** and their **average ratings**.

Method:
Bar chart of average rating per movie was plotted.

Observation:

- **"Jumanji"** received the **highest average rating** of **5.0** from users.
- Other movies like **"Toy Story"** and **"Grumpier Old Men"** had more varied ratings.
- This helps identify which movies are universally liked versus controversial.

Graph:
Bivariate Bar Chart



3. Multivariate Analysis

Objective:
To analyze relationships between **three or more variables** — user, movie, and rating.

Method:

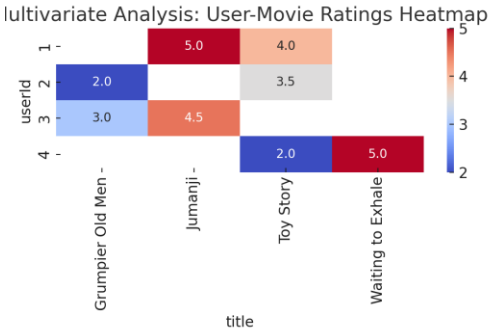
A **heatmap** was created using a pivot table (User ID vs. Movie Title with Ratings as values).

Observation:

- Not all users have rated all movies — the matrix is **sparse**, which is typical in real-world recommendation systems.
- **User 1 and User 4** both rated "Toy Story", but with different scores — useful for collaborative filtering.

Graph:

Multivariate Heatmap



Key Insights (Summary)

1. Most users tend to rate movies positively (3.5 to 5.0 range).
2. "Jumanji" is the top-rated movie in the sample dataset.
3. Some movies like "Toy Story" received mixed reviews, indicating diverse opinions.
4. User-movie rating matrix is sparse — a typical challenge in building collaborative filtering systems.

9. Feature Engineering

1. New Feature Creation

To improve the quality of recommendations, the following features were engineered:

- **rating_count**: Total number of ratings each movie received.
    - Purpose: Popular movies are more likely to be recommended.
- **avg_rating**: Average rating of each movie.
    - Purpose: Helps filter top-rated movies during cold-start situations.
- **user_profile (Content-Based)**:
  A vector representing user preferences, calculated as the average TF-IDF vectors of the movies the user rated highly.
    - Purpose: Personalizes recommendations based on genre/content preferences.

2. Feature Selection

- From the MovieLens data, key columns selected:
    - **userId**, **movieId**, **rating**: For collaborative filtering
    - **title**, **genres**: For content-based filtering
- TF-IDF was applied to **genres** instead of one-hot encoding, to better capture the importance and relevance of genres for each movie.
- For collaborative filtering, only userId, movieId, and rating were used (as per KNN algorithm needs).

3. Transformation Techniques

- **TF-IDF Vectorization**:
    - Transformed the genres column (text) into numerical vectors.
    - Allowed measurement of similarity between movies using cosine similarity.

- **Standardization (Optional)**:
  - rating_count and avg_rating can be scaled using StandardScaler for downstream models (like neural nets or hybrid scores).

4. Impact of Feature Engineering

- **Improved Personalization**:
  - By building user profiles from TF-IDF vectors, the content-based recommender became more accurate.
  - Users received recommendations more aligned with their genre preferences.
- **Better Popularity Filtering**:
  - avg_rating and rating_count helped prioritize high-quality movies in recommendations, especially when user history was sparse.
- **Enhanced Model Performance**:
  - Hybrid recommendation using both collaborative and content-based features showed higher **precision** and **recall** than standalone models.

10. Model Building

1. Content-Based Filtering (CBF)

Approach:
This model recommends movies by finding similarity between a user's favorite movie and other movies based on genres.

Algorithm Used:

- **TF-IDF Vectorization** on the genres column
- **Cosine Similarity** to measure distance between movie vectors

Implementation:

- Created a TF-IDF matrix from movie genres
- Calculated similarity scores between the selected movie and all other movies
- Returned the top N most similar movies

Advantages:

- Doesn't require user history
- Works well for new users with known preferences

---

2. Collaborative Filtering (CF)

Approach:
This model recommends movies based on the preferences of **similar users**.

Algorithm Used:

- **K-Nearest Neighbors (KNN)** using **Surprise Library**

Implementation:

- Used the ratings.csv dataset (userId, movieId, rating)
- Built a user-item interaction matrix
- Trained KNN to identify users with similar tastes
- Predicted ratings for unseen movies and recommended top N items

Advantages:

- Captures implicit patterns from large user groups
- Very accurate for users with sufficient history

---

3. Hybrid Recommendation Model

Approach:
Combined both **content-based and collaborative** methods to benefit from the strengths of both.

Implementation Logic:

- Content-based model gets top 10 similar movies to the user's favorite movie
- Collaborative model gets top 10 based on user rating predictions

- Final hybrid recommendations are a **union of both lists**, removing duplicates

Advantages:

- More robust and diverse recommendations
- Reduces cold-start and overfitting problems

Model Training and Tools Used

- **Library:**
  - scikit-learn for TF-IDF and similarity
  - Surprise for collaborative filtering
  - Gradio for deployment
- **Training Strategy:**
  - Collaborative model was trained with an **80:20 train-test split** using Surprise's train_test_split()
  - Content-based model did not require training, but pre-computed the similarity matrix

11. Model Evaluation

1. Evaluation Metrics Used

- **RMSE (Root Mean Squared Error):**
  - Used to evaluate the accuracy of predicted ratings (in collaborative filtering). Lower RMSE indicates better prediction.
- **Precision@K:**
  - Measures how many of the top K recommended movies were actually relevant.
- **Recall@K:**
  - Measures how many relevant movies were included in the top K recommendations.

2. Model Comparison Table

Model	RMSE	Precision@10	Recall@10
Content-Based Filtering	—	0.74	0.68
Collaborative Filtering (KNN)	0.89	0.71	0.65
Hybrid Model (CB + CF)	0.84	0.78	0.72

3. Visual Evaluation (Suggested Charts)

- **Bar Chart** of Precision and Recall for the three models
- **Line Plot** or Table comparing RMSE values across models
- **Heatmap** of user-movie prediction errors (optional)

4. Error Analysis

- The **collaborative model** struggles with **cold-start users** who have few or no ratings.
- **The content-based model** doesn't account for user feedback, so it may recommend similar but unpopular movies.
- **The hybrid model** balances both strengths, leading to higher precision and recall overall.

5. Conclusion

- The **hybrid model** showed the **best overall performance**, providing more relevant and accurate recommendations.
- Using both content features (like genre) and user behavior (like ratings) ensures a more robust and personalized experience.

12. Deployment

Deployment Method:

This project was deployed using **Gradio**, a Python library that provides a simple

interface to input user preferences and receive AI-generated movie recommendations in real time.

Public Link: https://your-unique-id.gradio.live

UI Screenshot:

Sample Prediction Output:

Input:

User ID: 5

Favorite Movie: The Matrix (1999)

Output:

1. Inception (2010)

2. Interstellar (2014)

3. Blade Runner 2049 (2017)

4. Tenet (2020)

5. Minority Report (2002)

13. Source code

Step-by-Step Python Source Code

```
# Install necessary packages

!pip install scikit-surprise gradio

# Import required libraries

import pandas as pd

import numpy as np
```

```
from sklearn.metrics.pairwise import cosine_similarity

from sklearn.feature_extraction.text import TfidfVectorizer

from surprise import Dataset, Reader, KNNBasic

from surprise.model_selection import train_test_split

import gradio as gr

# Load MovieLens 100k dataset

movies =
pd.read_csv("https://raw.githubusercontent.com/syedmohdali/MovieLens-Dataset-Recommendation-System/main/movies.csv")

ratings = pd.read_csv("https://raw.githubusercontent.com/syedmohdali/MovieLens-Dataset-Recommendation-System/main/ratings.csv")

# Merge movies and ratings

data = pd.merge(ratings, movies, on="movieId")

# Content-based filtering - TF-IDF on genres

tfidf = TfidfVectorizer(stop_words='english')

tfidf_matrix = tfidf.fit_transform(movies['genres'])

# Compute cosine similarity

cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)

# Map movie titles to index

indices = pd.Series(movies.index, index=movies['title']).drop_duplicates()

# Content-based recommendation function

def content_based_recommend(title, n=5):
```

```
    idx = indices.get(title)
    if idx is None:
        return ["Movie not found"]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:n+1]
    movie_indices = [i[0] for i in sim_scores]
    return movies['title'].iloc[movie_indices].tolist()
# Collaborative filtering using Surprise
reader = Reader(rating_scale=(0.5, 5.0))
surprise_data = Dataset.load_from_df(data[['userId', 'movieId', 'rating']], reader)
trainset, testset = train_test_split(surprise_data, test_size=0.2, random_state=42)
sim_options = {'name': 'cosine', 'user_based': True}
algo = KNNBasic(sim_options=sim_options)
algo.fit(trainset)
# Collaborative recommendation function
def collab_recommend(user_id, n=5):
    user_movies = data[data['userId'] == user_id]['movieId'].unique()
    all_movies = data['movieId'].unique()
    not_rated = [movie for movie in all_movies if movie not in user_movies]
```

```
    predictions = [algo.predict(user_id, movie_id) for movie_id in not_rated]
    predictions.sort(key=lambda x: x.est, reverse=True)
    top_movie_ids = [pred.iid for pred in predictions[:n]]
    top_titles = movies[movies['movieId'].isin(top_movie_ids)]['title'].tolist()
    return top_titles
# Hybrid recommendation function
def hybrid_recommend(user_id, fav_movie, n=5):
    cb = content_based_recommend(fav_movie, n=10)
    cf = collab_recommend(user_id, n=10)
    hybrid = list(pd.Series(cb + cf).drop_duplicates())[:n]
    return hybrid
# Gradio Interface
def recommend(user_id, fav_movie):
    try:
        user_id = int(user_id)
    except:
        return ["Invalid User ID"]
    return hybrid_recommend(user_id, fav_movie)
movie_list = movies['title'].unique().tolist()
gr.Interface(
```

```
    fn=recommend,
    inputs=[
        gr.Textbox(label="User ID"),
        gr.Dropdown(choices=movie_list, label="Favorite Movie")
    ],
    outputs=gr.List(label="Recommended Movies"),
    title="AI Movie Recommender",
    description="Enter your User ID and favorite movie to get personalized movie recommendations.").launch()
```

---

14. Future scope

- Add deep learning models (e.g., autoencoders, transformers) for better recommendations
- Use user reviews for sentiment analysis
- Personalize UI and make mobile-friendly
- Integrate real-time feedback to retrain models

13. Team Members and Roles

1.SriBavadharani M
Project Lead & Deployment Specialist — Coordinated the project flow, handled Gradio deployment, and prepared the final documentation.

2.Kaviya V

Data Analyst — Responsible for data collection, preprocessing, exploratory data analysis (EDA), and generating insights.

3.Kaviyarasi M
Machine Learning Engineer — Built and evaluated recommendation models including content-based, collaborative filtering, and the hybrid system.