Team 6:

Andrew Jaynes
Nitu Bola
Riley Laughlin
Val Keeranan

MOVIE RECOMMENDATION SYSTEM

Project 4

TABLE OF CONTENTS

INTRO

This project aims to build a movie recommendation system that provides personalized suggestions by using content-based and collaborative filtering. It will use machine learning techniques, including Text Vectorization (TF-IDF), the Alternating Least Squares (ALS) model and matrix factorization, to analyze movie attributes and user ratings. The system will be developed using Python, Pandas, PySpark, Scikit-learn, NumPy, Flask, HTML and CSS.

DATA PROCESSING

Data Collecting:

- The Movie Database (TMDB) API themoviedb.org
- Movielens grouplens.org/datasets/movielens

Ethics & Usage:

- The TMDB API is free to use for non-commercial purposes as long as you attribute TMDB as the source of the data.
- The MovieLens database is hosted for the purpose of developing and evaluating novel techniques for recommendation and personalization.

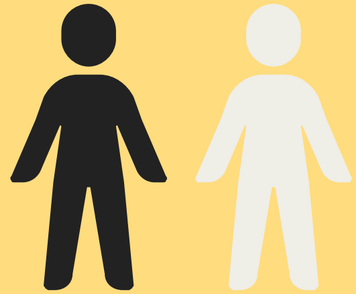
Data Cleaning: Uploaded the data into a Spark Data Frame and filtered for movies where the popularity score is greater than 5 - which reduced the number of rows from 1M+ to 7.6K

MACHINE LEARNING MODELS

For our movie recommendation systems we took a hybrid approach to identify which machine learning model would work best:

Content-based filtering leverages machine learning techniques to identify movies with similar genres and plot descriptions. First, it converts movie overviews into **numerical vectors using TF-IDF (Term Frequency-Inverse Document Frequency)**, which highlights important words in each description. Then, it calculates the similarity between these vectors using **cosine similarity**. Finally, it identifies the movie with the highest overall similarity score.

Collaborative filtering leverages user-item interaction data, such as ratings, to predict user preferences. **Matrix factorization, specifically the Alternating Least Squares (ALS)** decomposes the sparse rating matrix into latent factor matrices, representing hidden user and item preferences. By iteratively optimizing these matrices to minimize prediction errors, the model infers missing ratings, enabling personalized recommendations based on shared interaction patterns.

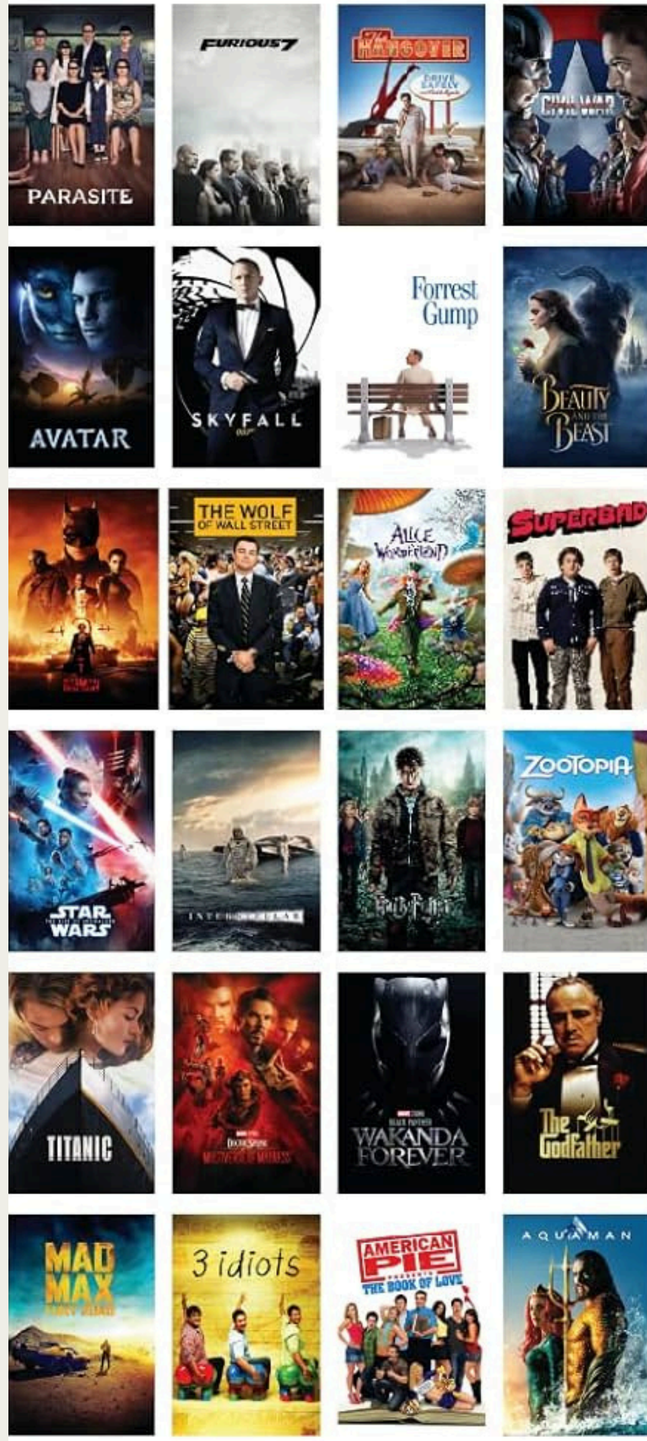


CONTENT-BASED

Content-based filtering, recommends movies based on their inherent attributes. It utilizes the TMDB API to retrieve movies filtered by genre, mood, popularity, and vote count.

Genre and mood mappings rely on predefined dictionaries, translating user inputs into TMDB genre IDs. Then sorts the resulting movies based on popularity or vote average, returning the top 10.

A rule-based augmentation is introduced through the mood parameter, mapping moods to genre combinations. This approach, while not traditional machine learning, enhances genre filtering by incorporating mood-based preferences.



Step 1

Text Vectorization (TF-IDF):

- The TfidfVectorizer is central to this process. It converts movie descriptions into numerical vectors, where each dimension represents a word, and the value indicates the word's importance (TF-IDF score).
- TF-IDF effectively extracts and weights the most relevant words, reducing the dimensionality of the text data while preserving important semantic information.
- The stop_words='english' parameter removes common, uninformative words, further improving the quality of the vectors.

Step 2

Cosine Similarity:

- The cosine_similarity function calculates the similarity between the TF-IDF vectors of movie descriptions.
- This measures the angle between vectors; smaller angles (higher cosine values) indicate greater similarity.
- This is the core of the similarity calculation. The argmax of the summed similarity matrix gives the index of the most similar movie.

Step 3

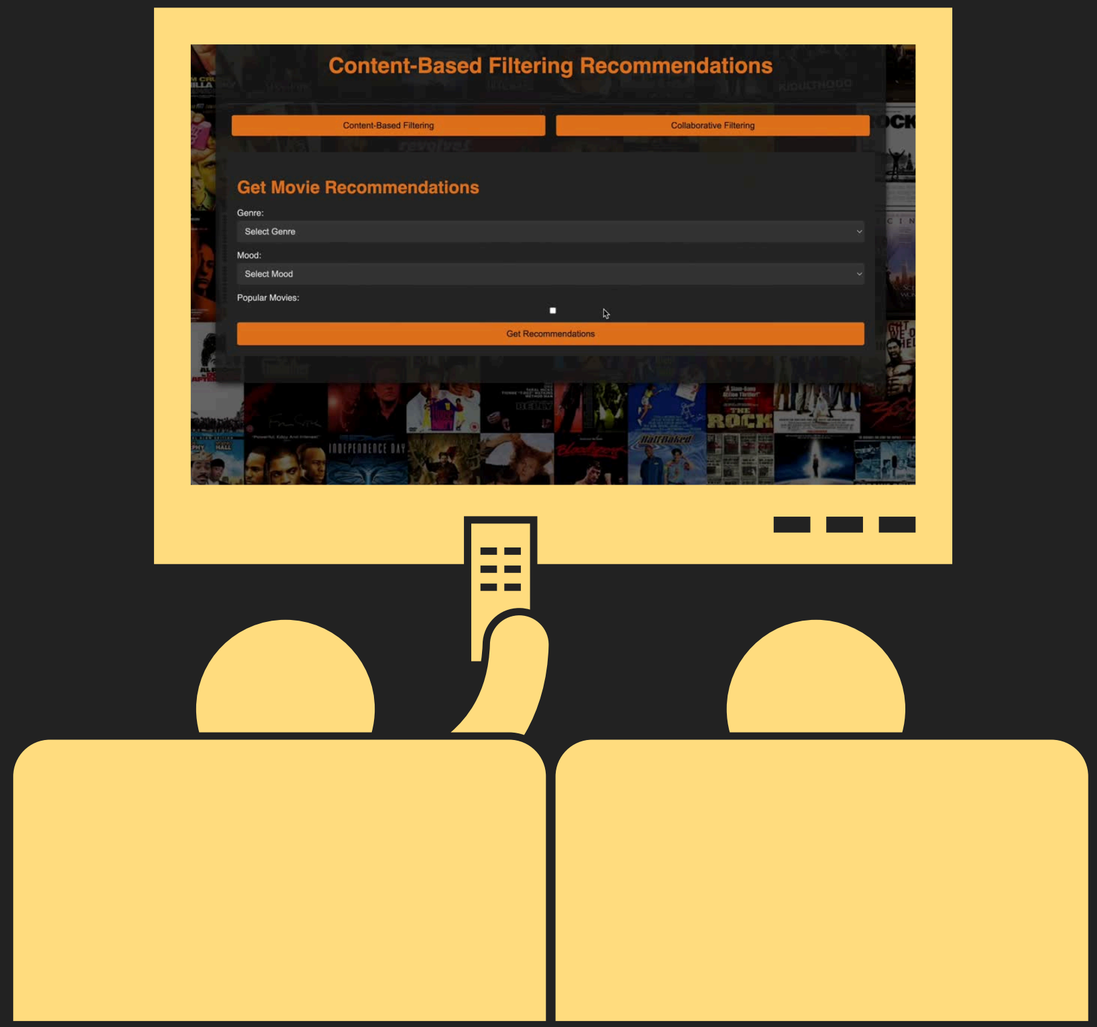
Selection of Most Similar Movie:

- The code identifies the movie with the highest cumulative similarity to all other movies in the input list. This is the "recommendation."
- There is no training of a model, but rather a calculation of similarity based on the existing text data.

RESULTS

Here’s a preview of the Flask web application that recommends movies based on user-selected genres, moods, or popularity, leveraging the TMDB API to fetch movie data.

The application serves a simple HTML interface to collect user preferences and display the recommended movies.



COLLABORATIVE

What is Collaborative Filtering?

Collaborative Filtering (CF) is a recommendation system that suggests items based on the preferences of similar users. ***It's like asking your friends for movie suggestions - you trust their recommendations because their tastes align with yours!***

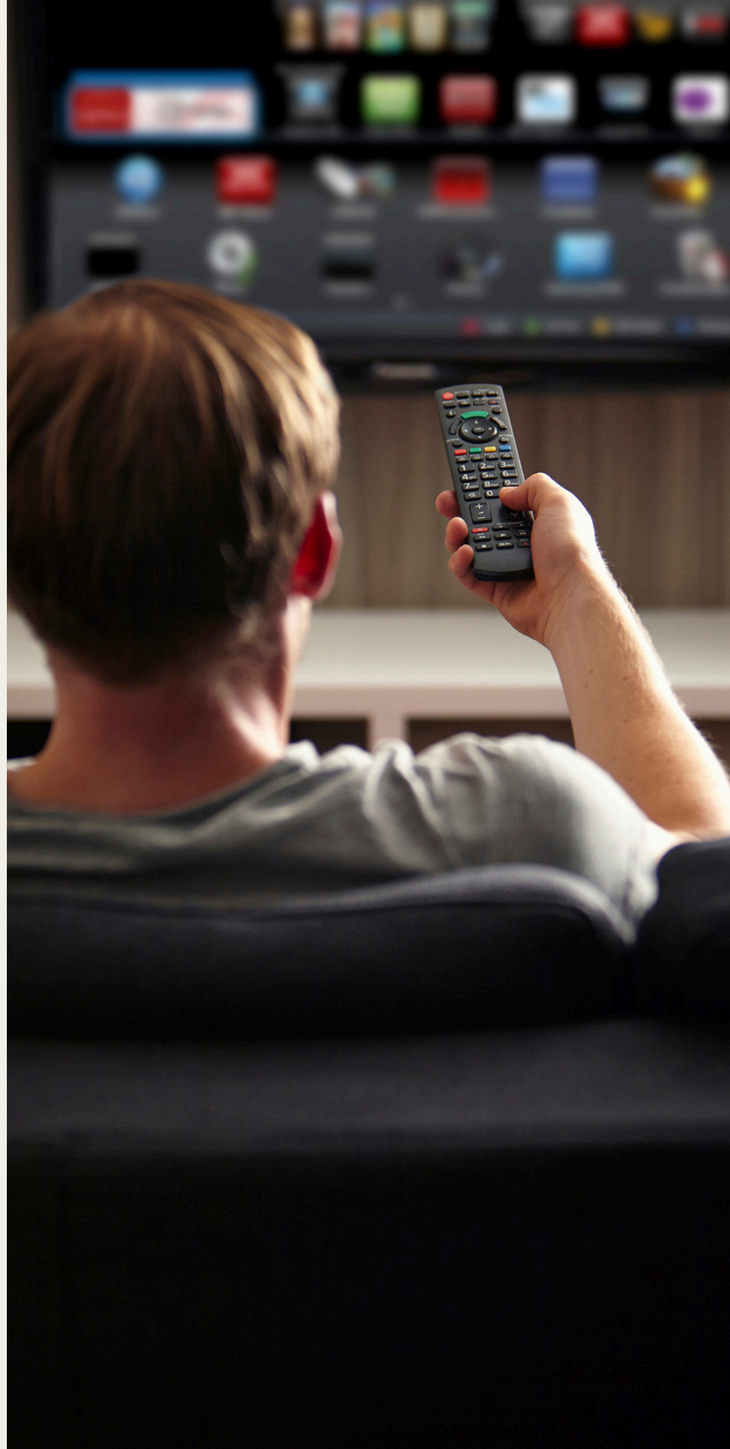
How Does Collaborative Filtering Work?

1. **Find Similar Users:** Looks for users who have rated movies similarly.
2. **Make Predictions:** Recommends movies based on what similar users liked.
3. **Two Types:**
    - User-based CF: Finds users with similar tastes.
    - Item-based CF: Recommends similar items based on ratings.

ALS is widely used for matrix factorization in collaborative filtering. It alternates between fixing user matrix (U) and item matrix (V), then optimizing the error by minimizing the difference between the actual and predicted ratings.

Where Is It Used?

- **Netflix:** Suggests movies and shows based on users' past ratings.
- **Amazon:** Recommends products based on what other customers liked.
- **Spotify:** Creates playlists based on music preferences of similar listeners.



Step 1

The Alternating Least Squares (ALS) algorithm iteratively optimizes latent factor matrices, alternating between:

1. Fixing item latent factors, optimize user latent factors.
2. Fixing user latent factors, optimize item latent factors.

For each step, it uses least squares regression to minimize the difference between the predicted ratings and the actual ratings in the training data.

Step 2

Loss Function:

- The loss function measures the error between predicted and actual ratings (commonly RMSE or r2).
- The **regParam** adds regularization to the loss function to prevent overfitting and control the complexity of the model.

Step 3

Iterative Updates:

- The ALS algorithm repeats optimization steps until it converges or reaches the **maxIter limit.**
- Each iteration refines the latent factor matrices, reducing error and improving model accuracy.

Snippet of our optimized model:

```
als = ALS(
 userCol="userId",
 itemCol="tmdbId",
 ratingCol="rating",
 maxIter=18, # Keeping iterations constant
 rank=100,
 regParam=0.05, # Regularization
 alpha=0.5, # For implicit feedback
 coldStartStrategy="drop"
)
```

EXAMPLE OF ALS IN ACTION

How ALS Works:

1. ALS analyzes the patterns between users and their movie preferences.
2. It fills in missing ratings by learning from users who have similar tastes.

Explanation of ALS Predictions

Alice:
Predicted lower rating for Romance (3 ⭐ for The Notebook): Alice prefers Action and Sci-Fi, so the model predicts she's less likely to enjoy Romantic movies.

Bob:
Predicted moderate ratings for Sci-Fi and Action (4 ⭐ for Inception and Avengers): Bob's Romantic preference is strong, but the model predicts moderate ratings for genres he hasn't rated much, like Sci-Fi and Action.

Charlie:
Predicted higher rating for Romance (4 ⭐ for The Notebook): Charlie shows a more balanced preference for multiple genres, including Romance, so the model predicts he'll enjoy Romantic movies more.

Original ratings table before ALS

User / Movie	Inception (Sci-Fi)	Titanic (Romance)	Avengers (Action)	The Notebook (Romance)
Alice	5 ⭐ (Actual)	3 ⭐ (Actual)	4 ⭐ (Actual)	?
Bob	? ⭐ (Actual)	5 ⭐ (Actual)	?	4 ⭐ (Actual)
Charlie	4 ⭐ (Actual)	5 ⭐ (Actual)	5 ⭐ (Actual)	?

ALS model output (after prediction)

User / Movie	Inception (Sci-Fi)	Titanic (Romance)	Avengers (Action)	The Notebook (Romance)
Alice	5 ⭐ (Actual)	3 ⭐ (Actual)	4 ⭐ (Actual)	3 ⭐ (Predicted)
Bob	4 ⭐ (Predicted)	5 ⭐ (Actual)	4 ⭐ (Predicted)	4 ⭐ (Actual)
Charlie	4 ⭐ (Actual)	5 ⭐ (Actual)	5 ⭐ (Actual)	4 ⭐ (Predicted)

Actual Output for our model

userId	tmdbId	rating	prediction
1	97	4.0	3.7280726
1	137	4.0	4.0945673
1	274	4.0	4.1607547
1	275	5.0	4.306444
1	275	5.0	4.306444

EVALUATION

The efficacy of the ALS recommendation model was evaluated using the R-squared metric. Upon successful generation of predictions, a RegressionEvaluator computed the R-squared score by comparing the model's predicted ratings against the actual ratings.

Initial Model Evaluation:

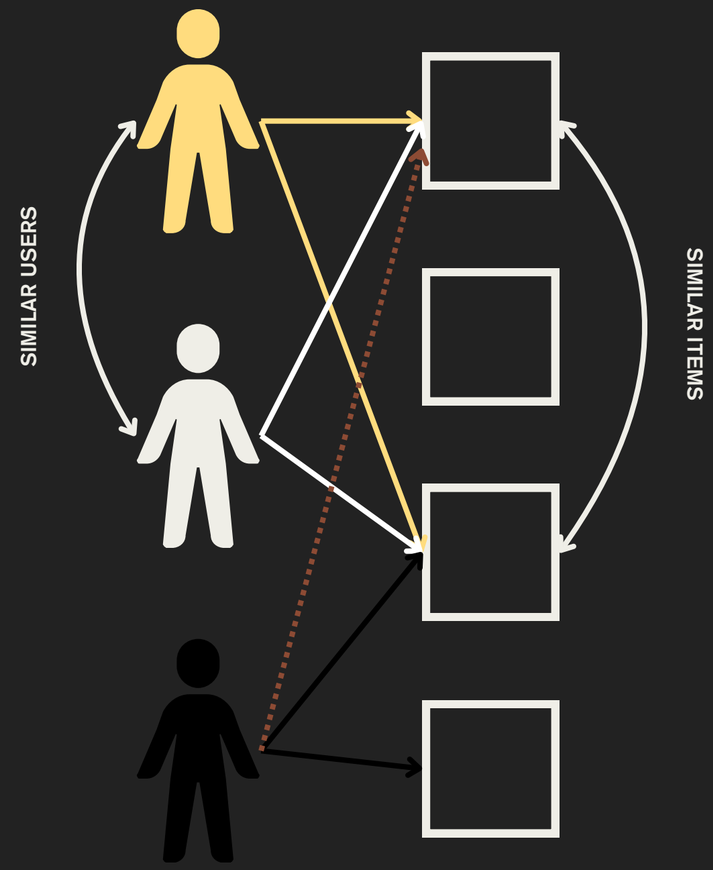
- R-Squared (R^2) Metric: Used to evaluate model performance.
  - Initial R^2 = 0.32 (Far from the required target, indicating poor predictive power).

Optimization Process:

1. Added Genre Feature:
   - Integrated the genre feature to capture user preferences based on movie categories, improving model relevance.
2. Adjusted Rank Parameter:
   - Increased rank to 100, allowing for better factorization and more accurate representation of user-movie interactions.

Optimized Model Performance:

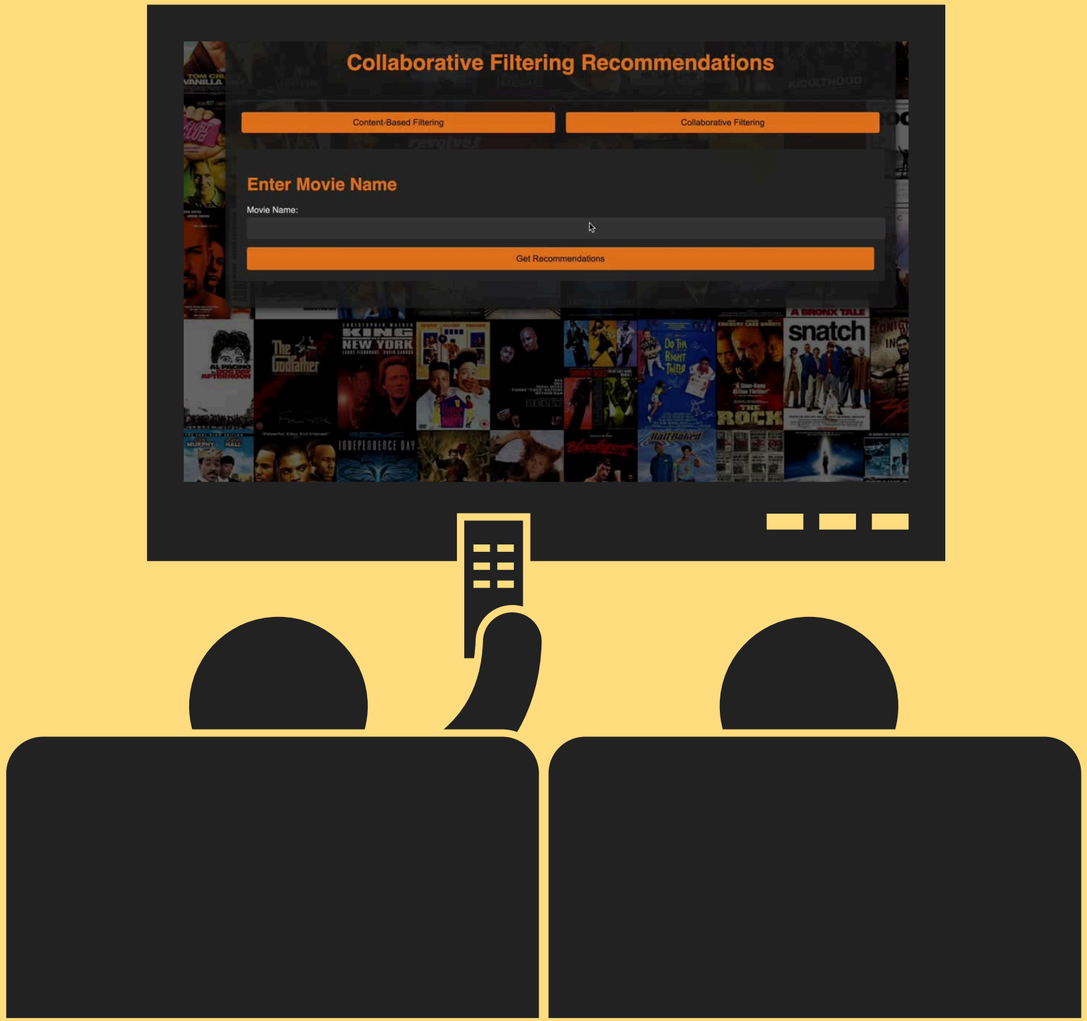
- New R^2 = 0.8116 (Achieved the required R^2 score, meeting the target for strong predictive power).



RESULTS

Here’s a preview of the Flask web application that recommends movies based on genre and user ratings, leveraging the TMDB API to fetch movie data and Movielens to fetch ratings data.

The application serves a simple HTML interface to collect user preferences and display the recommended movies.



CONCLUSION

The choice between content-based and collaborative filtering for movie recommendations hinges on the specific needs of the application.

Content-based methods prioritize item attributes and mitigate the cold-start issue, while collaborative filtering excels at capturing user preferences and serendipitous discoveries. However, collaborative filtering suffers from cold-start and sparsity problems.

By effectively blending these techniques, we can build robust movie recommendation engines that leverage item attributes, user preferences, and address inherent challenges like the cold-start problem, ultimately providing a richer and more personalized user experience.

Next Steps:

- Improve the ALS model performance and address the cold start problem
- Further optimize Flask app for better user experience and explore other hybrid recommendation techniques