Thesis submitted to the

UNIVERSITY OF MOHAMED BOUDIAF – MSILA

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

In partial fulfillment of the requirements for the degree of

Master in **Computer science**

By

KHODJA, Omar

KORICHI, Abdallah

Title of the thesis

A Movie Recommender System Based on Machine Learning and Deep Learning

Under the supervision of

BRAHIMI Mahmoud

Composition of the jury

First name Last name	University of Msila	President
First name Last name	University of Msila	Reporter
First name Last name	University of Msila	Examiner

05/ 2026.

Abstract

Digital entertainment platforms provide access to massive collections of movies and multimedia content, making personalized content discovery increasingly important. This work presents a hybrid movie recommendation system combining collaborative filtering based on ALS matrix factorization with content-based filtering using semantic embeddings and cosine similarity. The proposed system integrates web and mobile applications connected to a backend infrastructure and an intelligent recommendation engine capable of generating dynamic personalized recommendations. In addition, periodic retraining mechanisms were incorporated to continuously adapt recommendation models according to newly collected user interactions. Experimental results demonstrate the effectiveness of the proposed hybrid approach in improving recommendation relevance and personalization quality.

Key words: Recommender Systems, Hybrid Recommendation, Collaborative Filtering, Content-Based Filtering, ALS, Semantic Embeddings.

الملخص

ت وفر منصات الترفيه الرقمية إمكانية الوصول إلى مجموعات ضخمة من الأفلام والمحتوى المتعدد الوسائط، مما يجعل اكتشاف المحتوى المخصص أمرًا متزايد الأهمية. يقدم هذا العمل نظامًا هجينًا والتصفية القائمة على ALS لتوصية الأفلام يجمع بين التصفية التعاونية القائمة على تحليل مصفوفة المحتوى باستخدام التضمينات الدلالية وتشابه جيب التمام. يدمج النظام المقترح تطبيقات الويب والهواتف المحمولة المتصلة ببنية تحتية خلفية ومحرك توصية ذكي قادر على توليد توصيات ديناميكية مخصصة. بالإضافة إلى ذلك، تم دمج آليات إعادة تدريب دورية لتكييف نماذج التوصية باستمرار وفقًا لتفاعلات المستخدمين التي يتم جمعها حديثًا. تُظهر النتائج التجريبية فعالية النهج الهجين المقترح في تحسين ملاءمة التوصيات وجودة التخصيص..

الكلمات المفتاحية: أنظمة التوصية، التوصية الهجينة، التصفية التعاونية، التصفية القائمة على المحتوى، ALS التضمينات الدلالية،.

Dedications

“I dedicate this work to my beloved parents, whose boundless love, sacrifices, and encouragement have been the cornerstone of my journey.

To my family and friends, whose patience, kindness, and unwavering support carried me through every challenge and inspired me to persevere.

This achievement is not mine alone, but a reflection of the faith, strength, and hope you have all instilled in me.

May it stand as a humble tribute to your trust and devotion.”

KHODJA OMAR

“Some journeys are measured not by the distance traveled, but by the people who walk beside us through every difficult step, every moment of doubt, and every small victory along the way. This humble work is first dedicated to my family especially my parents, whose love, and endless support shaped the person I am today. No words could truly express the gratitude and respect I hold for them, nor repay everything they have done for me throughout my life. To my friends, for the support, laughter, and unforgettable moments that made this journey meaningful. Last but not least, I want to thank me. I want to thank me for believing in me, I want to thank me for doing all this hard work.”

Korichi Abdallah

Acknowledgments

Completing this work would not have been possible without the support, guidance, and encouragement of many people who accompanied us throughout this academic journey.

First of all, we express our deepest gratitude to ALLAH for granting us the strength, patience, and determination needed to complete this project.

We would like to sincerely thank our parents and families for their endless sacrifices, trust, and unconditional support. Their encouragement has always motivated us to move forward, especially during the most challenging moments.

Our heartfelt appreciation also goes to our friends and everyone who supported us throughout this journey with motivation, advice, and encouragement.

We would like to express our sincere respect and gratitude to our supervisor, **Brahimi Mahmoud**, for his valuable guidance, continuous support, and constructive remarks throughout the realization of this work. His advice and experience greatly contributed to the completion and improvement of this project.

Finally, we would like to thank the faculty and all the teachers who contributed to our academic formation and provided us with the opportunity to carry out this enriching work.

Contents

List of figures

List of Tabel

General Introduction

Choosing what to watch has become increasingly difficult in modern digital entertainment environments. Streaming platforms and online media services now provide access to massive collections of movies, series, and multimedia content available instantly to users worldwide. While this abundance of content improves accessibility and diversity, it also creates an information overload problem where discovering relevant and interesting content becomes a complex task.

To address this challenge, modern entertainment platforms rely on intelligent personalization technologies capable of analyzing user behavior and suggesting content adapted to individual preferences. Recommendation systems have therefore become essential components in digital platforms due to their ability to improve user experience, facilitate content discovery, and increase user engagement.

Several recommendation approaches have been proposed in the literature, including collaborative filtering and content-based filtering. Collaborative filtering exploits similarities between user interactions and behavioral patterns, while content-based filtering relies on item attributes and semantic information. Although both techniques have demonstrated significant effectiveness, each approach presents limitations when used independently, particularly in sparse data and cold-start scenarios.

This work proposes a hybrid movie recommendation system that combines collaborative filtering based on Alternating Least Squares (ALS) matrix factorization with content-based filtering using semantic embeddings and cosine similarity. The proposed approach aims to improve recommendation relevance, personalization quality, and recommendation robustness by integrating behavioral and semantic recommendation strategies within a unified framework.

The developed system integrates web and mobile applications connected to a backend infrastructure and an intelligent recommendation engine capable of generating dynamic personalized recommendations. In addition, periodic retraining mechanisms were incorporated to allow the recommendation models to continuously adapt to newly collected user interactions.

This thesis is organized as follows:

Chapter One: Recommender System - presents the theoretical background and state of the art related to recommender systems and hybrid recommendation techniques;

Chapter Two: System design - describes the design and architecture of the proposed recommendation system;

Chapter Three: System Implementation and Results - presents the implementation process, experimental evaluation, and obtained results of the proposed system.

Chapter 1 Recommendation Systems

1.Introduction

Watching a movie is no longer limited by availability, but rather by choice. Digital streaming platforms and online entertainment services now provide access to enormous collections of movies, series, and multimedia content available instantly to users worldwide. Although this accessibility transformed the entertainment experience, it also introduced a new challenge: identifying content that truly matches user interests among thousands of available options.

Modern entertainment platforms aim not only to provide content, but also to understand user preferences and viewing habits in order to deliver personalized experiences. Services such as Netflix and Amazon Prime Video heavily depend on intelligent recommendation technologies capable of analyzing user behavior and suggesting relevant content adapted to individual tastes [1].

Recommender systems (RSs) emerged as an effective solution to address information overload problems by filtering large amounts of data and predicting user preferences using machine learning and data analysis techniques [2]. These systems have become fundamental components in modern digital platforms due to their impact on personalization, user engagement, and content discovery.

Research in recommendation systems has also benefited from the availability of benchmark datasets containing real-world user interactions and movie metadata, allowing researchers to evaluate recommendation algorithms under realistic conditions [3].

This chapter presents the fundamental concepts of recommender systems, their principal recommendation approaches, and the main challenges associated with recommendation generation. It also introduces the evaluation methods and theoretical foundations related to the hybrid recommendation system proposed in this work.

2.Overview of Recommender Systems

RSs are software tools designed to suggest relevant items to users based on their preferences, behavior, or interactions. These items can include movies, products, music, or any type of digital content. The main goal of a recommender system is to reduce information overload and help users quickly find content that matches their interests [2].

In movie platforms such as Netflix and Amazon Prime Video, RSs play a central role by guiding users through large catalogs of movies. Instead of manually searching, users receive personalized suggestions based on their viewing history, ratings, and preferences. This improves user satisfaction and increases platform engagement [1].

A typical recommender system relies on three main types of data:

- **Users**: information about user profiles and preferences;
- **Items**: characteristics of the items (e.g., movie genres, descriptions);
- **Interactions**: user actions such as ratings, clicks, or watch history.

These components are often represented in the form of a user-item interaction matrix, where each entry reflects the relationship between a user and an item.

From a structural point of view, a RS generally follows a simple architecture:

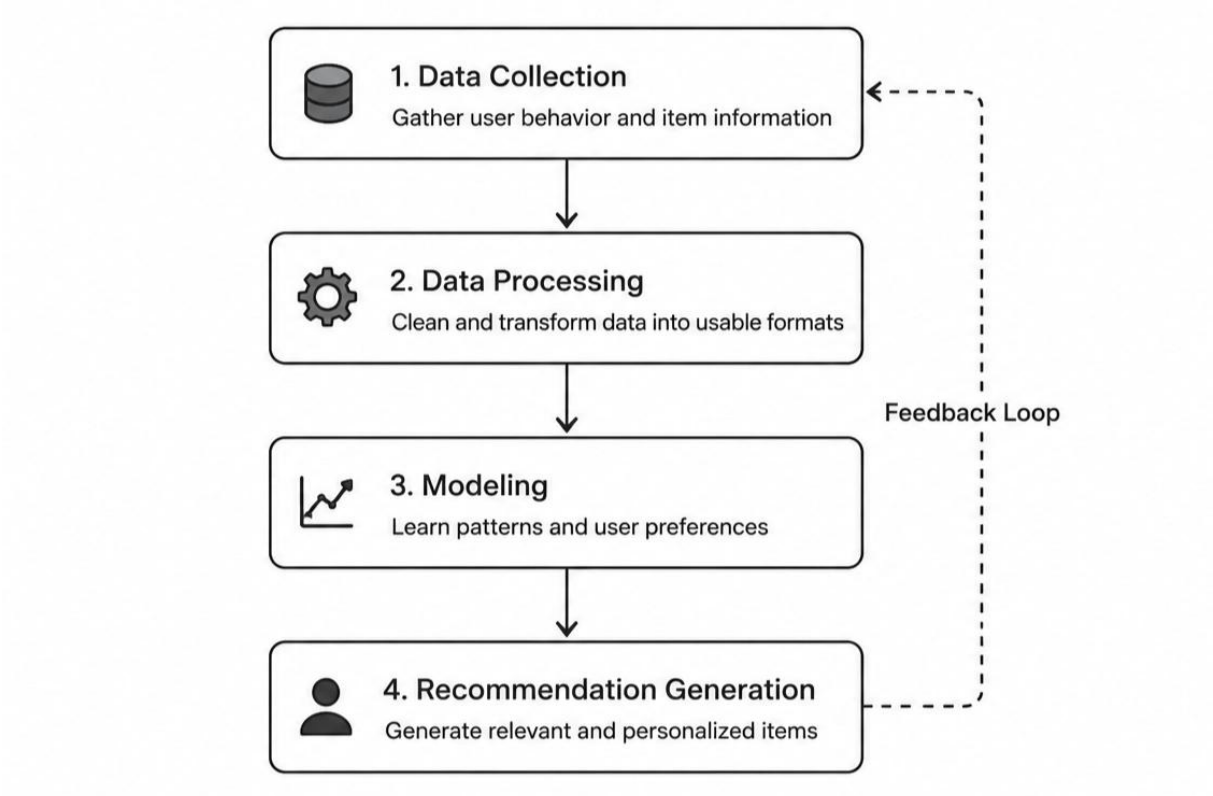

Figure 1:General Architecture of a Recommender System

The system continuously updates its recommendations as new data becomes available, allowing it to adapt to changing user preferences over time. In modern applications, RSs often integrate machine learning techniques to improve prediction accuracy and scalability [2].

Recommender systems act as intelligent filters that personalize user experience by analyzing large volumes of data and delivering relevant suggestions. Their effectiveness depends on the quality of data, the choice of algorithms, and the ability to adapt to user behavior.

3. Types of Recommender Systems

Recommender systems can be categorized into several types based on the techniques they use to generate recommendations. The most widely used approaches are content-based filtering, collaborative filtering, and hybrid methods. Each of these approaches models user preferences differently and presents specific advantages and limitations [2].

3.1 Content-Based Filtering

Content-based (CB) filtering recommends items by analyzing their features and matching them with a user's preferences. In the context of movie recommendation, items are described using attributes such as genres, keywords, or textual descriptions.

This approach builds a user profile based on previously liked items and recommends similar items by computing similarity between item features. Techniques such as vector space models and similarity measures (e.g., cosine similarity) are commonly used.

CB systems rely on item descriptions and user profiles to generate personalized recommendations without requiring data from other users [5].

One of the main advantages of this method is its ability to work independently of other users, which makes it suitable for scenarios with limited interaction data. However, it often suffers from over-specialization, where the system repeatedly recommends similar items and lacks diversity [5].

3.1.1 Feature Representation

Feature representation is a key component of CB filtering, as it defines how items are transformed into numerical vectors that can be compared. The effectiveness of the recommendation system depends heavily on how well these features capture the characteristics of items. Two main types of representations are commonly used: sparse and dense representations.

3.1.1.1 Sparse Representations

Sparse representations are traditional techniques used to encode item features in high-dimensional vectors. Common approaches include Bag of Words, one-hot encoding, and TF-

IDF, where each dimension corresponds to a specific feature such as a word or category. In these representations, most values are zero because each item contains only a small subset of all possible features.

These representations are simple and interpretable, but they suffer from high dimensionality and inefficient memory usage. They are widely used in early CB systems and remain useful for text-based features [5].

Characteristics:

- High-dimensional;
- Mostly zeros;
- Interpretable;
- Memory inefficient.

3.1.1.2 Dense Representations

Dense representations, also known as embeddings, are low-dimensional vectors learned from data. Unlike sparse representations, dense vectors capture semantic relationships between items, allowing similar items to be positioned close to each other in the vector space. These representations are typically learned using ML or DL models. Dense representations are compact, efficient, and better suited for large-scale recommendation tasks.

Characteristics:

- Low-dimensional;
- Semantically rich;
- Compact;
- Learned representations.

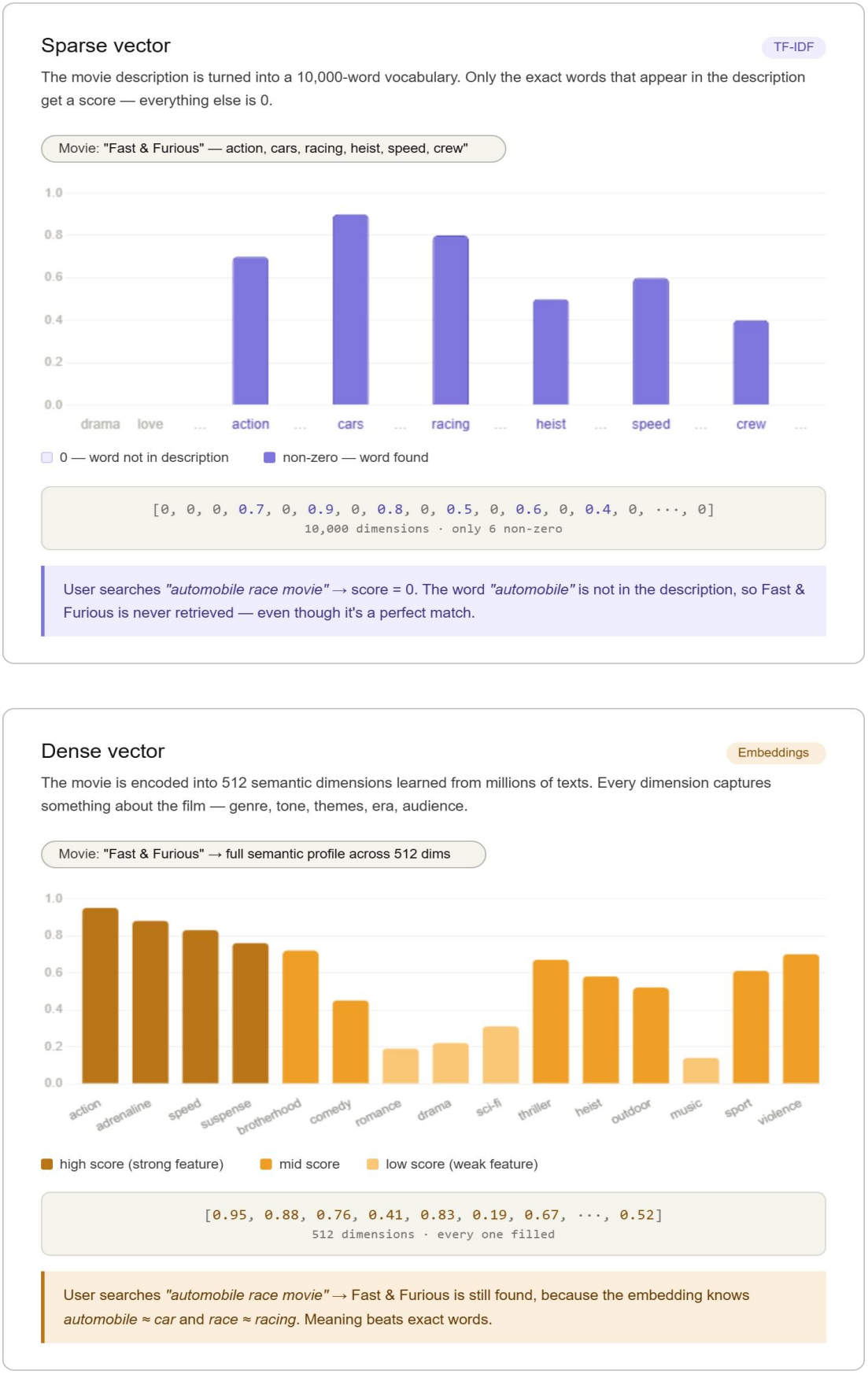

Figure 2: Sparse vs. Dense Vector Representation in Content-Based Movie Retrieval

3.2 Collaborative Filtering

Collaborative filtering (CF) is based on the assumption that users who had similar preferences in the past will likely share similar tastes in the future. Instead of relying on item features, it uses user interactions such as ratings or viewing history.

There are two main approaches:

- **User-based CF**: This method identifies users with similar preferences and recommends items that those users have liked;
- **Item-based CF**: This approach focuses on the similarity between items. Items that receive similar ratings from many users are considered related and can be recommended together.

CF has been widely studied and applied in real-world systems, including platforms like Netflix. it is effective in capturing complex user behavior patterns and generating accurate recommendations [4].

CF has been widely studied and applied in real-world systems, including platforms like Netflix. it is effective in capturing complex user behavior patterns and generating accurate recommendations [4].

3.3 Hybrid Approaches

Hybrid recommender systems combine multiple recommendation techniques in order to overcome the limitations of individual approaches. In most cases, they integrate content-based filtering with collaborative filtering to improve the overall quality of recommendations. By leveraging both user interaction data and item features, hybrid systems are able to provide more accurate and relevant suggestions compared to using a single method alone. This combination helps reduce common issues such as the cold start problem and data sparsity, while also improving the diversity of recommendations [6]. Hybrid systems are particularly effective in real-world applications where both types of data are available and can be exploited together. As a result, they are widely adopted in modern platforms because they offer a balanced solution that enhances both performance and user satisfaction.

4. Systems Machine Learning (ML) in Recommender

Machine learning (ML) plays a central role in modern recommender systems, as it enables systems to learn from data and continuously improve their predictions. Instead of relying on fixed rules, ML models analyze user behavior and interactions to identify patterns and generate personalized recommendations. This makes them highly effective in dynamic environments where user preferences evolve over time [2].

4.1 Traditional ML Approaches

Traditional ML methods such as k-nearest neighbors (KNN), decision trees, and clustering techniques have been widely used in recommender systems. These approaches help identify similarities between users or items and group them based on shared characteristics. For instance, clustering can segment users with similar tastes, while KNN can recommend items based on similarity measures. These techniques provide a solid foundation for recommendation tasks and remain relevant for building simple and interpretable models [7].

4.2 Regression and Classification in Recommendation

Regression and classification are common ML approaches used to model recommendation problems. Regression techniques are applied to predict numerical values such as user ratings, while classification methods are used to predict categorical outcomes, such as whether a user will like or dislike a movie. By transforming recommendation into a predictive problem, these methods allow the use of a wide range of ML algorithms to estimate user preferences and improve recommendation accuracy [7][2].

4.3 Matrix Factorization

Matrix factorization is one of the most effective techniques in recommender systems, particularly in collaborative filtering. It works by decomposing the user-item interaction matrix into lower-dimensional representations that capture latent features of users and items. This allows the system to uncover hidden relationships and make accurate predictions even when the data is sparse. Factorization-based models such as factorization machines extend this idea and are highly effective in handling large-scale recommendation tasks [8].

5 Deep Learning (DL) in Recommender Systems

Deep learning (DL) has significantly improved recommender systems by allowing them to learn complex patterns from large-scale data. Unlike traditional ML methods, DL models can process different types of information such as user interactions, textual data, and sequential behavior, which makes them suitable for modern applications like movie recommendation systems. Their ability to model non-linear relationships between users and items helps improve recommendation accuracy and personalization [2].

5.1 Embeddings (Users and Items)

A fundamental concept in DL-based recommender systems is the use of embeddings, which are dense vector representations of users and items in a continuous feature space. These embeddings capture meaningful characteristics of both users and items, allowing the system to represent similar entities close to each other and compute similarity efficiently. Embedding-based models have become widely adopted in recommendation tasks because they can model complex user-item interactions more effectively than traditional methods [9]. Embeddings are particularly useful in movie recommendation systems, where they represent user preferences and movie features in vector form for similarity comparison and ranking. These representations are compact and enable efficient computation of similarities using standard measures such as cosine similarity. In large-scale systems, embeddings make it possible to handle a high number of users and items while maintaining good performance. In addition, large industrial systems such as YouTube have successfully applied DL and embeddings in recommendation pipelines, demonstrating their effectiveness in handling millions of users and items [10]. DL models can also incorporate sequential information, such as watch history, to better understand changes in user preferences over time and improve recommendation quality in dynamic environments.

6 Evaluation Metrics

Evaluating a recommender system is important to understand how well it performs and whether the recommendations are actually useful to users. Different metrics are used depending on the goal of the system, such as predicting ratings accurately or ranking relevant items at the top. In practice, evaluation focuses mainly on accuracy and ranking quality [2].

6.1 Accuracy Metrics

Accuracy metrics measure how close the predicted values are to the real user preferences. One common approach is to evaluate rating prediction using Root Mean Square Error **(RMSE)** and Mean Absolute Error **(MAE)**. These metrics calculate the difference between predicted ratings and actual ratings, where lower values indicate better performance. They are especially useful when working with explicit feedback, such as movie ratings.

Another way to evaluate accuracy is by using classification-based metrics such as **precision** and **recall**. Precision measures how many of the recommended items are actually relevant, while recall measures how many relevant items are successfully recommended. These metrics are widely used to evaluate the effectiveness of recommendation models in practical systems [7].

6.2 Ranking Metrics

In many real-world scenarios, the order of recommended items is more important than predicting exact ratings. Ranking metrics are therefore used to evaluate how well the system places relevant items at the top of the recommendation list. One widely used metric is Normalized Discounted Cumulative Gain **(NDCG)**, which takes into account both the relevance of items and their position in the list.

Another simple metric is **Hit Rate**, which checks whether at least one relevant item appears in the Top-N recommendations. Ranking-based evaluation provides a more realistic measure of system performance, since users typically focus only on the top results [11].

7 Context of Our Work

7.1 Problem Statement

Users often face difficulties finding movies that match their preferences because of the large number of available choices. Traditional search methods depend mainly on manual exploration and do not provide personalized suggestions adapted to user interests. Furthermore, recommendation approaches based on a single technique may generate limited or less accurate results. For this reason, there is a need for a recommendation system capable of producing more relevant and personalized movie recommendations.

7.2 Proposed Solution

To address these limitations, our work proposes a hybrid movie recommendation system that combines content-based filtering and collaborative filtering. This approach aims to improve recommendation accuracy by exploiting both movie features and user interactions, allowing the system to generate more relevant and personalized recommendations.

7.3 Technical Choices

Several technical choices were adopted in the proposed system to improve recommendation quality and efficiency. Content-based filtering was selected to recommend movies according to their characteristics, while collaborative filtering was used to learn user preferences from ratings and interactions. Embeddings were employed as dense feature representations to capture semantic relationships between movies, and matrix factorization techniques were considered to better model user-item interactions. Cosine similarity was also adopted due to its effectiveness in comparing vector representations and generating recommendations.

8 Challenges and Risks

Recommender systems face several challenges that can affect their performance and reliability in real-world applications. The main challenges include:

- **Cold Start Problem:** Occurs when there is insufficient data about new users or new items. Without prior interactions, the system struggles to generate accurate recommendations, especially in collaborative filtering approaches. This issue is widely recognized as one of the major limitations of recommender systems [12];
- **Data Sparsity:** The user-item interaction matrix is often sparse because users interact with only a small subset of available items. This lack of information makes similarity computation difficult and negatively impacts recommendation quality [13];
- **Scalability:** As recommender systems grow to include millions of users and items, processing large-scale data efficiently becomes a significant challenge. Scalability is particularly important in systems that require real-time recommendations [14];
- **Bias and Fairness:** Recommender systems may favor popular items or reinforce existing user preferences, which can reduce diversity and create filter bubbles. Ensuring fairness and balanced recommendations has become an important research topic in modern recommender systems [15];

- **Privacy Issues:** Since recommender systems rely heavily on user data, concerns related to privacy, data collection, and information security remain critical challenges. Protecting user information while maintaining recommendation quality is essential [16].

9 Conclusion

In this chapter, we presented a comprehensive overview of recommender systems, starting from their fundamental concepts and main approaches, including content-based and collaborative filtering, as well as hybrid methods. We also explored the role of machine learning and deep learning techniques in improving recommendation quality, with a focus on feature representation and embeddings. In addition, we discussed the main evaluation metrics used to assess system performance, such as accuracy and ranking measures, and highlighted the key challenges and risks that affect recommender systems in real-world applications.

In the following chapter, we will focus on the design and implementation of the proposed recommendation system, detailing the architecture, the chosen methods, and the techniques used to build an effective and scalable solution

.

Chapter 2 System design

1. Introduction

Modern streaming platforms contain massive collections of movies, making it increasingly difficult for users to discover content that fits their preferences. Recommender systems address this challenge by analyzing available data and automatically suggesting relevant movies to each user.

The system proposed in this work aims to provide personalized movie recommendations through a hybrid recommendation approach that combines collaborative filtering and content-based filtering. Instead of relying on a single technique, the system exploits both user interactions and movie metadata to produce more accurate and reliable recommendations.

This chapter presents the overall design of the proposed system, including its architecture, data modeling, recommendation methods, and evaluation process.

2. General System Architecture

The proposed recommendation system is designed using a modular architecture composed of several interconnected components. The objective of this architecture is to ensure efficient communication between the user interfaces, the recommendation engine, and the data storage layer.

The system consists of four main components:

- **Frontend Layer:** Provides the user interface through both web and mobile applications, allowing users to browse movies, view details, submit ratings, and receive personalized recommendations;
- **Backend Layer:** Acts as the intermediary between the frontend, the database, and the recommendation engine. It manages user requests, processes application logic, and returns recommendation results to the user interfaces;
- **Database Layer:** Stores user information, movie metadata, and rating data used by the recommendation process. The stored ratings represent the main source of user interactions exploited by the system;

- **Recommendation Engine:** Responsible for generating personalized movie recommendations using a hybrid approach that combines collaborative filtering and content-based filtering techniques.

The system workflow starts when a user submits a movie rating through the application. The rating is stored in the database and forwarded to the recommendation engine, where it is processed and integrated into the recommendation models. The generated recommendations are then returned to the backend and displayed to the user through the web or mobile interface.

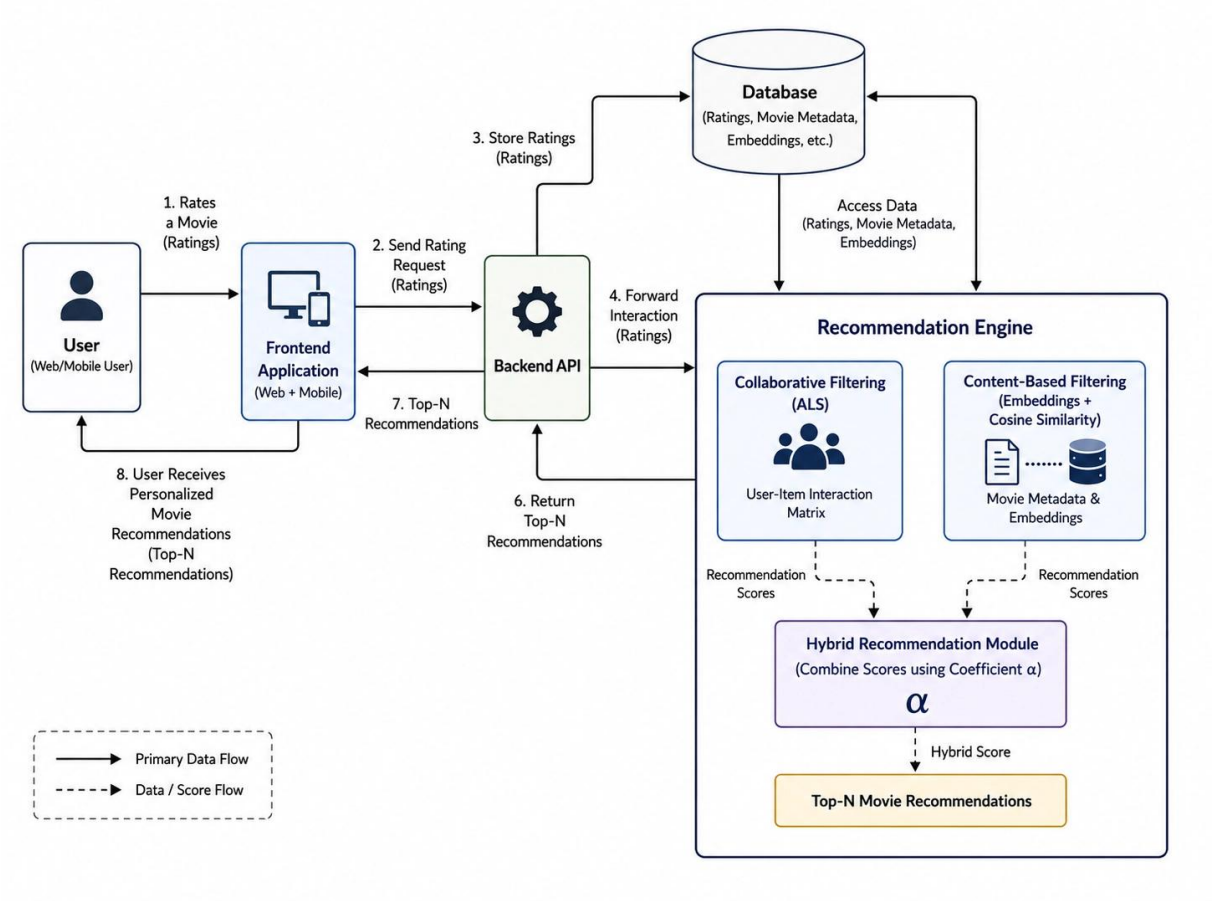

Figure 3:Data Flow Diagram of the Hybrid Movie Recommendation System

2.1 Recommendation Workflow

Figure 3 illustrates the overall workflow of the proposed hybrid recommendation system. The recommendation process starts when the user interacts with the application by submitting a movie rating. The interaction then passes through several system components before personalized recommendations are generated and returned to the user.

The workflow can be summarized as follows:

1. The user submits a movie rating through the web or mobile application;
2. The frontend application sends the rating request to the backend API;
3. The backend stores the rating and related information in the database;
4. The interaction data is forwarded to the recommendation engine for processing;
5. The recommendation engine accesses rating data, movie metadata, and embeddings from the database. The collaborative filtering module processes the user-item interaction matrix using ALS, while the content-based module computes movie similarity using embeddings and cosine similarity;
6. The hybrid recommendation module combines the recommendation scores generated by both approaches using the coefficient α in order to produce final recommendation scores;
7. The generated Top-N movie recommendations are returned to the backend API and transmitted to the frontend application;
8. The user receives personalized movie recommendations through the application interface.

3 Hybrid Recommendation Approach

The proposed system adopts a hybrid recommendation approach that combines collaborative filtering and content-based filtering within a unified recommendation framework. The objective of this combination is to exploit both user interaction patterns and movie descriptive information in order to generate more accurate and personalized recommendations. Hybrid recommendation models are widely adopted because they reduce the limitations associated with single-method recommendation techniques [17].

Using a single recommendation technique often leads to several limitations. Collaborative filtering may suffer from sparse interaction data and cold-start situations, while content-based filtering can produce limited recommendation diversity. By integrating both approaches, the system benefits from their complementary strengths and achieves a more robust recommendation process [18].

The hybrid architecture is composed of three main components:

- a collaborative filtering module based on ALS matrix factorization;
- a content-based filtering module based on embeddings and cosine similarity;
- a hybrid combination module responsible for merging recommendation scores using a weighting coefficient α.

This architecture allows the system to capture hidden user preferences while simultaneously exploiting semantic relationships between movies.

3.1 Collaborative Filtering (ALS)

The collaborative filtering component learns recommendation patterns directly from user interactions. Instead of analyzing movie attributes, the model focuses on behavioral similarities extracted from rating data. The underlying assumption is that users who exhibit similar interaction behaviors are likely to appreciate similar movies [19].

To model these relationships, the proposed system uses the **Alternating Least Squares (ALS)** algorithm based on matrix factorization. ALS is particularly effective for large sparse datasets due to its scalability, optimization efficiency, and ability to learn latent relationships between users and movies [20].

3.1.1 Matrix Factorization

The recommendation process starts by constructing a sparse user-item interaction matrix where rows represent users and columns represent movies. Since each user interacts with only a small subset of available movies, most matrix entries remain empty, producing a highly sparse structure.

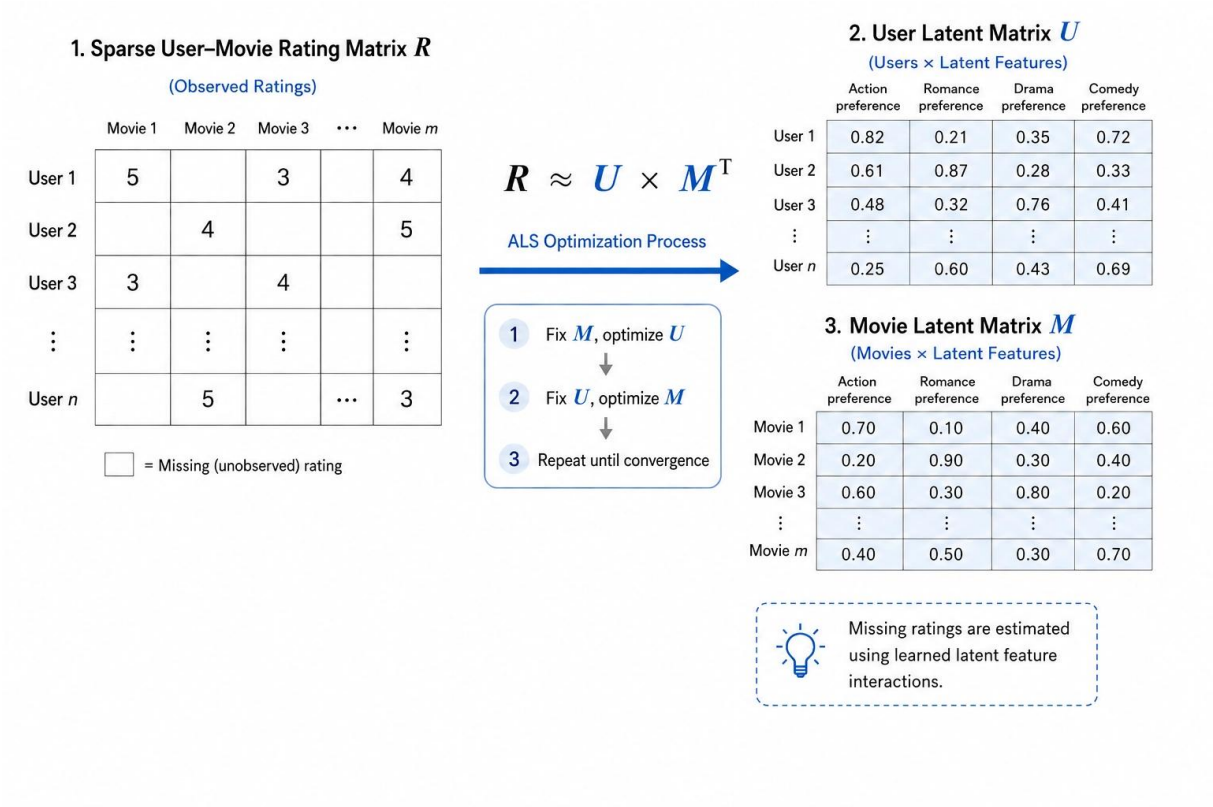

Figure 4:Matrix Factorization using Alternating Least Squares (ALS) for Movie Recommendation

To address this sparsity problem, matrix factorization decomposes the interaction matrix into two lower-dimensional latent matrices:

$$\boldsymbol{R = U \times M^T}$$

Where:

- R represents the user-item interaction matrix;
- U represents the user latent matrix;
- M represents the movie latent matrix.

The latent matrices contain hidden feature representations learned during training. Instead of explicitly describing genres or movie categories, these vectors capture implicit behavioral characteristics extracted from interaction patterns. Users with similar preferences tend to obtain similar latent vectors, while movies appreciated by similar users become close in the latent feature space [21].

The ALS optimization process alternates between optimizing user latent vectors while keeping movie vectors fixed, and optimizing movie vectors while keeping user vectors fixed. This iterative process continues until convergence is reached and prediction error is minimized.

The implemented ALS model uses:

- 128 latent factors;
- 20 training iterations;
- regularization techniques to reduce overfitting.

Once the latent representations are learned, unknown interactions can be estimated using:

$$\hat{r}_{ui} = U_u . M_i^T$$

Where:

- $\hat{r}_{ui}$ represents the predicted interaction score between user u and movie **i**;
- U_u represents the latent vector of the user;
- M_i^T represents the latent vector of the movie.

This mechanism enables the model to estimate unseen preferences and generate personalized recommendations from learned interaction patterns.

3.1.2 ALS Optimization Principle

ALS optimizes the latent matrices iteratively by alternating between user-factor optimization and movie-factor optimization. During each iteration:

- user vectors are updated while movie vectors remain fixed;
- movie vectors are then updated while user vectors remain fixed.

This alternating optimization strategy simplifies the matrix factorization problem into smaller optimization tasks that can be solved efficiently using least squares methods. As a result, ALS remains computationally efficient even when handling millions of interactions and highly sparse datasets [22].

3.1.3 User-Item Interaction Matrix

The collaborative model relies on implicit feedback generated from user ratings. Although the original dataset contains ratings ranging from 1 to 5 stars, the implemented ALS model operates using binary interaction signals [23].

During preprocessing:

- ratings greater than or equal to 4 are converted to 1, representing positive interactions;
- ratings lower than 4 are ignored.

This transformation allows the model to focus on movies genuinely appreciated by users while reducing noisy or ambiguous interactions.

The resulting interaction matrix is represented as follows:

Table 1: User–Movie Interaction Matrix

User/Movies	Movie1	Movie2	Movie3
User1	1	0	1
User2	0	1	1
User3	1	1	0

Where:

- 1 indicates a positive interaction;
- 0 indicates the absence of interaction.

This sparse matrix constitutes the primary input of the ALS recommendation model.

3.2 Content-Based Filtering

The content-based filtering component generates recommendations by analyzing movie characteristics and semantic similarities between items. Unlike collaborative filtering, which relies on user interaction behavior, this approach focuses on the descriptive attributes of movies in order to identify items sharing similar content.

The recommendation process is based on the assumption that users tend to appreciate movies with related themes, genres, keywords, or semantic descriptions. Consequently, if a user positively interacts with a movie, the system recommends other movies whose content representations are semantically close.

3.2.1 Feature-Based Representation

Each movie is represented using a set of descriptive attributes extracted from the dataset and

preprocessing pipeline. These attributes include:

- genres;
- movie tags and keywords;
- title information;
- metadata descriptions.

During preprocessing, these features are merged into a unified textual representation called content. This representation summarizes the semantic characteristics of each movie and serves as the main input for the embedding generation process.

Instead of relying on traditional sparse textual representations such as Bag-of-Words or TF-IDF alone, the proposed system uses dense semantic embeddings generated through transformer-based language models. This approach enables the system to capture contextual and semantic relationships between movies more effectively [24].

3.2.2 Sentence Transformer Embeddings

To generate semantic movie representations, the system uses the **SentenceTransformer** model based on the **all-MiniLM-L6-v2** architecture. This transformer model converts movie textual content into dense vector embeddings of 384 dimensions.

The embedding generation process starts by tokenizing the textual movie representation before passing it through multiple transformer encoder layers. Self-attention mechanisms are then used to learn contextual relationships between words and semantic dependencies inside the movie description. Finally, mean pooling is applied to generate a fixed-length sentence embedding representing the entire movie content.

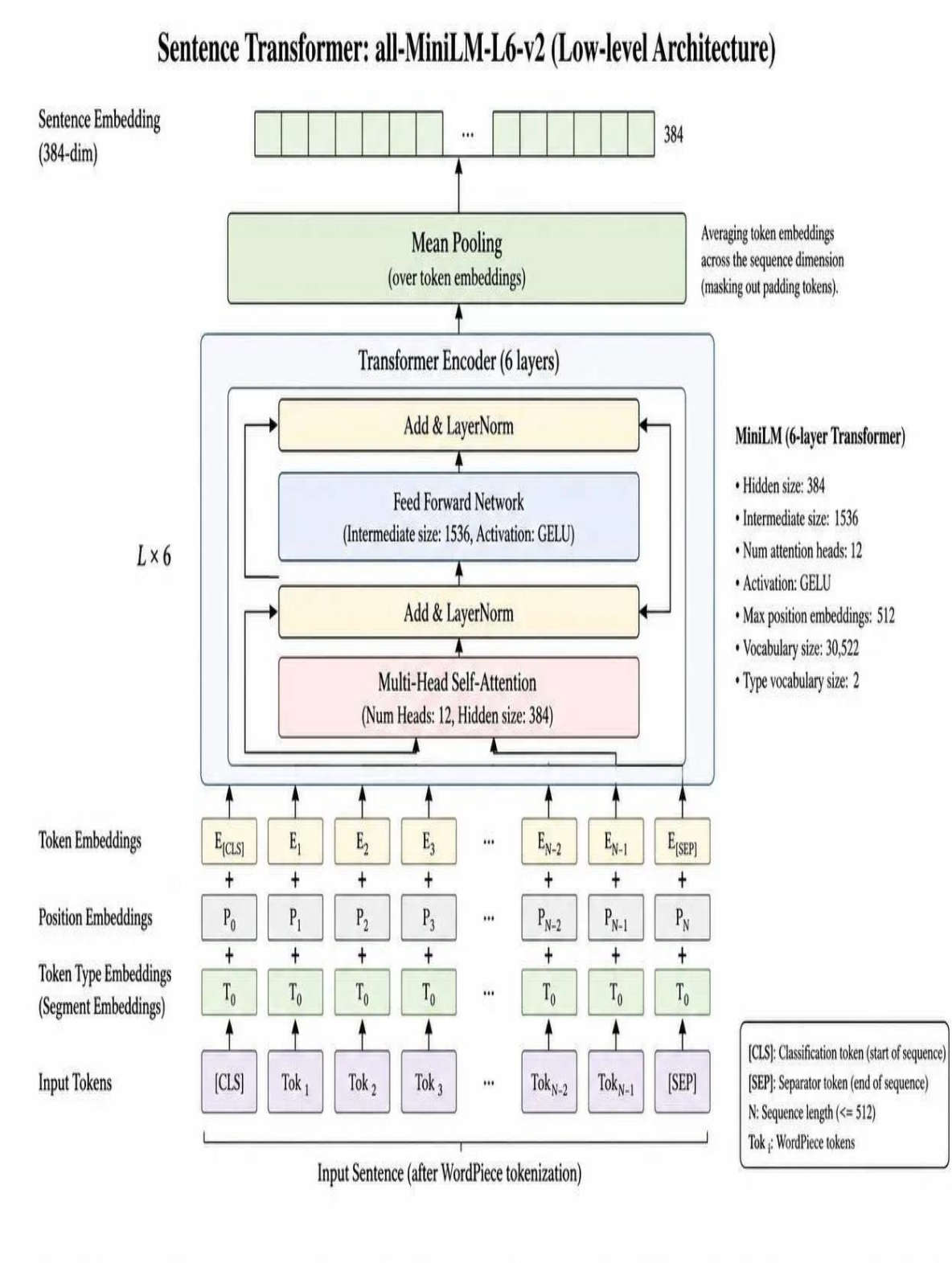

Figure 5: Low-Level Architecture of the all-MiniLM-L6-v2 Sentence Transformer Model

As illustrated in **Figure 5**, the transformer architecture combines:

- token embeddings;
- positional embeddings;
- self-attention layers;
- feed-forward neural networks;
- mean pooling operations.

This architecture allows the model to learn semantic relationships between movies beyond simple keyword matching, improving the quality of similarity computation and recommendation generation [25].

The generated embeddings place semantically similar movies close to each other in the vector space. For example, movies sharing similar genres, themes, or narrative structures tend to produce nearby embedding representations even if they do not contain identical keywords.

3.2.3 Cosine Similarity

Once movie embeddings are generated, similarity between movies is computed using **cosine similarity**.

The cosine similarity between two movie vectors A and B is defined as:

$$Similarity(A, B) = \frac{A.B}{||A||||B||}$$

This metric measures the angular similarity between vectors rather than their magnitude, making it particularly suitable for high-dimensional embedding spaces.

A similarity score close to:

- 1 indicates highly similar movies;
- 0 indicates weak similarity;
- -1 indicates opposite semantic orientation.

Using cosine similarity allows the system to efficiently retrieve movies whose semantic representations are close to the target movie embedding.

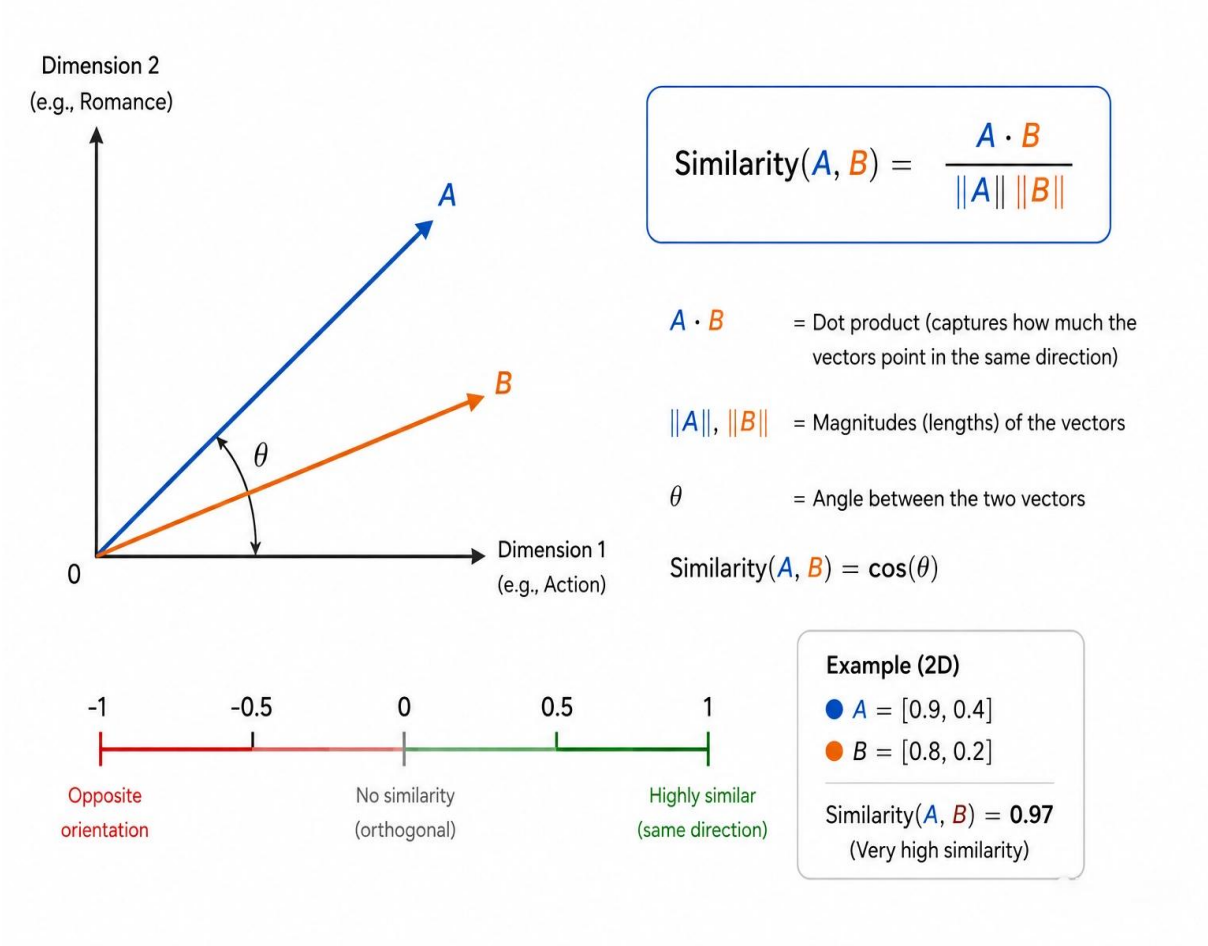

Figure 6:Illustration of Cosine Similarity Between Movie Embeddings

When a user interacts positively with a movie, the system retrieves the corresponding embedding representation and computes similarity scores with all other movie embeddings stored in the recommendation engine.

Movies obtaining the highest similarity scores are selected as candidate recommendations. The resulting recommendations therefore preserve semantic consistency with the movies previously appreciated by the user.

Compared with traditional keyword-based approaches, transformer embeddings provide richer semantic understanding and improve recommendation relevance by capturing contextual relationships between movie descriptions [26].

3.3 Hybrid Combination (Coefficient α)

The final recommendation stage relies on a weighted hybrid strategy that combines the outputs generated by the collaborative filtering and content-based filtering modules. Instead of depending exclusively on a single recommendation source, the system integrates both

recommendation scores in order to exploit their complementary strengths and improve recommendation quality.

The hybrid recommendation score is computed using the following weighted combination:

$$HybridScore = \alpha \times CollaborativeScore + (1 - \alpha) \times Content_BasedScore$$

Where:

- α represents the weighting coefficient;
- the collaborative score corresponds to the prediction generated by the ALS model;
- the content-based score corresponds to the similarity score computed using embeddings and cosine similarity.

This formulation allows the recommendation engine to balance behavioral information and semantic similarity during recommendation generation.

3.3.1 Role of the Coefficient α

The coefficient α determines the contribution of each recommendation component in the final prediction process.

Different values of α influence recommendation behavior differently:

- α→1: recommendations rely mainly on collaborative filtering;
- α→0: recommendations rely mainly on content-based filtering;
- intermediate values create a balanced recommendation strategy.

The weighting mechanism therefore plays an important role in maintaining recommendation stability across different interaction scenarios. Collaborative filtering contributes stronger personalization through learned behavioral patterns, while content-based filtering strengthens recommendation reliability when interaction data is limited.

By combining both approaches, the system benefits simultaneously from:

- latent preference learning from ALS;
- semantic movie similarity from embeddings.

3.3.2 Hybrid Recommendation Workflow

The recommendation generation process is performed in three main stages:

1. The collaborative filtering module predicts user preference scores using ALS latent representations;
2. The content-based module computes semantic similarity scores between movie embeddings using cosine similarity;
3. The generated scores are normalized and combined using the coefficient α.

The resulting hybrid score is then used to rank movies and produce the final personalized Top-N recommendations.

3.3.3 Benefits of Hybridization

The integration of collaborative and content-based filtering improves recommendation quality by reducing the weaknesses associated with each individual technique [27].

Collaborative filtering is effective for discovering hidden relationships between users and movies, but its performance may decrease in sparse interaction environments. In contrast, content-based filtering remains operational even when interaction history is limited because it relies on movie metadata and semantic representations instead of collective user behavior.

The hybrid combination therefore improves:

- recommendation robustness;
- personalization accuracy;
- recommendation diversity;
- resistance to sparsity and cold-start situations.

By integrating both behavioral and semantic information within a unified recommendation framework, the proposed system generates recommendations that are more adaptive and reliable than those produced using a single recommendation method [28].

4 Cold Start Problem Management

One of the main challenges in recommendation systems is the **cold start problem**, which appears when the system does not possess sufficient interaction data to generate reliable personalized recommendations. This situation mainly concerns new users who have not yet rated enough movies, as well as newly added items with limited interaction history. Since collaborative filtering depends heavily on user interactions, the absence of ratings reduces the model’s ability to learn user preferences accurately [29].

4.1 Cold Start Strategy

To address this limitation, the proposed system integrates a fallback recommendation strategy based on popular and trending movies. When a user has not yet provided enough ratings, the recommendation engine temporarily relies on globally appreciated movies instead of personalized predictions. The recommendation process therefore remains operational even in the absence of sufficient behavioral data.

The popularity-based recommendations are generated using several indicators, including:

- average movie ratings;
- rating frequency;
- overall interaction activity.

Movies with high popularity scores are prioritized during the early usage stage. This approach is widely adopted by modern streaming platforms because it provides stable and generally relevant recommendations for users with limited interaction history [30].

4.2 Transition Toward Personalization

As users progressively rate movies, the system accumulates interaction data and updates the recommendation models accordingly. Once sufficient behavioral information becomes available, the recommendation engine gradually transitions from popularity-based suggestions toward fully personalized hybrid recommendations generated through collaborative and content-based filtering.

This progressive transition improves recommendation continuity, enhances user experience during early interactions, and ensures smoother personalization over time. By combining popularity-based initialization with adaptive hybrid recommendation, the system maintains recommendation availability while progressively refining recommendation accuracy according to user preferences.

6 Dynamic Model Update

The proposed recommendation system is designed to continuously adapt to user preferences through dynamic model updating. Instead of generating static recommendations, the system progressively refines recommendation quality whenever new user interactions become available. This adaptive mechanism enables the recommendation engine to maintain personalization relevance over time and reflect the evolution of user interests.

The update workflow starts when a user submits a new movie rating through the application interface. The interaction is immediately transmitted to the backend API and stored in the database as part of the user interaction history. Once the new rating is recorded, the recommendation engine integrates the updated interaction data into the recommendation pipeline.

The collaborative filtering component then retrains the ALS model using the updated user-item interaction matrix. During this process, latent user and movie representations are recalculated in order to incorporate the newly observed preferences. As a result, recommendation scores evolve dynamically according to recent user behavior.

After retraining, the recommendation engine regenerates recommendation predictions and updates the personalized Top-N recommendation list returned to the user. This continuous update cycle allows the system to react to preference changes and improve recommendation relevance progressively over time.

The dynamic update mechanism provides several important advantages:

- continuous adaptation to evolving user preferences;
- improved personalization accuracy;
- integration of recent interactions into recommendation generation;
- increased recommendation responsiveness.

Unlike static recommendation approaches where recommendation outputs remain unchanged for extended periods, the proposed system maintains an adaptive recommendation environment capable of learning continuously from user activity. This adaptive behavior is particularly important in movie recommendation systems where user interests may evolve according to viewing habits and newly discovered preferences [31].

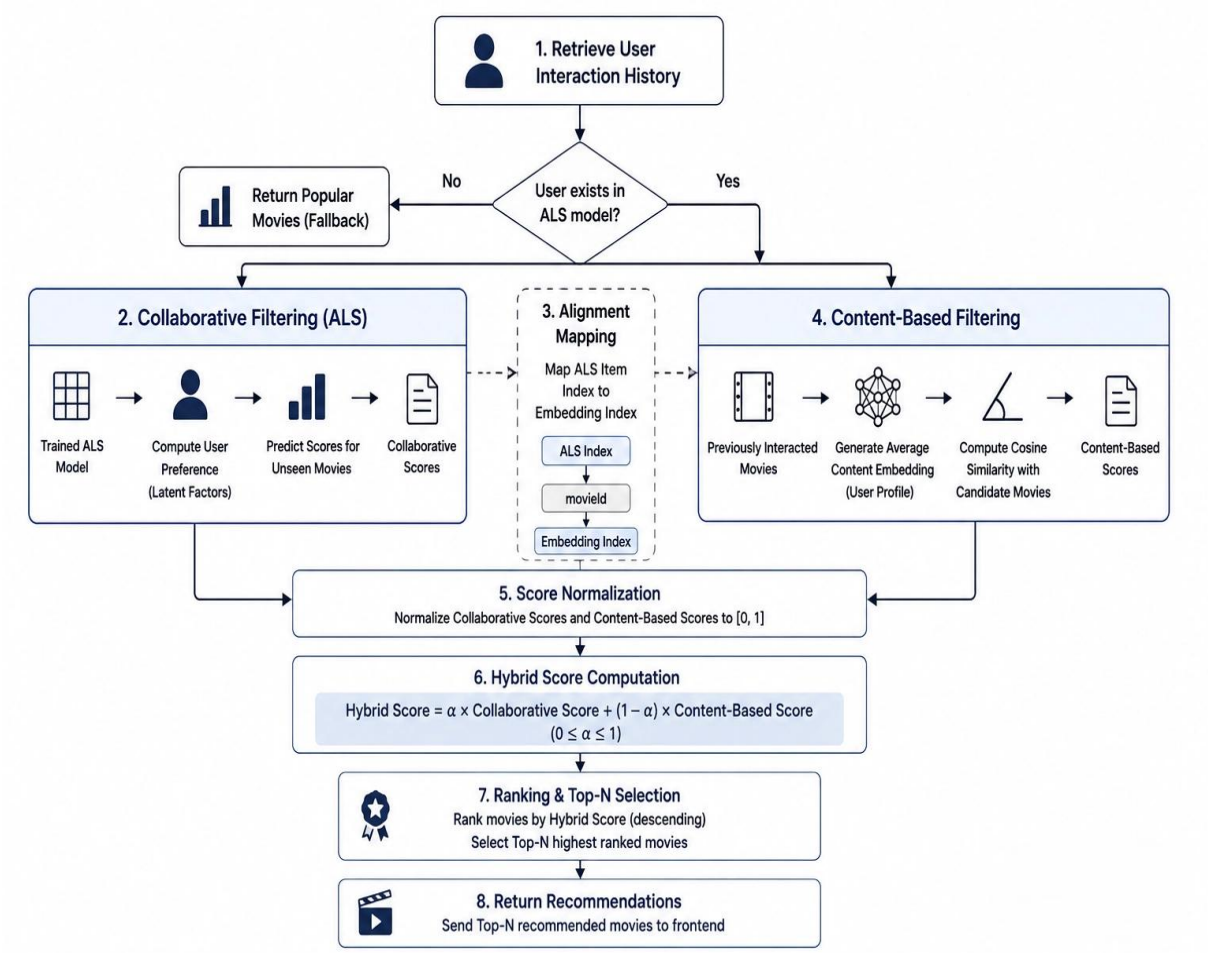

Figure 7:Workflow of the Hybrid Recommendation Generation Process

7 Design Diagrams

7.1 UML Definition

Unified Modeling Language (UML) is a standardized graphical modeling language used to visualize, design, and document software systems. UML provides a set of diagrams that describe both the structural and behavioral aspects of a system, allowing developers and designers to represent system architecture, interactions, and workflows in a clear and organized manner [32].

7.2 Use Case Diagram

A use case diagram is a behavioral UML diagram that represents the interactions between users and the system functionalities. It provides a high-level view of the services offered by the application and identifies the main actions performed by users within the system [33].

In the proposed movie recommendation system, the use case diagram illustrates the primary

functionalities available to users, including:

- user authentication;
- movie browsing and searching;
- movie rating;
- recommendation generation;
- profile interaction.

The diagram also highlights the relationship between the user and the recommendation engine through the rating and recommendation processes.

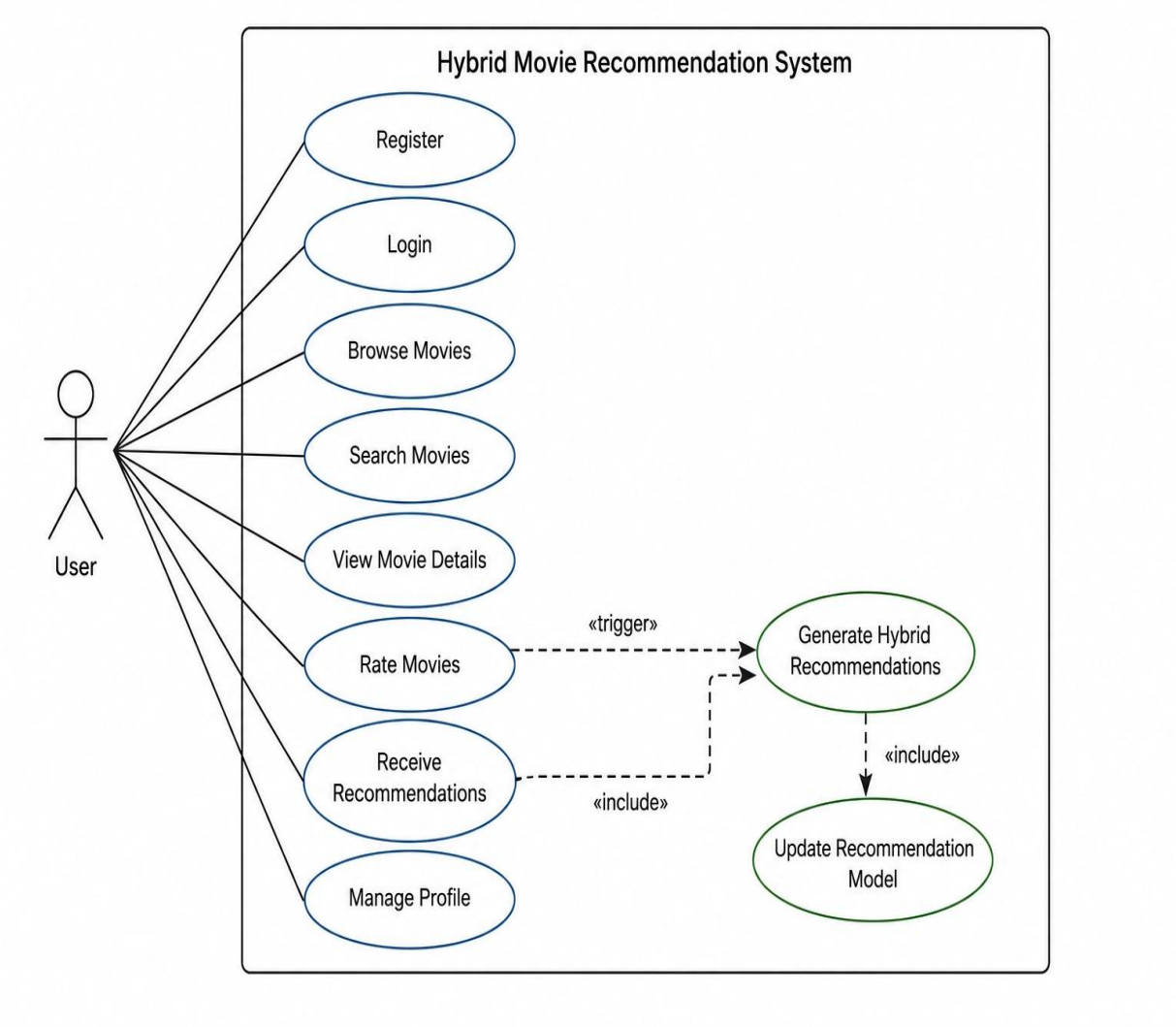

Figure 8: Use Case Diagram for the Hybrid Movie Recommendation System

7.3 Sequence Diagram

A sequence diagram is a dynamic UML diagram used to represent the chronological exchange of messages between system components during a specific process. It emphasizes interaction order and illustrates how different system entities collaborate to accomplish a given operation [34].

In the proposed system, the sequence diagram models the recommendation workflow triggered after a user submits a movie rating. The diagram illustrates:

- communication between the frontend application and backend API;
- database interaction;
- recommendation engine processing;
- ALS model update;
- recommendation generation and delivery.

This representation provides a detailed visualization of the dynamic behavior of the hybrid recommendation system and clarifies the interaction flow between architectural components.

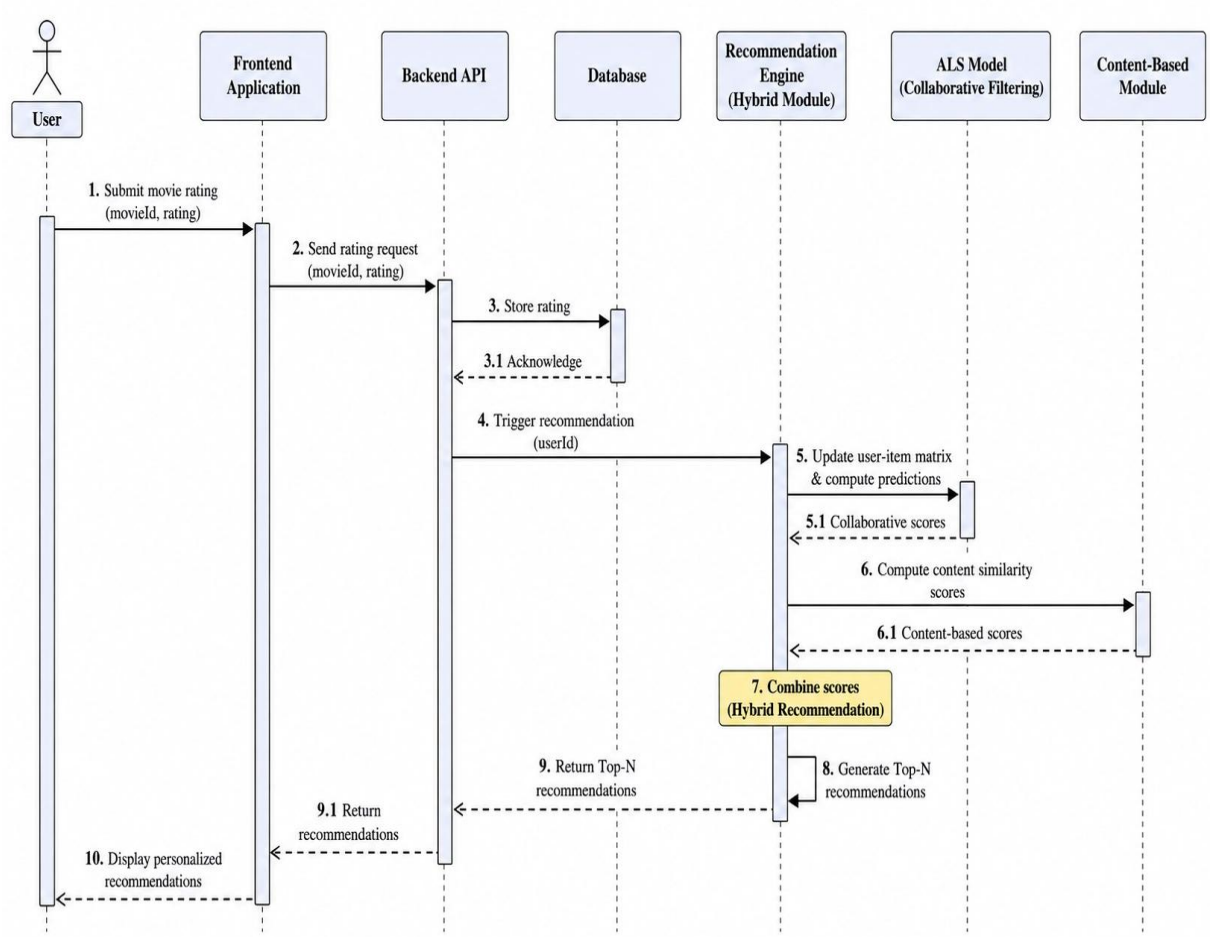

Figure 9: Sequence Diagram of the Hybrid Recommendation Generation Process

8 Conclusion

This chapter presented the design of the proposed hybrid movie recommendation system and the main architectural choices adopted during system conception. The proposed architecture combines collaborative filtering based on ALS matrix factorization with content-based filtering using semantic embeddings and cosine similarity in order to improve recommendation relevance and personalization quality. The chapter also introduced the dynamic update mechanism, cold start management strategy, and UML design diagrams used to model system behavior and recommendation workflows.

These design choices provide a scalable and adaptive recommendation framework capable of integrating both behavioral and semantic information within a unified recommendation process. The next chapter focuses on the implementation phase, where the different system components, technologies, and experimental configurations are presented in

Chapter 3 System Implementation and Results

1 Introduction

This chapter presents the implementation of the proposed hybrid movie recommendation system and the experimental results obtained during system evaluation. Following the design phase presented previously, this chapter focuses on the practical realization of the system components, including the web and mobile applications, backend services, recommendation engine, and database integration. The chapter first introduces the technologies, frameworks, and development tools used during development. It then describes the dataset characteristics and preprocessing operations performed before training the recommendation models. In addition, the implementation of the hybrid recommendation engine and the communication workflow between the different system components are presented.

Finally, the chapter discusses the evaluation methodology and analyzes the experimental results obtained from the proposed recommendation approach in order to assess recommendation quality and overall system performance.

2 Used Technologies, Frameworks and Tools

This section presents the main technologies, frameworks, and development tools used during the implementation of the proposed hybrid movie recommendation system. Different technologies were integrated to support frontend development, backend services, recommendation generation, database communication, and system deployment.

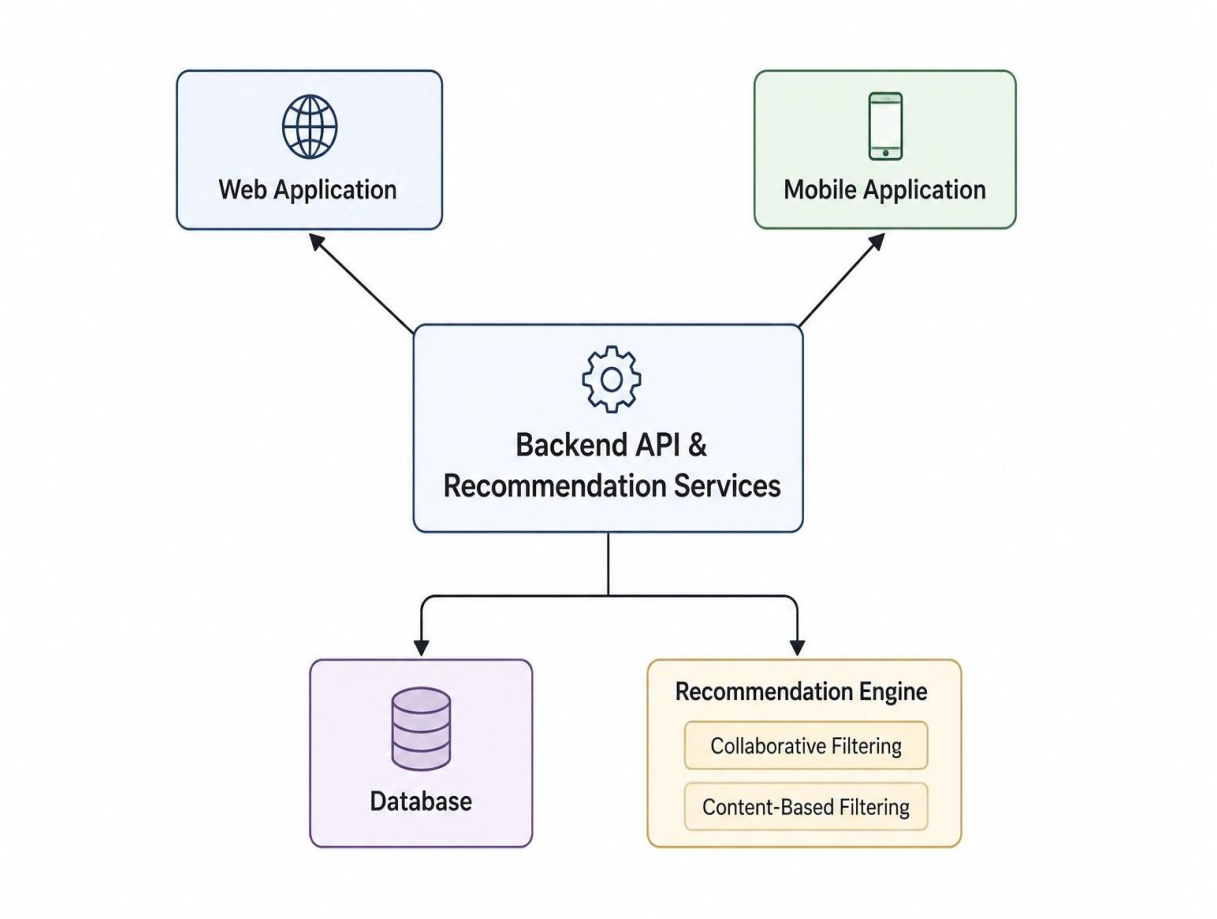

Figure 10:Backend Communication Architecture for Web and Mobile Clients

2.1 Development Environment

The development and experimentation processes were conducted using different hardware environments for the web and mobile applications.

For the web application development, the following hardware configuration was used:

- Processor: 12th Gen Intel(R) Core(TM) i5-12400F (2.50 GHz);
- RAM: 16 GB;
- GPU: NVIDIA GeForce RTX 2060 SUPER.

For the mobile application development, the following configuration was used:

- Processor: 11th Generation Intel® Core™ i5-1135G7;
- RAM: 16 GB;
- GPU: Intel(R) Iris(R) Xe Graphics.

The mobile application testing phase was performed using a physical Android device with the following specifications:

- Device: Redmi Note 8 Pro;
- Screen Size: 16.59 cm;
- Operating System: Android 11.

The implementation process was mainly performed using Visual Studio Code (VS Code) as the integrated development environment (IDE). VS Code is a lightweight and extensible source-code editor widely used for web, mobile, and backend development due to its flexibility, debugging capabilities, and large extension ecosystem [35].

For version control and collaborative development management, the project used Git and GitHub. Git provides distributed version control functionalities allowing efficient source-code tracking and modification management, while GitHub offers cloud-based repository hosting and collaborative development features [36].

2.2 Frontend Technologies

2.2.1 Web Application Technologies

- **React & Vite:** React is a JavaScript library used for building interactive Single Page Applications (SPA) through reusable user interface components. Vite is a modern frontend build tool that provides fast development startup and optimized production builds, improving application performance and development efficiency [37];
- **Tailwind CSS:** Tailwind CSS is a utility-first CSS framework used for designing responsive and customizable user interfaces. It simplifies frontend styling by providing predefined utility classes that accelerate interface development while maintaining consistent design structures [38];
- **React Router:** React Router is a client-side routing library used to manage navigation between application pages without requiring full page reloads. It enables smooth transitions and dynamic URL-based navigation inside the web application [39];
- **Axios:** Axios is a promise-based HTTP client used to establish communication between the frontend application and backend API services. It supports asynchronous HTTP requests and efficient handling of API responses [40];

- **Framer Motion:** Framer Motion is a React animation library used to create smooth page transitions and UI micro-interactions. It improves user experience by providing fluid visual animations and interactive effects [41].

2.2.2 Mobile Application Technologies

- **Flutter SDK (v3.41.2):** Flutter is an open-source UI framework developed by Google for building cross-platform mobile applications using a single codebase. It enables the development of high-performance and responsive mobile interfaces for Android and iOS platforms [42];
- **Dart (v3.11.0):** Dart is the programming language used by Flutter for implementing application logic and reactive user interfaces. It is optimized for fast execution and modern mobile application development [43];
- **Riverpod:** Riverpod is a reactive state-management library used to simplify dependency management and state handling inside Flutter applications. It improves scalability and maintainability of mobile application architecture [44];
- **Dio:** Dio is an HTTP networking library for Dart used to perform asynchronous API requests and manage communication between the mobile application and backend services [45].

2.3 Backend Technologies

- **Node.js v22.18 & Express.js:** Node.js is a JavaScript runtime environment used for executing backend services outside the browser environment. Combined with Express.js, it was used to implement the API Gateway responsible for handling HTTP requests, routing operations, authentication processes, and communication between the frontend applications and the recommendation engine [46];
- **MongoDB & Mongoose:** MongoDB is a NoSQL document-oriented database used for storing user information, movie metadata, and rating interactions. Mongoose was integrated as an Object Data Modeling (ODM) library to simplify schema definition and database manipulation within the Node.js environment [47];
- **JWT (JSON Web Tokens) & bcryptjs:** JWT was used for implementing secure user authentication and session management through token-based authorization mechanisms. bcryptjs was used for password hashing and encryption in order to improve user credential security [48].

2.4 Recommendation Engine Technologies

The recommendation engine was implemented using **Python v3.13** due to its extensive ecosystem of machine learning and data processing libraries.

- **FastAPI & Uvicorn:** FastAPI is a modern asynchronous Python framework used to expose recommendation services through REST APIs. Uvicorn was used as the ASGI server responsible for running the FastAPI application efficiently [49];
- **implicit:** The implicit library provides implementations of collaborative filtering algorithms designed for implicit feedback datasets. In this project, it was used to implement the ALS (Alternating Least Squares) matrix factorization model for collaborative recommendation generation [50];
- **sentence-transformers:** The sentence-transformers library was used to generate semantic embeddings from movie metadata using the all-MiniLM-L6-v2 transformer model. These embeddings allow the system to compute semantic similarity between movies for content-based recommendation [51];
- **Pandas & NumPy:** Pandas and NumPy were used for data loading, preprocessing, matrix manipulation, and numerical computations during recommendation model preparation and training [52];
- **scikit-learn:** scikit-learn was used for evaluation operations and cosine similarity computation during the content-based recommendation process [53];
- **APScheduler:** APScheduler was integrated to automate background retraining tasks and periodic recommendation model updates [54];
- **PyMongo:** PyMongo was used as the Python driver responsible for retrieving user interactions and recommendation-related data directly from MongoDB [55].

3 Dataset and Data Preprocessing

The recommendation system was trained and evaluated using a large-scale movie recommendation dataset containing user rating activities and movie metadata collected over several years. The dataset contains:

- 20,000,263 ratings;
- 465,564 tag applications;
- 27,278 movies;
- 138,493 users.

The collected interactions span from January 1995 to March 2015. All selected users rated at least 20 movies, which improves the reliability of user preference extraction and collaborative learning processes.

The dataset includes several types of information exploited during recommendation generation:

- user identifiers;
- movie identifiers;
- rating values;
- timestamps;
- movie genres;
- movie titles;
- free-text tags and metadata.

These elements provide both behavioral information for collaborative filtering and semantic information for content-based recommendation generation.

The large number of users and interactions makes the dataset highly sparse, since each user interacts with only a small subset of available movies. This sparsity characteristic represents a common challenge in recommendation systems and motivates the use of matrix factorization techniques such as ALS.

3.2 Data Preprocessing

Before training the recommendation models, several preprocessing operations were performed in order to prepare the dataset for collaborative and content-based recommendation tasks.

The preprocessing phase started with data cleaning and metadata organization. Irrelevant attributes and incomplete entries were removed in order to improve data consistency and reduce noise during model training.

Movie metadata originating from different dataset files was then merged into a unified representation. The generated movie content combines multiple descriptive attributes including:

- genres;

- movie titles;
- tags;
- additional textual metadata.

This unified content representation is later used by the content-based filtering module to generate semantic embeddings and compute movie similarity scores.

For collaborative filtering, the original rating values ranging from 1 to 5 were transformed into implicit interaction signals:

- ratings greater than or equal to 4 were converted to positive interactions (1);
- ratings lower than 4 were ignored.

This transformation allows the ALS model to focus on movies genuinely appreciated by users while reducing weak or noisy interactions.

After preprocessing, the interactions were converted into a sparse user-item matrix where:

- rows represent users;
- columns represent movies;
- matrix values represent implicit interactions.

This sparse matrix constitutes the primary input of the collaborative filtering model.

In parallel, the processed movie content was transformed into semantic vector representations using embedding techniques in order to support the content-based recommendation process.

4 System Implementation

4.1 Frontend Implementation

4.1.1 Web Application Interfaces

The web application was implemented as a responsive Single Page Application (SPA) designed to provide smooth navigation and interactive user experience. The interface allows users to browse movies, search for specific titles, view movie details, submit ratings, and receive personalized recommendations generated by the hybrid recommendation engine.

The implemented web interfaces include:

- authentication pages for user registration and login;
- home page displaying movie collections and recommendations;
- search interface for movie retrieval;
- movie details pages containing metadata and rating options;
- personalized recommendation pages generated dynamically according to user preferences.

The communication between the web frontend and backend services is performed through REST API requests, allowing real-time retrieval of recommendation results and user-related information.

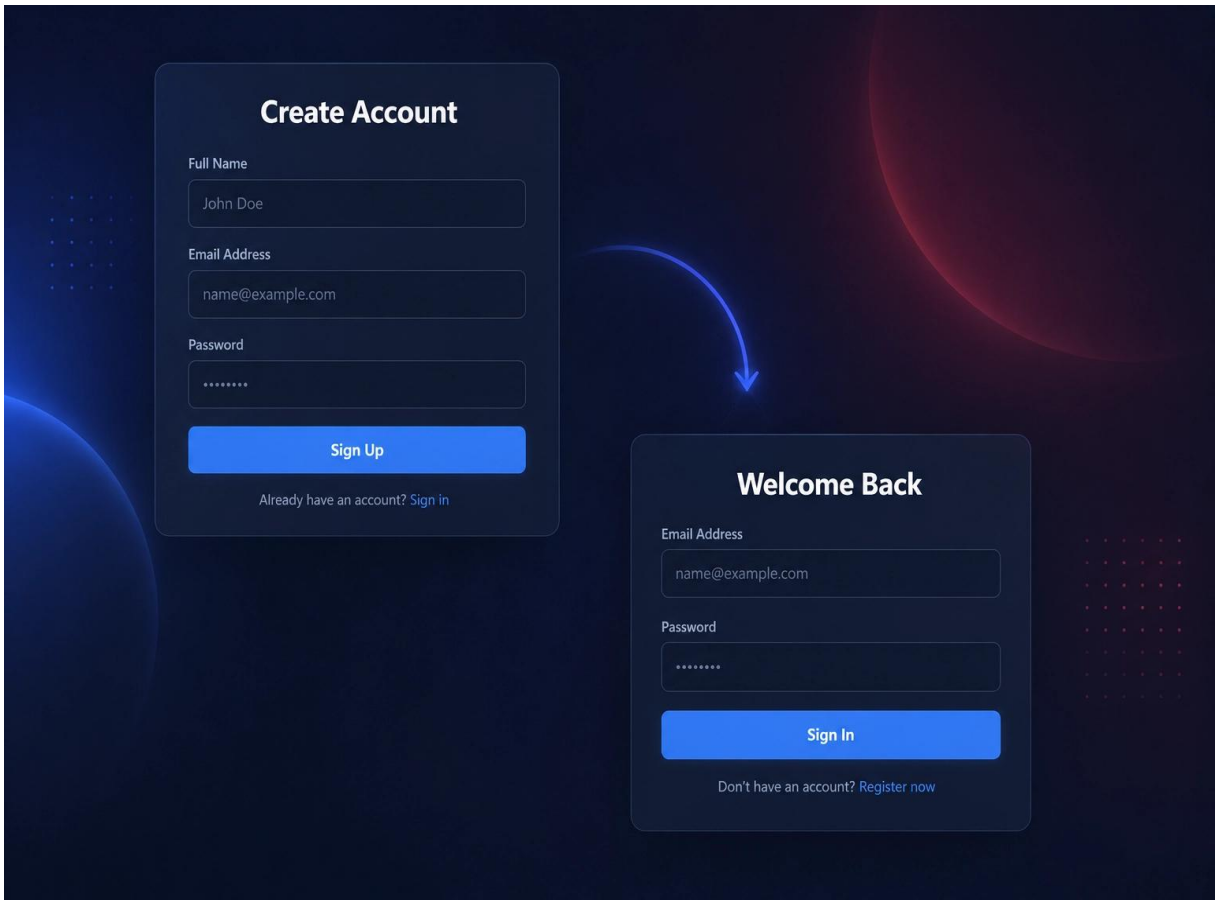

Figure 11:Registration and Log-in Pages

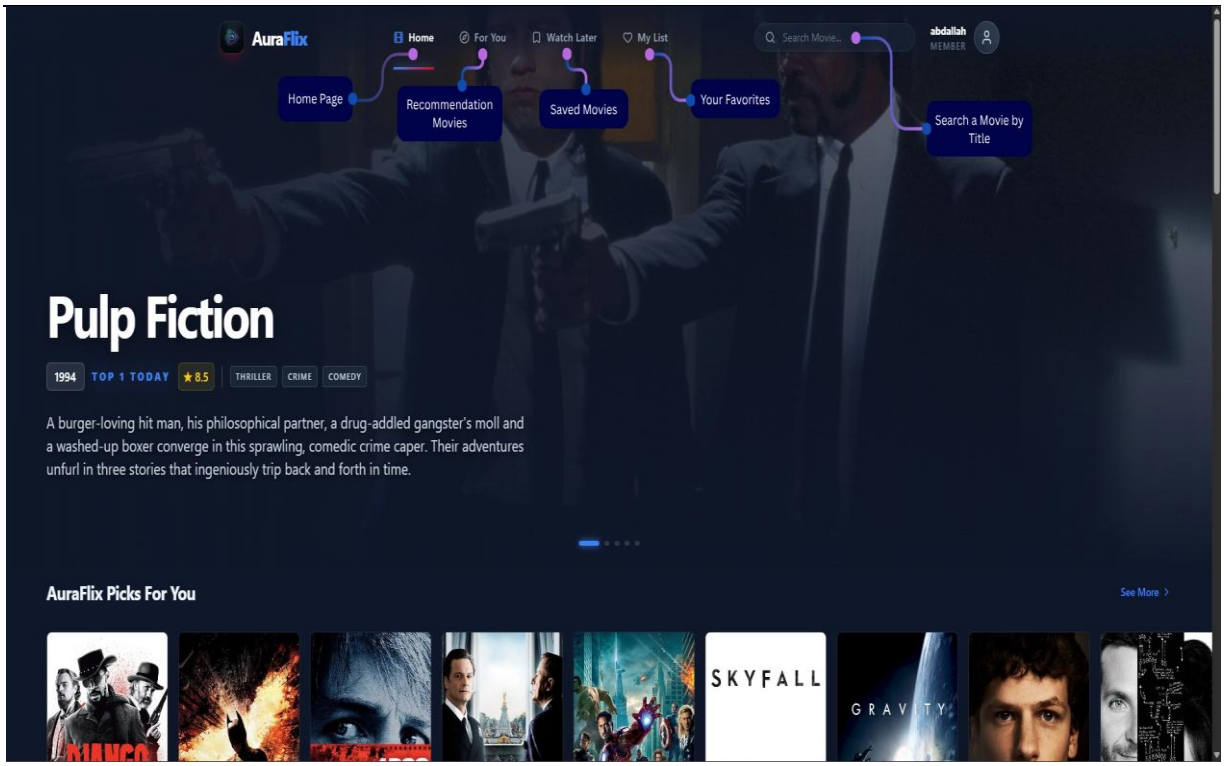

Figure 12:Home Page (Web Application)

Figure 13:Movie Card details (Web Application)

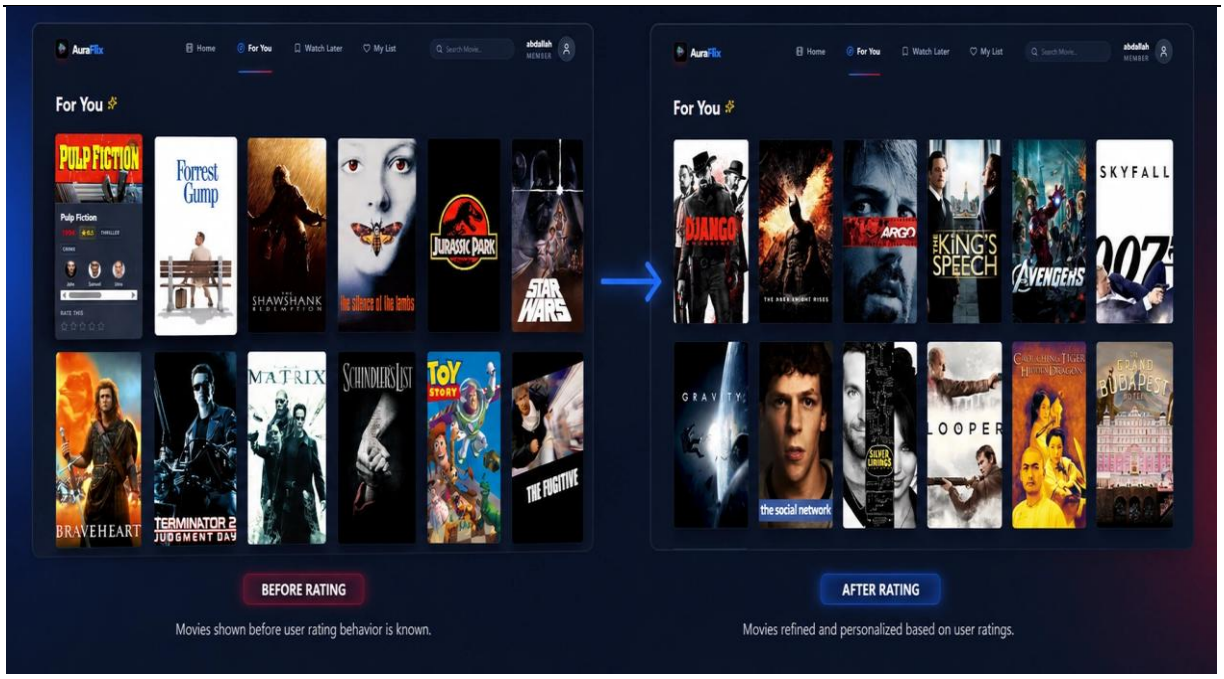

Figure 14:Transition of Personalized Recommendations Before and After User Rating Interaction (Web Application)

4.1.2 Mobile Application Interfaces

The mobile application was developed to provide a lightweight and responsive recommendation experience optimized for both Android and iPhone devices. Similar to the web platform, the mobile application enables users to explore movies, rate content, and access personalized recommendations directly from mobile interfaces.

The mobile application communicates with backend services through HTTP requests and dynamically updates recommendation results according to user interactions.

Special attention was given to responsive interface design and smooth navigation in order to improve usability across different mobile screen sizes.

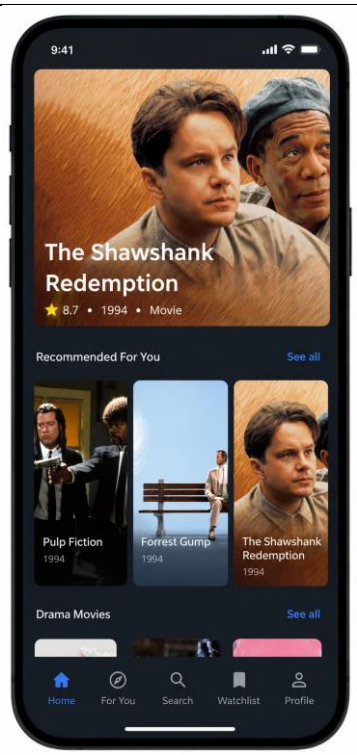

Figure 15: Home Page (Mobile App)

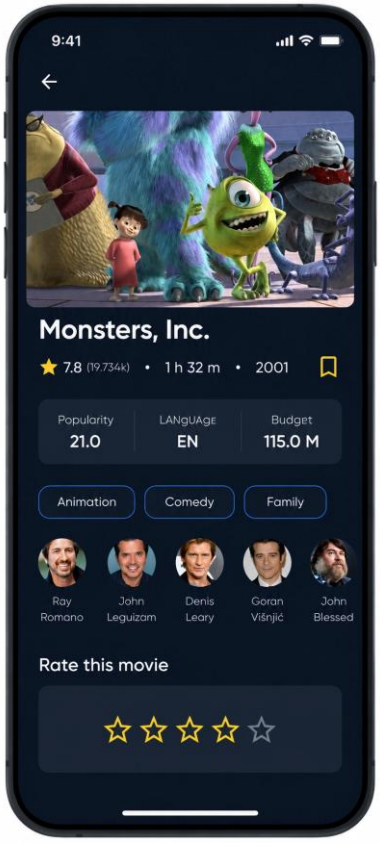

Figure 16:Movie Cards Details (Mobile App)

After the user submits a movie rating, the system analyzes the user's preferences and generates personalized recommendations using similarity signals such as genre, ratings, and movie features. This improves the experience by suggesting movies that closely match the user's interests and previously liked content. The recommendations are then displayed dynamically in For You Page interface to increase user engagement and satisfaction.

Figure 17: Transition of Personalized Recommendations Before and After User Rating Interaction (Mobile App)

4.2 Backend Implementation

The backend layer was implemented as the central communication component of the system. It is responsible for processing user requests, managing authentication operations, interacting with the database, and forwarding recommendation requests to the recommendation engine.

The backend exposes REST API endpoints used by both the web and mobile applications.

These endpoints handle operations such as:

- user authentication;
- movie retrieval;
- rating submission;
- recommendation requests;

- user profile management.

When a user submits a movie rating, the backend stores the interaction inside the database and forwards the corresponding request to the recommendation engine. After generating recommendation results, the backend returns the processed Top-N recommendations to the frontend applications in JSON format.

The backend architecture was designed to ensure modularity and efficient communication between the different system components.

4.3 Recommendation Engine and Dynamic Update

The implemented recommendation engine combines collaborative filtering and content-based filtering in order to generate personalized movie recommendations. The collaborative module relies on the ALS matrix factorization algorithm trained on the sparse user-item interaction matrix, while the content-based module exploits semantic embeddings generated from movie metadata using transformer-based language models and cosine similarity computation.

After generating both recommendation outputs, the system merges the obtained scores using the weighting coefficient α to produce the final ranked Top-N recommendation list delivered to users.

The recommendation process is continuously updated according to newly collected user interactions. When a user submits a movie rating, the interaction is stored in the database and integrated into the recommendation workflow in order to refresh personalized recommendation results.

To maintain model freshness and improve long-term recommendation quality, the system integrates an automatic retraining mechanism implemented through a dedicated scheduling process. The scheduler module runs independently alongside the recommendation API and uses **APScheduler** to periodically execute the retrain script every Sunday at 3:00 AM. This retraining process rebuilds the ALS model using newly collected interaction data, allowing the recommendation engine to progressively adapt to evolving user preferences over time.

5 System Evaluation

5.1 Evaluation Methodology

The evaluation of the proposed hybrid recommendation system was performed using an offline evaluation strategy based on historical user interactions. In order to simulate a realistic recommendation scenario, the recommendation models were trained using past interactions and evaluated on unseen future interactions extracted from the testing subset.

The evaluation process measures the capability of the system to generate relevant and personalized movie recommendations while maintaining recommendation quality and ranking effectiveness. For each user, the system produces a Top-N recommendation list which is then compared with the movies actually appreciated by the user in the testing data.

Several experiments were also conducted to analyze the impact of the hybrid weighting coefficient α on recommendation performance. Different values of α were tested in order to identify the best balance between collaborative filtering and content-based filtering contributions.

The obtained experimental results show that values of α close to **0.7** provide the best compromise between recommendation precision, ranking quality, and recommendation diversity.

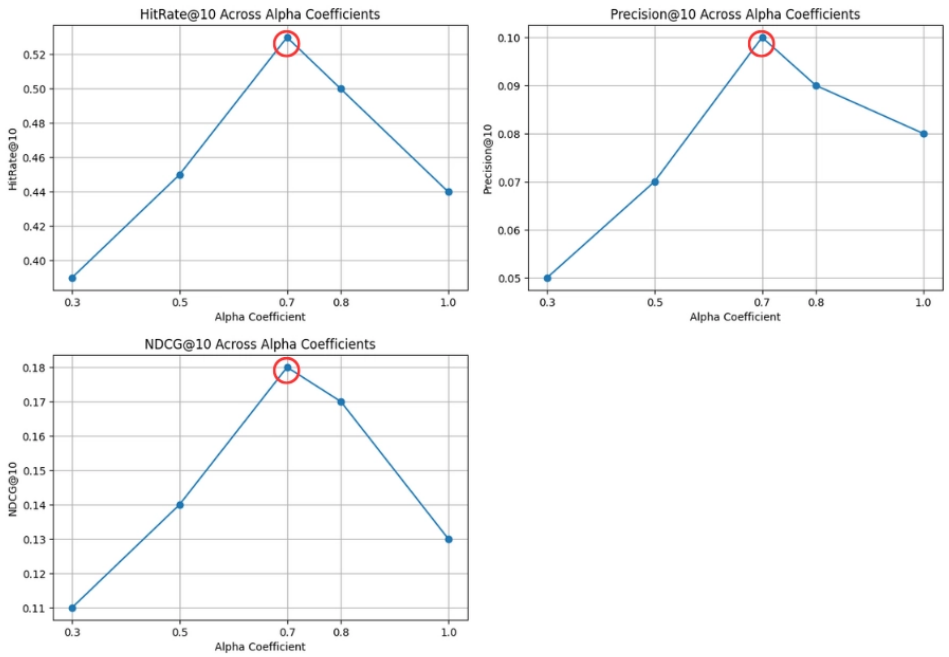

Figure 18:Performance Evaluation Across Different Alpha Values

5.2 Evaluation Metrics

To evaluate the effectiveness of the proposed recommendation approach, several recommendation metrics were used. Each metric measures a different aspect of recommendation quality and allows a more comprehensive analysis of system performance.

- **Precision:** Evaluates the proportion of recommended movies that are actually relevant to the user.

$$precision@K = \frac{Number\ of\ relevent\ items\ in\ top\ K}{K} = \frac{1}{K}\sum_{i=1}^{K} 1[r_i \in R]$$

Where:

- K is the number of recommendations shown to the user (e.g., top 10);
- $1[r_i \in R]$ counts 1 if the recommended item r_i is relevant, and 0 if it is not.

- **Recall:** Measures the capability of the system to retrieve relevant movies from all available relevant items.

$$Recall@K = \frac{number\ of\ relevant\ items\ in\ top\ K}{Total\ relevant\ items} = \frac{\sum_{i=1}^{K} 1[r_i \in R]}{|R|}$$

Where:

- **|R|** is the total number of movies the user actually likes.

- **NDCG:** evaluates the ranking quality of the recommendation list by assigning higher importance to relevant movies appearing at top positions.

First:

$$DCG@K = \sum_{i=1}^{k} \frac{1[r_i \in R]}{log_2(i+1)}$$

Where:

- position $\boldsymbol{i}$ starts at 1;
- log_2 ($\boldsymbol{i}$ +1) is a penalty that reduces the score the lower a relevant item appears in the list.

And:

$$IDCG@K = \sum_{i=1}^{min(|R|,k)} \frac{1}{log_2(i+1)}$$

Where:

- IDCG@K represents the best possible DCG score.

Then:

$$NDCG@k = \frac{DCG@K}{IDCG@k}$$

- **Hit Rate:** Measures whether at least one relevant movie appears in the generated recommendation list.

$$HR@k = min(1, \sum_{i=1}^{K} 1[r_i \in R])$$

Where:

- K is the number of top recommendations;
- r_i is the recommended item at position $\boldsymbol{i}$;
- R is the set of relevant items;
- 1[.]is the indicator function.

5.3 Experimental Results

The obtained results demonstrate the effectiveness of the proposed hybrid recommendation approach in improving recommendation quality and personalization performance.

Table 2:Evaluation Metrics of the Proposed Hybrid Recommendation System at Top-10

Metric	**Value**
Precision@10	0.10
Recall@10	0.05
NDCG@10	0.18
HitRate@10	0.53

The experimental results indicate that the hybrid recommendation approach improves recommendation relevance and ranking quality compared with recommendation systems relying exclusively on collaborative filtering or content-based filtering techniques.

The integration of semantic movie representations with collaborative preference learning enables the system to generate more balanced and personalized recommendations while maintaining acceptable recommendation diversity.

6 Discussion

The experimental results demonstrate the effectiveness of the proposed hybrid recommendation approach in improving recommendation relevance and personalization quality. The combination of collaborative filtering and content-based filtering enables the system to exploit both user interaction patterns and semantic movie information, leading to more balanced recommendation results.

The obtained results also highlight the importance of the weighting coefficient α\alphaα, where combining both recommendation techniques provides better performance than relying on a single approach. In addition, the dynamic retraining mechanism improves recommendation adaptability by continuously integrating newly collected user interactions.

Despite these improvements, recommendation quality may still be affected for users with very limited interaction histories, and periodic retraining may increase computational cost in large-scale environments. Overall, the proposed system provides an effective and scalable hybrid recommendation framework for personalized movie recommendation tasks.

7 Conclusion

This chapter presented the implementation of the proposed hybrid movie recommendation system, including the development environment, technologies, dataset preprocessing operations, and integration of the different system components. The implementation combined web and mobile applications with a hybrid recommendation engine based on collaborative and content-based filtering techniques.

The chapter also presented the experimental evaluation and discussion of the obtained results, the achieved performance demonstrates the effectiveness of the proposed hybrid approach in generating personalized movie recommendations while maintaining adaptability and recommendation quality.

General Conclusion

This work focused on the design and implementation of a hybrid movie recommendation system capable of generating personalized movie suggestions in large-scale digital entertainment environments. The proposed approach combines collaborative filtering based on ALS matrix factorization with content-based filtering using semantic embeddings and cosine similarity in order to exploit both user interaction patterns and movie descriptive information.

The first part of this thesis was dedicated to the study of recommender systems and the analysis of the principal recommendation approaches, techniques, and challenges related to recommendation generation. This theoretical background provided the necessary foundation for understanding hybrid recommendation strategies and selecting the methods adopted in this work.

The second part presented the design and implementation of the proposed system, including the system architecture, backend infrastructure, web and mobile applications, recommendation engine, and dynamic retraining mechanisms. The implementation also involved preprocessing large-scale movie interaction data and integrating recommendation models capable of adapting continuously to evolving user preferences.

Finally, the obtained experimental results demonstrated the effectiveness of the proposed hybrid recommendation approach in improving recommendation relevance and personalization quality. Although several improvements remain possible for future developments, this work provides a scalable and extensible foundation for the development of intelligent recommendation systems dedicated to personalized digital entertainment services.

References

[1] X. Amatriain, “Big & Personal: Data and Models Behind Netflix Recommendations,” in Proc. 2nd Int. Workshop on Big Data, Streams and Heterogeneous Source Mining, 2013, pp. 1–6.

[2] F. Ricci, L. Rokach, and B. Shapira, “Introduction to Recommender Systems,” in Recommender Systems Handbook, 2nd ed., Boston, MA, USA: Springer, 2015, ch. 1, pp. 1–35.

[3] F. M. Harper and J. A. Konstan, “The MovieLens Datasets: History and Context,” ACM Trans. Interact. Intell. Syst., vol. 5, no. 4, Article 19, pp. 1–19, 2015.

[4] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, “Evaluating collaborative filtering recommender systems,” ACM Trans. Inf. Syst., vol. 22, no. 1, pp. 5–53, Jan. 2004.

[5] M. J. Pazzani and D. Billsus, “Content-Based Recommendation Systems,” in The Adaptive Web, Berlin, Germany: Springer, 2007, pp. 325–341.

[6] R. Burke, “Hybrid recommender systems: Survey and experiments,” User Model. User-Adapt. Interact., vol. 12, no. 4, pp. 331–370, Nov. 2002.

[7] C. C. Aggarwal, “Matrix Factorization Techniques,” in Recommender Systems: The Textbook, Cham, Switzerland: Springer, 2016, ch. 3, pp. 71–138.

[8] S. Rendle, “Factorization Machines,” in Proc. IEEE Int. Conf. on Data Mining (ICDM), Sydney, Australia, 2010, pp. 995–1000.

[9] X. He et al., “Neural Collaborative Filtering,” in Proc. Int. World Wide Web Conf. (WWW), 2017, pp. 173–182.

[10] P. Covington, J. Adams, and E. Sargin, “Deep Neural Networks for YouTube Recommendations,” in Proc. ACM Conf. on Recommender Systems (RecSys), 2016, pp. 191–198.

[11] G. Shani and A. Gunawardana, “Evaluating Recommendation Systems,” in Recommender Systems Handbook, 2nd ed., Boston, MA, USA: Springer, 2015, ch. 8, pp. 265–308.

[12] A. Schein, A. Popescul, L. Ungar, and D. Pennock, “Methods and Metrics for Cold-Start Recommendations,” in Proc. 25th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, Tampere, Finland, 2002, pp. 253–260.

[13] Y. Koren, R. Bell, and C. Volinsky, “Matrix Factorization Techniques for Recommender Systems,” Computer, vol. 42, no. 8, pp. 30–37, Aug. 2009.

[14] G. Linden, B. Smith, and J. York, “Amazon.com Recommendations: Item-to-Item Collaborative Filtering,” IEEE Internet Computing, vol. 7, no. 1, pp. 76–80, Jan.–Feb. 2003.

[15] M. D. Ekstrand, A. Burke, and F. Diaz, “Fairness and Discrimination in Recommendation and Retrieval,” in Recommender Systems Handbook, 3rd ed., Cham, Switzerland: Springer, 2022, pp. 767–803.

[16] J. Canny, “Collaborative Filtering with Privacy,” in Proc. IEEE Symp. on Security and Privacy, Berkeley, CA, USA, 2002, pp. 45–57.

[17] R. Burke, “Hybrid Recommender Systems: Survey and Experiments,” *User Modeling and User-Adapted Interaction*, vol. 12, no. 4, pp. 331–370, 2002.

[18] F. Ricci, L. Rokach, and B. Shapira, *Recommender Systems Handbook*, 3rd ed. New York, NY, USA: Springer, 2022.

[19] Y. Koren, R. Bell, and C. Volinsky, “Matrix Factorization Techniques for Recommender Systems,” *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[20] X. Su and T. M. Khoshgoftaar, “A Survey of Collaborative Filtering Techniques,” *Advances in Artificial Intelligence*, vol. 2009, pp. 1–19, 2009.

[21] Y. Hu, Y. Koren, and C. Volinsky, “Collaborative Filtering for Implicit Feedback Datasets,” in *Proceedings of the 8th IEEE International Conference on Data Mining*, Pisa, Italy, 2008, pp. 263–272.

[22] G. Takács, I. Pilászy, B. Németh, and D. Tikk, “Scalable Collaborative Filtering Approaches for Large Recommender Systems,” *Journal of Machine Learning Research*, vol. 10, pp. 623–656, 2009.

[23] Y. Hu, Y. Koren, and C. Volinsky, “Collaborative Filtering for Implicit Feedback Datasets,” in *Proceedings of the 8th IEEE International Conference on Data Mining*, Pisa, Italy, 2008, pp. 263–272.

[24] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Hong Kong, China, 2019, pp. 3982–3992.

[25] A. Wang, O. Michael, Y. Liu, et al., “MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers,” in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 5776–5788.

[26] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” in *Proceedings of NAACL-HLT*, Minneapolis, MN, USA, 2019, pp. 4171–4186.

[27] M. Balabanović and Y. Shoham, “Fab: Content-Based, Collaborative Recommendation,” *Communications of the ACM*, vol. 40, no. 3, pp. 66–72, 1997.

[28] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, “Recommender Systems Survey,” *Knowledge-Based Systems*, vol. 46, pp. 109–132, 2013.

[29] A. Bobadilla, F. Ortega, A. Hernando, and J. Bernal, “A Collaborative Filtering Approach to Mitigate the New User Cold Start Problem,” *Knowledge-Based Systems*, vol. 26, pp. 225–238, 2012.

[30] M. Park and J. Chang, “Individual and Group Behavior-Based Customer Profile Model for Personalized Product Recommendation,” *Expert Systems with Applications*, vol. 36, no. 2, pp. 1932–1939, 2009.

[31] J. Ben Schafer, J. Konstan, and J. Riedl, “Recommender Systems in E-Commerce,” in *Proceedings of the 1st ACM Conference on Electronic Commerce*, Denver, CO, USA, 1999, pp. 158–166.

[32] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, 2nd ed. Boston, MA, USA: Addison-Wesley Professional, 2005.

[33] A. Dennis, B. H. Wixom, and D. Tegarden, *Systems Analysis and Design: An Object-Oriented Approach with UML*, 5th ed. Hoboken, NJ, USA: John Wiley & Sons, 2015.

[34] C. Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2004.

[35] Visual Studio Code. Accessed on: 2026/05/21. Available: https://code.visualstudio.com/docs

[36] Git and GitHub. Accessed on: 2026/05/21. Available: https://git-scm.com/docs ,https://docs.github.com/en

[37] React and Vite. Accessed on: 2026/05/21. Available: https://react.dev ,https://vite.dev

[38] Tailwind CSS. Accessed on: 2026/05/21. Available: https://tailwindcss.com/docs/installation/using-vite

[39] React Router. Accessed on: 2026/05/21. Available: https://reactrouter.com/home

[40] Axios. Accessed on: 2026/05/21. Available: https://axios.rest/pages/getting-started/first-steps

[41] Framer Motion. Accessed on: 2026/05/21. Available: https://motion.dev/docs/react

[42] Flutter. Accessed on: 2026/05/21. Available: https://docs.flutter.dev/

[43] Dart. Accessed on: 2026/05/21. Available: https://dart.dev/docs

[44] Riverpod. Accessed on: 2026/05/21. Available: https://riverpod.dev/docs/introduction/getting_started

[45] Dio. Accessed on: 2026/05/21. Available: https://pub.dev/packages/dio

[46] Node.js and Express.js. Accessed on: 2026/05/21. Available: https://nodejs.org/docs/latest/api/ ,https://expressjs.com/en/

[47] MongoDB and Mongoose. Accessed on: 2026/05/21. Available: https://www.mongodb.com/docs/ ,https://mongoosejs.com/docs

[48] JWT and bcryptjs. Accessed on: 2026/05/21. Available: https://www.jwt.io/introduction#what-is-json-web-token ,https://www.npmjs.com/package/bcryptjs

[49] FastAPI and Uvicorn. Accessed on: 2026/05/21. Available: https://fastapi.tiangolo.com/ ,https://uvicorn.dev/

[50] implicit Library. Accessed on: 2026/05/21. Available: https://benfred.github.io/implicit/

[51] Sentence Transformers. Accessed on: 2026/05/21. Available: https://www.sbert.net/

[52] Pandas and NumPy. Accessed on: 2026/05/21. Available: https://pandas.pydata.org/docs/ ,https://numpy.org/doc/

[53] scikit-learn. Accessed on: 2026/05/21. Available: https://scikit-learn.org/stable/index.html

[54] APScheduler. Accessed on: 2026/05/21. Available: https://apscheduler.readthedocs.io/en/stable/

[55] PyMongo. Accessed on: 2026/05/21. Available: https://pymongo.readthedocs.io/en/stable/

Appendix: List of Acronyms

RS: Recommendation System

CB: Content-Based

CF: Collaborative Filtering

ML: machine learning

DL: Deep Learning

NDCG: Normalized Discounted Cumulative Gain

ALS: Alternating Least Squares