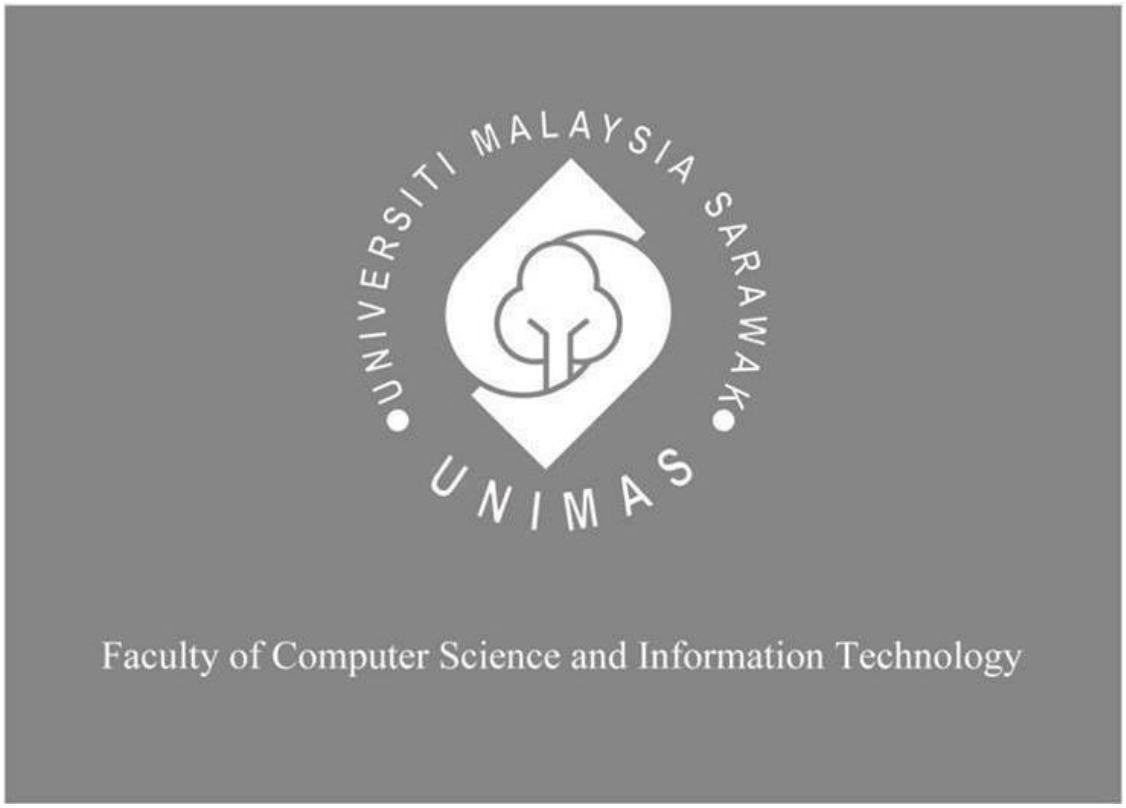


HYBRID PRODUCT RECOMMENDATION SYSTEM USING CONTENT BASED FILTERING AND NEURAL COLLABORATIVE FILTERING

JOSH TING SIONG LUNG

Bachelor of Computer Science with Honours

(Information Systems)

2025

UNIVERSITI MALAYSIA SARAWAK

THESIS STATUS ENDORSEMENT FORM

TITLE Hybrid Product Recommendation System Using Content-based Filtering and Collaborative Filtering

ACADEMIC SESSION: 2024/2025

JOSH TING SIONG LUNG

hereby agree that this Thesis* shall be kept at the Centre for Academic Information Services, Universiti Malaysia Sarawak, subject to the following terms and conditions:

1. The Thesis is solely owned by Universiti Malaysia Sarawak
2. The Centre for Academic Information Services is given full rights to produce copies for educational purposes only
3. The Centre for Academic Information Services is given full rights to do digitization in order to develop local content database
4. The Centre for Academic Information Services is given full rights to produce copies of this Thesis as part of its exchange item program between Higher Learning Institutions [or for the purpose of interlibrary loan between HLI]
5. ** Please tick (√)

	CONFIDENTIAL	(Contains classified information bounded by the OFFICIAL SECRETS ACT 1972)
	RESTRICTED	(Contains restricted information as dictated by the body or organization where the research was conducted)
√	UNRESTRICTED	

Validated by

(AUTHOR'S SIGNATURE)

(SUPERVISOR'S SIGNATURE)

Permanent Address

5A, Jalan Foochow, 96000 Sibu, Sarawak

Date: 23/07/2025

Date: 25/7/2025

Note * Thesis refers to PhD, Master, and Bachelor Degree

** For Confidential or Restricted materials, please attach relevant documents from relevant organizations / authorities

HYBRID PRODUCT RECOMMENDATION SYSTEM USING CONTENT BASED FILTERING AND NEURAL COLLABORATIVE FILTERING

JOSH TING SIONG LUNG

This project is submitted in partial fulfilment of the requirements for the degree of Bachelor of Computer Science with Honours (Software Engineering)

Faculty of Computer Science and Information Technology
UNIVERSITI MALAYSIA SARAWAK
2025

SISTEM CADANGAN PRODUK HIBRID MENGGUNAKAN PENAPISAN BERASASKAN KANDUNGAN DAN PENAPISAN KOLABORATIF NEURAL

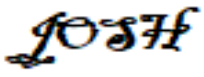
JOSH TING SIONG LUNG

Projek ini merupakan salah satu keperluan untuk Ijazah Sarjana Muda Sains Komputer dengan Kepujian (Kejuruteraan Perisian)

Fakulti Sains Komputer dan Teknologi Maklumat
UNIVERSITI MALAYSIA SARAWAK
2025

Declaration

I hereby declare that this project is my original work. I have not copied from any other student's work or from any other sources except where due reference acknowledgement is not made explicitly in the text, nor has any part had been written for me by another person.



……

(JOSH TING SIONG LUNG) 13 July 2025

Faculty of Computer Science and Information Technology

Universiti Malaysia Sarawak

Acknowledgement

Firstly, I would like to thank God Almighty for the blessings and opportunities to have come this far. I would also like to express my sincere gratitude and appreciation to my Final Year Project supervisor, Dr. Chiu Po Chan, for her guidance, patience, and supervision throughout this final year project. I am honoured to be given the opportunity to work with her on this project apart from receiving valuable insights and motivation. I would also like to extend my heartfelt gratitude to my examiner, Professor Dr. Narayanan A/L N Kulathu Ramaiyer, for his valuable feedback and constructive suggestions, which have greatly contributed to improving the quality of this project. Next, I would like to heartily thank my family for the constant love, support and encouragement that fuels me with more passion and motivation that I need as a lifelong learner. Lastly, I would like to thank my friends for the conversations shared over the ideas and improvements that could be implemented to refine our final year projects.

Table of Contents

List of Figures

List of Tables

Abstract

With the rapid growth of e-commerce platforms, users are faced with an overwhelming number of product choices. This has created a pressing need for more intelligent and personalized recommendation systems to enhance user experience and drive engagement. Motivated by this trend, this project aims to develop a hybrid recommendation system that combines Content-Based Filtering (CBF) and Neural Collaborative Filtering (NCF) to deliver accurate and relevant product suggestions. The system leverages item attributes such as category and brand in the CBF component, while the NCF model captures deeper user-item interaction patterns. A custom e-commerce prototype was built using Flask to demonstrate how the hybrid model can be integrated into a real-world application. The model was evaluated using metrics like NDCG, precision, and recall, as well as time efficiency to ensure practical usability. Despite limitations such as data quality and sparsity, the project successfully showcases the potential of hybrid approaches in improving recommendation quality. Future improvements may include incorporating more diverse data sources and refining the model for better scalability and performance.

Abstrak

Dengan pertumbuhan pesat platform e-dagang, pengguna kini berhadapan dengan terlalu banyak pilihan produk. Situasi ini menimbulkan keperluan mendesak untuk sistem cadangan yang lebih pintar dan diperibadikan bagi meningkatkan pengalaman pengguna dan menarik penglibatan. Projek ini dimotivasikan oleh perkembangan tersebut dengan tujuan untuk membangunkan sistem cadangan hibrid yang menggabungkan Content-Based Filtering (CBF) dan Neural Collaborative Filtering (NCF) bagi memberikan cadangan produk yang lebih tepat dan relevan. Komponen CBF menggunakan atribut produk seperti kategori dan jenama, manakala model NCF mengenal pasti corak interaksi pengguna-item yang lebih mendalam. Prototaip laman e-dagang dibina menggunakan Flask bagi menunjukkan bagaimana model hibrid ini dapat disepadukan dalam aplikasi sebenar. Penilaian model dijalankan menggunakan metrik seperti NDCG, precision, dan recall, serta masa pemprosesan untuk memastikan kebolehgunaan secara praktikal. Walaupun terdapat beberapa kekangan seperti kualiti dan kepadatan data, projek ini berjaya membuktikan potensi pendekatan hibrid dalam meningkatkan kualiti sistem cadangan. Penambahbaikan masa hadapan boleh merangkumi penggunaan sumber data yang lebih pelbagai dan pengoptimuman model bagi prestasi yang lebih baik dan berskala.

Chapter 1: Introduction

1.1 Introduction

Recommender Systems are commonly found and extensively applied in various fields. Specifically, their usage in e-commerce has led to their increased popularity, as they assist users in discovering products that align with their individual preferences. Recommender Systems enable users to search through an overwhelming number of choices (referred to as items) offered by commercial services. Consequently, they have become a crucial technology for e-commerce. Acting as virtual experts, these systems understand users' preferences and tastes and effectively filter out vast amounts of irrelevant information to suggest the most suitable products. A typical example is a product recommendation system designed to aid users in choosing an item from a product catalog. Well-known online shopping platforms like Amazon utilize a product recommendation system to tailor the experience to each individual user. The suggestions provided are based on the user's purchase history, resulting in distinctive recommendations for different users.

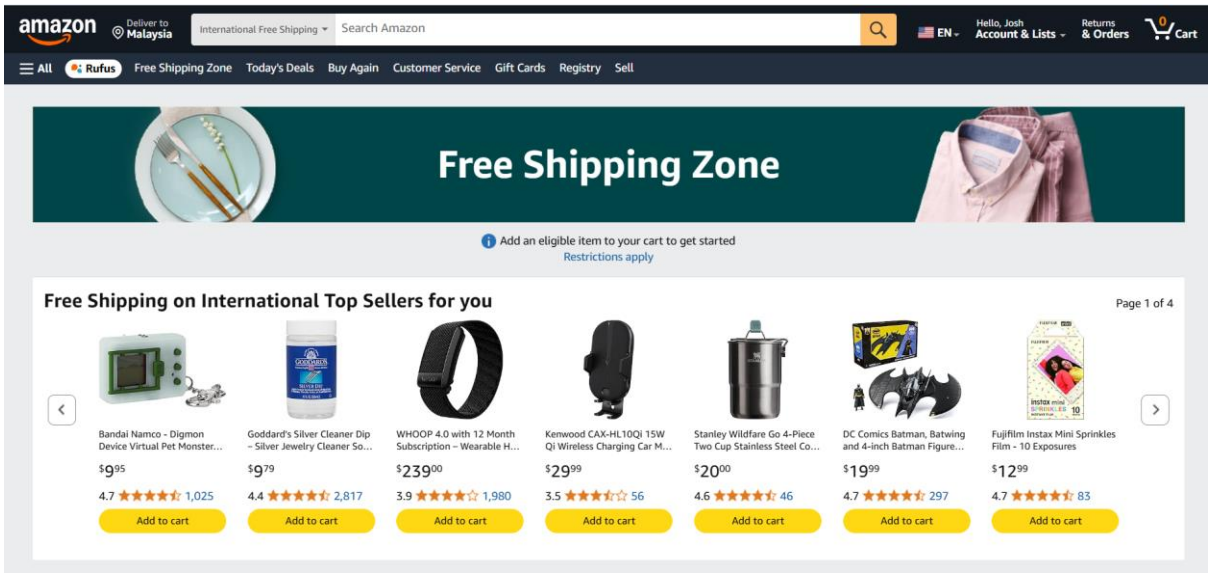


Figure 1.1 Amazon e-commerce website (Amazon, 2024)

Recommendation systems can be broadly categorized into three types: collaborative filtering, content-based filtering, and hybrid methods. Collaborative filtering relies on the collective behavior of users to make recommendations. It identifies patterns based on similar users' preferences. On the other hand, content-based filtering recommends items based on the

attributes of products and the user's past interactions. Hybrid methods combine both approaches to utilize their strengths and mitigate weaknesses.

In e-commerce, recommendation systems play a critical role in enhancing customer engagement, increasing conversion rates, and driving sales. By providing personalized experiences, these systems help users discover products they might not have found otherwise. Despite advancements in recommendation technologies, many existing systems struggle to fully capture the complexities of data in recommendation systems.

1.2 Problem Statement

As the e-commerce industry continues to expand, the number of users and items increases gradually. As a result, the computational resources needed to train and deploy the recommendation system could become a bottleneck, which can hinder scalability (C. Li et al., 2023). Additionally, deep learning techniques may introduce ethical issues, including potential privacy violations or biases in recommendations (Zhang & Zhang, 2021).

With the increase in product variety, consumers are confronted with an overwhelming array of products across various categories. This vast selection often leads to decision fatigue where the abundance of options makes it difficult for users to make satisfying decisions. When faced with too many choices, shoppers may experience frustration or anxiety, which can negatively impact their overall shopping experience.

Currently, many existing systems have inherent limitations that hinder their effectiveness. They often rely on simplistic models that may not fully capture the complexities of user behavior and preferences (Chirravuri & Immidi, 2022). For instance, many systems are still using Simple Algorithm for Recommendation (SAR) and K-nearest neighbors (KNN) based collaborative filtering methods. Both SAR and KNN methods can struggle with scalability as the size of the user-item interaction matrix increases. KNN, in particular, requires calculating similarities between all pairs of users or items, which can become computationally expensive with large datasets.

Another limitation is the sparsity of data. Data sparsity refers to the phenomenon where the user-item interaction matrix is predominantly empty or contains very few ratings (Choi et al., 2023). E-commerce platforms typically have vast catalogues with millions of products and numerous users. In many cases, users only rate a limited number of items, leading to a situation

where most of the potential user-item interactions are unobserved (Trabelsi et al., 2021). SAR and KNN methods rely heavily on user-item interactions to make recommendations. In sparse datasets, where many items are not rated by enough users, these algorithms may fail to provide meaningful recommendations due to insufficient data.

In e-commerce, effective recommendation systems are essential for enhancing user experience and engagement. Content-based filtering (CBF) recommends items based on their features and a user's past interactions. It analyzes item attributes such as product categories to suggests similar items to those the user has previously engaged with. As stated by Liliana et al. (2024), CBF typically encounters fewer issues compared to collaborative filtering method but may yield less performance. To solve this performance issue, neural collaborative filtering (NCF) can be applied to generate more personalized product recommendations. NCF is a deep learning-based approach that models user-item interactions using neural networks. It extends traditional collaborative filtering by learning complex, non-linear patterns in user preferences. Therefore, there is a need to use hybrid recommendation model that includes content-based filtering (CBF) and neural collaborative filtering (NCF) to ensure more accurate recommendations.

1.3 Objectives

The aim of this project is to develop an e-commerce website that integrates deep learning model for electronic products recommendations. According to the problem statement, the objectives of the project are listed below:

1. To propose a web-based system that integrates hybrid recommendation model for product recommendation system.
2. To evaluate the performance of various deep learning models in the context of product recommendation system.

1.4 Research Methodology

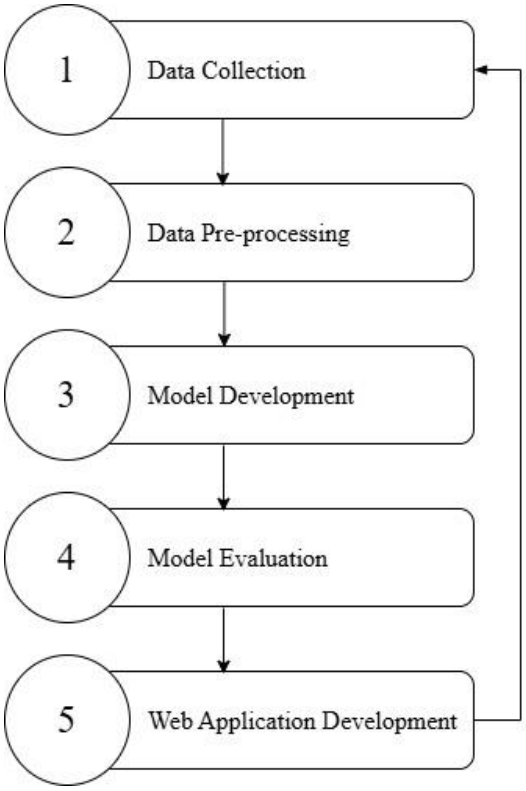


Figure 1.2 Research Methodology

The first phase involves gathering datasets that include the transaction history of all users, and most importantly, the rating given by the user for that particular product. This is important for analyzing product recommendations as users tend to prefer products with higher ratings.

In phase 2, the data collected undergoes preprocessing to ensure it is clean and suitable for analysis. This mostly includes data cleaning processes such as removing duplicates, handling missing values, and correcting inconsistencies in the dataset. In the meantime, it is crucial to convert categorical data like product categories into numerical formats, so that it is suitable for machine learning algorithms.

In phase 3, several recommendation algorithms are implemented based on deep learning techniques and traditional methods.

In phase 4, the performance of each developed model will be assessed. various metrics such as Mean Average Precision (MAP) and Normalized Discounted Cumulative Gain (NDCG) will be calculated to evaluate how well the recommendations align with actual user preferences.

In phase 5, a web-based application will be developed after evaluating the models. The best model with the overall highest accuracy will be chosen.

1.5 Scope

This project will focus on integrating a recommendation system into a functional web application, providing users with personalized electronic product recommendations. The backend of the application will be developed using Flask, a versatile Python web framework to enable seamless integration with the machine learning models. Additionally, the system will utilize Python's powerful machine learning libraries to train, evaluate, and deploy the models.

1.6 Significance of Project

The recommendation system enhances the user's shopping experience by providing personalized recommendations. It reduces decision fatigue and improves satisfaction. On the other hand, it increases sales for the retailers. Businesses can improve customer retention and loyalty, which leads to higher revenue.

1.7 Schedule

Schedule for FYP 1

				2024			2025	
TASK	START DATE	END DATE	DAYS	October	November	December	January	February
Final Year Project 1	9/10/24	19/2/25	133					
Phase 1: Planning	9/10/24	21/11/24	44					
Propose project title	9/10/24	9/10/24	1					
Submit brief proposal	10/10/24	19/10/24	10					
Submit full proposal	20/10/24	27/10/24	8					
Submit chapter 1: Introduction	28/10/24	21/11/24	25					
Phase 2: Analysis	22/11/24	13/12/24	22					
Research related work	22/11/24	28/11/24	7					
Recommendation system Literature Review	29/11/24	12/12/24	14					
Submit Chapter 2: Literature Review	13/12/24	13/12/24	1					
Phase 3: Methodology	14/12/24	5/1/25	23					
Writing methodology	14/12/24	24/12/24	11					
Complete UML diagram	25/12/24	4/1/25	11					
Submit Chapter 3: Methodology	5/1/25	5/1/25	1					
FYP1 Report Submission	6/1/25	17/1/25	12					
FYP1 Symposium	18/1/25	25/1/25	8					
FYP1 Report Amendment	26/1/25	10/2/25	16					
FYP1 Final Report Submission	11/2/25	19/2/25	9					

Figure 1.3.1 Gantt Chart FYP1

Schedule for FYP 2

				2025				
TASK	START DATE	END DATE	DAYS	March	April	May	June	July
Final Year Project 2	17/3/25	28/7/25	134					
Phase 4: Implementation	17/3/25	15/5/25	60					
Develop e-commerce prototype	17/3/25	31/3/25	15					
Develop recommendation models	1/4/25	15/4/25	15					
Testing	16/4/25	30/4/25	15					
Analysis of results	1/5/25	15/5/25	14					
Phase 5: Conclusion	16/5/25	31/5/25	15					
Conclude objectives achieved	16/5/25	20/5/25	5					
Determine project limitations	21/5/25	26/5/25	5					
Propose future work	27/5/25	31/5/25	4					
FYP Report Submission	1/6/25	10/6/25	10					
FYP Full Submission	11/6/25	23/6/25	13					
FYP Symposium	24/6/25	2/7/25	8					
FYP Final Amendment	3/7/25	27/7/25	24					
FYP Final Submission	28/7/25	28/7/25	1					

Figure 1.3.2 Gantt Chart FYP2

1.8 Expected outcome

The expected outcome is a functional web application that integrates a deep learning-based recommendation system capable of providing personalized product suggestions. The system will be designed to adapt to user preferences over time, continually improving its recommendations based on new data.

1.9 Project Outline

Chapter 1: Introduction

This chapter provides background and overview of this project. This includes the problem statement, objectives, methodology, project scope, significance of the project, project schedule, and expected outcome.

Chapter 2: Background Study

This chapter outlines the existing recommendation system which will serve as a reference for the proposed project. A comparison of each of the existing systems with the proposed system will be discussed.

Chapter 3: Methodology

This chapter introduces the system design and methodology of this proposed project. The implementation of the methods will be explained in depth in this chapter.

Chapter 4: Implementation and Result

This chapter focuses on the implementation and evaluation of the proposed recommendation system. The hybrid recommendation model will use both content based filtering and neural collaborative filtering. Other benchmark models for the recommendation system will be evaluated and be compared to the proposed model.

Chapter 5: Web Prototype Development

This chapter focuses on the implementation and testing of the proposed recommendation system. Python and Flask will be used to develop the recommendation system and the web application. Different algorithms for the recommendation system will be tested, and their performance will be evaluated.

Chapter 6: Conclusion and Future Work

This chapter summarizes the project and addresses its limitations. Potential future work will also be discussed to further improve the recommendation system.

Chapter 2: Literature Review

2.1 Introduction

In this chapter, background study and literature review of the previous existing recommendation system will be discussed. The main purpose of conducting a literature review is to gain a thorough understanding of the landscape of recommendation algorithms, including collaborative filtering and content-based filtering, as well as more advanced techniques like hybrid models and deep learning. This knowledge is essential for selecting the most appropriate algorithms for the proposed recommendation system.

2.2 Background study

A recommendation system is a type of software or algorithm designed to suggest relevant items to users. Its goal is to enhance the user experience by presenting items that align with their preferences, behaviors, or needs. The types of recommendation system models mainly include content-based filtering, collaborative filtering, and hybrid systems (Ko et al., 2022).

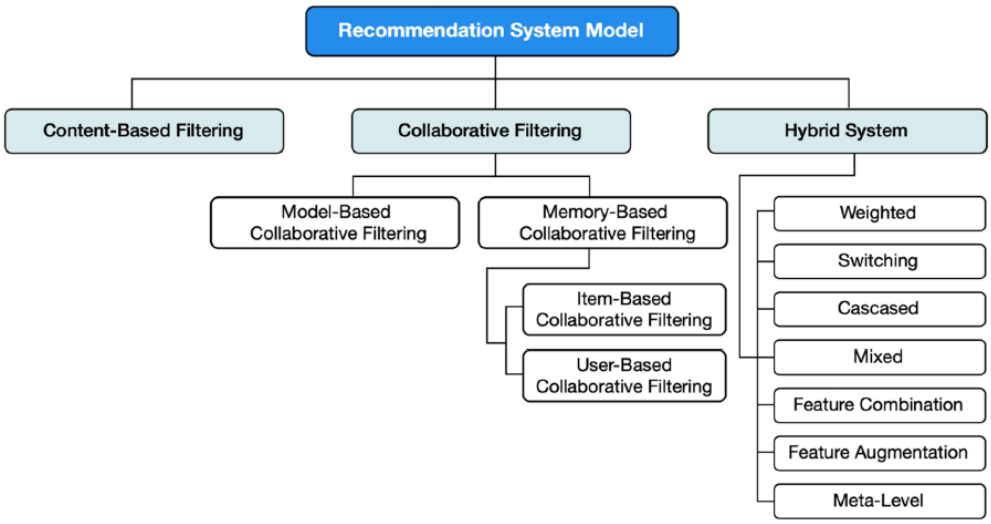


Figure 2.1: Overview of recommendation models.

2.2.1 Content-Based Filtering

Content-based filtering is a method employed to recommend items that possess attributes similar to those favored by users (Cantador et al., 2008). This is a technique for suggesting

similar products based on the user's previous selections. However, Content-Based Filtering recommends only items that are closely related to those the user has previously rated, resulting in a limitation where it doesn't suggest new items (Salter & Antonopoulos, 2006). This means that users miss out on opportunities to explore a new diverse range of content. Due to these constraints, this model is primarily applied in services that suggest items or text-based content that can be easily recommended based on both item information and user profile data, such as music, movies and e-commerce recommendation systems.

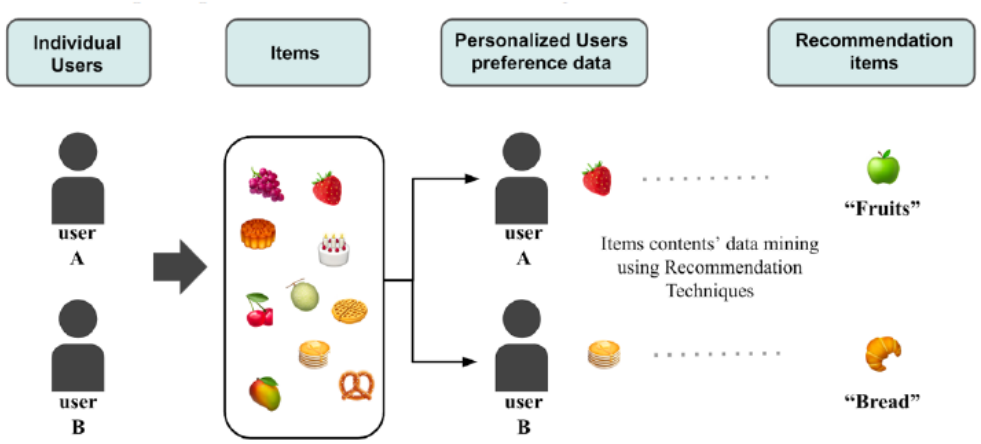


Figure 2.2: Principle of Recommendation in a Contents-Based Filtering Model

2.2.2 Collaborative Filtering

Collaborative Filtering is a model that develops a user's preference database by utilizing the user's evaluation data to forecast items that align with their individual tastes, subsequently using this information for recommendations (Im & Hars, 2007). This model has been widely adopted and utilized as a recommendation system, often more than Content-Based Filtering (Ko et al., 2022). However, despite advancements in collaborative filtering, challenges related to scalability and data sparsity remain unresolved, leading to limitations in the overall accuracy of recommendations (Barragáns-Martínez et al., 2010). To address these challenges, there has been a significant increase in the application of Hybrid System filtering models that combine elements of Content-Based Filtering.

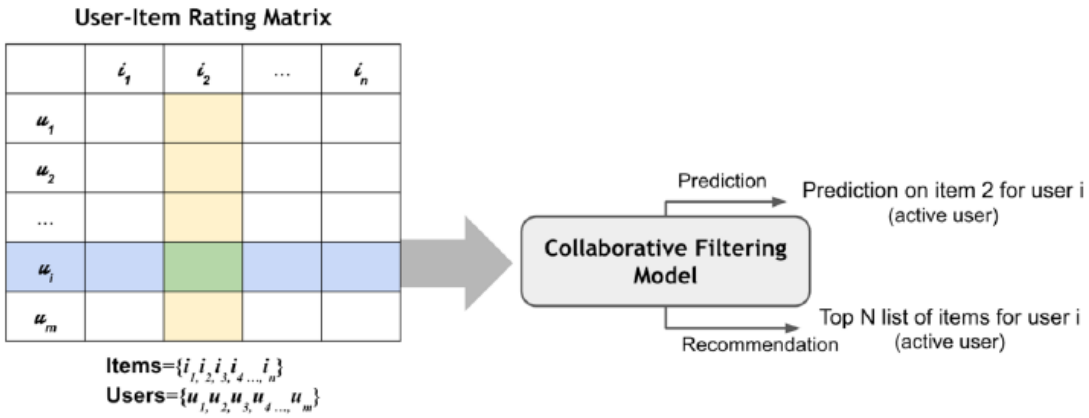


Figure 2.3: Principle of Recommendation in a Collaborative Filtering Model

2.2.3 Hybrid System

To address the shortcomings of content-based and collaborative filtering models, a Hybrid recommendation model has been proposed (Burke, 2002). The Hybrid Recommendation model is primarily designed to address the issue of data sparsity. As such, the main objective of many studies focused on this model is to increase the limited rating data through the integration of Content-Based Filtering and Collaborative Filtering information.

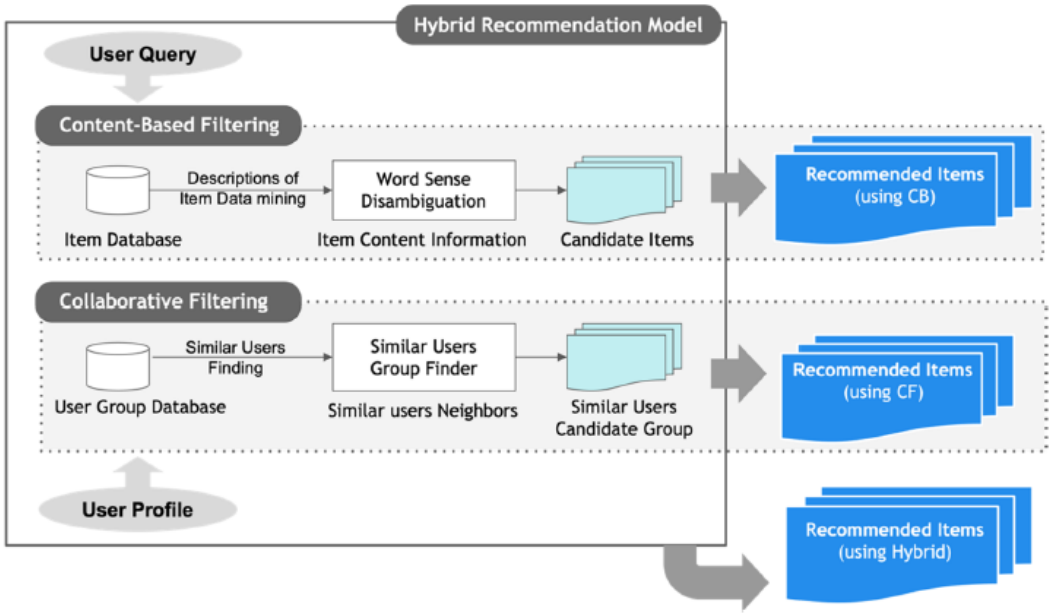


Figure 2.4: Principle of Recommendation in a Hybrid Model

2.3 Existing techniques used in the recommendation system

2.3.1 Content-based filtering (CBF)

Liliana et al. (2024) proposed a recommendation system using Content-Based Filtering (CBF). The system uses CBF to analyze the inherent features of items. Then, it compares these features with user input to recommend the most relevant items.

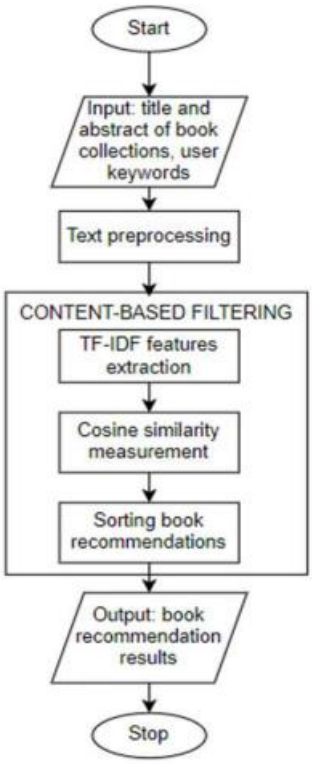


Figure 2.5: Overview of the proposed system

The model utilizes Term Frequency-Inverse Document Frequency (TF-IDF) to compute the importance of terms within item descriptions. To do this, it employs Cosine Similarity to measure how closely user queries match item features. Then, the top N items with the highest similarity scores are identified and presented to users as recommendations.

The performance of this model was evaluated. It achieved an accuracy of 90%. Liliana et al. (2024) stated that some irrelevant recommendations occurred when keywords matched frequently in less contextually relevant abstracts. That means a lack of contextual understanding may reduce relevance in some cases. The authors suggest combining CBF with Collaborative Filtering to enhance this model.

In 2020, Singla et al. proposed a recommendation system using TF-IDF and Doc2Vec for CBF. Similar to the proposed system by Liliana et al. (2024), this model also uses TF-IDF to calculate the similarity between items. However, this model also employs Doc2Vec to capture semantic similarities between items. A weighted formula combines all these similarities to produce the final recommendation list.

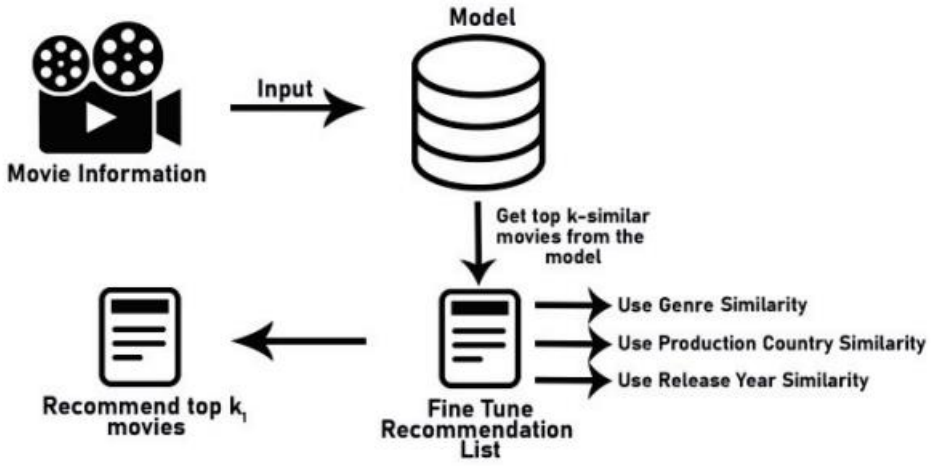


Figure 2.6: Overall flow of the proposed model

This model was evaluated to test its performance. It achieved a precision of 0.66. Singla et al. (2020) suggested utilizing user purchase history and geographically contextual trends for personalized recommendations. Same with Liliana et al. (2024), the authors also suggest that combine the content-based model with collaborative filtering techniques for a hybrid approach.

In 2020, Raheem & Ali proposed a modified and extended version of the CBF recommender system. The system builds item profiles and user profiles, then compares the similarity between user preferences (user profiles) and item characteristics (item profiles). Unlike CBF models proposed by Liliana et al. (2024) and Singla et al. (2020) which uses metadata or attributes of items, this paper employs affective-automated features (AAF) derived from user facial expressions with the help of Convolutional Neural Networks (CNN).

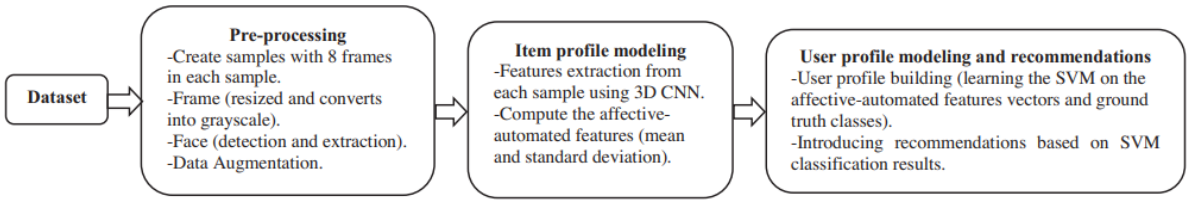


Figure 2.7: General flow of the proposed recommendation system

3D CNN extracts spatiotemporal features from video clips, capturing changes in facial expressions. Then, outputs from the third fully connected (FC) layer are used to compute AAF (mean and standard deviation values for emotional dimensions). These AAF values are stored in the item profile. For user profile modelling, a Support Vector Machines (SVM) classifier is trained on user preferences (items liked or disliked). Preferences are represented by the AAF vectors and their corresponding classes (positive or negative). The trained SVM builds a user profile, enabling personalized recommendations. User profiles generated by the SVM are matched with item profiles to generate top N recommendations.

The performance of the model was evaluated. The model (3D CNN - SVM) achieved a f-measure of 70%, outperforming the benchmark SVM model, which achieved 64%.

Yarahmadi Gharaei et al. (2021) proposed a content-based recommendation system using CNN. The main goal of this system is to solve the cold-start problem for new items with limited data. Instead of relying on attributes of products such as category or brand, CNN in this model extracts features directly from product images. CNN has 13 layers with a kernel size of 4x4. It extracts features from the 12th layer's output, creating feature vectors for each image. Then, the system calculates cosine similarity between extracted feature vectors. The system identifies the top N most similar items to the user's purchase history and generates the recommendations.

The performance of the model was evaluated. It achieved a precision of 73.7%. According to Yarahmadi Gharaei et al. (2021), users appreciated unexpected but relevant recommendations due to novel feature extraction from product images. The authors suggest comparing the proposed system with non-deep learning-based content-based recommendation methods.

2.3.2 Neural collaborative filtering (NCF)

In the recommendation system by Bhagat & Chatur (2023), Neural collaborative filtering (NCF) is utilized to generate recommendations with the help of Matrix factorization (MF).

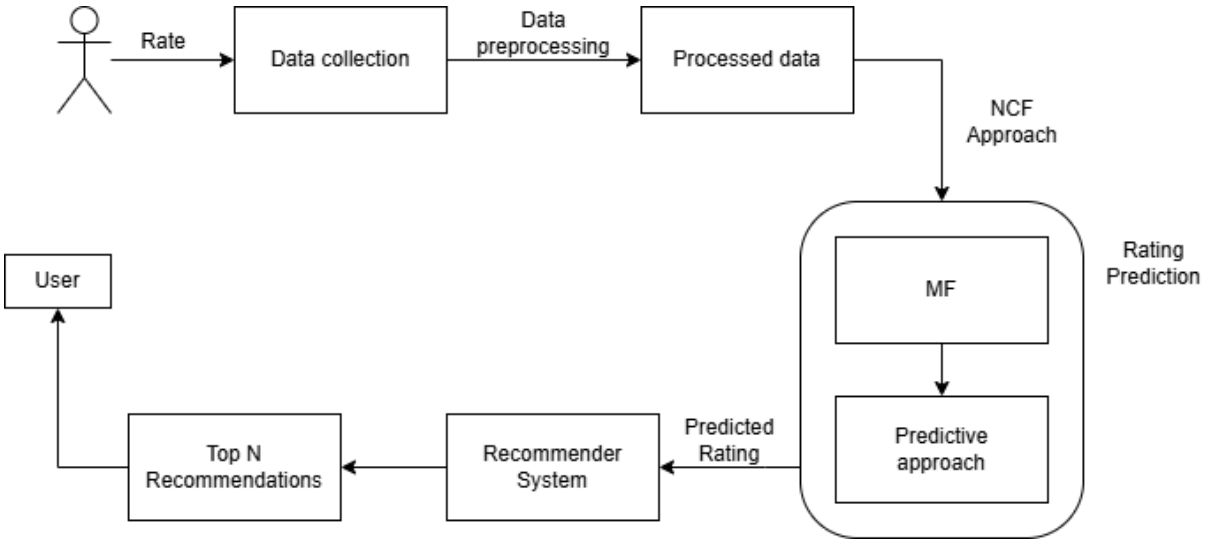


Figure 2.8: Overview of the proposed recommendation system

User-based collaborative filtering is employed to calculate similarity scores between users based on their preferences. The algorithm computes the similarity between users' ratings using a cosine similarity metric, which helps in identifying users with similar tastes (Bhagat & Chatur, 2023). Matrix factorization (MF) in this proposed system represents users or objects as vectors of latent attributes within a shared feature space. Then, user-item interactions are captured through the inner product of the user-item vectors. On the other hand, Neural collaborative filtering (NCF) employs a Multi-Layer Perceptron (MLP) to effectively learn user-item interactions. The figure below shows the proposed system approach.

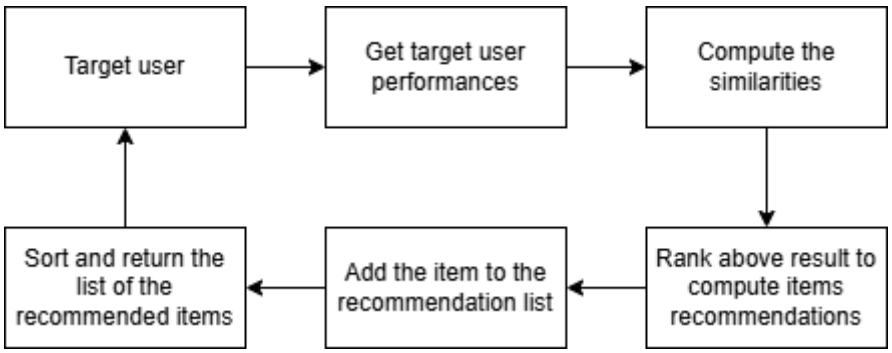


Figure 2.9: Proposed system approach

Bhagat & Chatur (2023) conducted a performance analysis using the Similarity Index as a metric to evaluate the effectiveness of the recommendations. The proposed NCF model (MF + MLP) achieved an accuracy of 0.97.

In another recommendation system by Qalbyassalam et al. (2022), an NCF model with implicit rating input is built. Sentiment analysis was conducted to get the implicit rating by calculating the sentiment score for each review. This analysis utilizes a lexicon-based approach, specifically employing the InSet (Indonesia Sentiment Lexicon). InSet consists of a word column and a corresponding weight column that assigns a score to each word. The process begins by tokenizing the review text into individual words. Each word's polarity is then determined by referencing its score in the lexicon. In the next step, the neural collaborative filtering (NCF) model is applied and trained on the implicit input.

To perform model evaluation, Qalbyassalam et al. (2022) built a benchmark model (NCF + explicit rating). The explicit rating here refers to the direct rating score provided by users for the product. The figure shows the result.

	NCF+*Explicit Rate*	NCF+*Implicit Rate*
MSE	0.49944	**0.2431**
RMSE	0.8033	**0.4931**

Figure 2.10: Performance evaluation

The combination of NCF and implicit ratings proves to be an effective model, achieving an RMSE of 0.4931. An RMSE value within the range of 0.2 to 0.5 suggests that the model can predict the data with a high degree of accuracy.

This research shows that using implicit ratings derived from sentiment scores for each review can outperform models that rely solely on direct user-provided ratings.

A deep learning hybrid recommendation system was proposed by Ibrahim et al. (2023). Compared to the previous two models by Bhagat & Chatur (2023) and Qalbyassalam et al. (2022) which rely only on NCF, this hybrid recommendation system utilizes both NCF model and Bidirectional Long Short-Term Memory (BiLSTM). BiLSTM is part of the Hierarchical User Attention and Hierarchical Product Attention (HUAPA) module, which captures user preferences and product characteristics through a hierarchical attention mechanism. BiLSTM helps in understanding the context of user reviews by processing the information in both forward and backward directions. Meanwhile, NCF focuses on modeling explicit interactions between users and products. It analyzes user-product interactions, enhancing the prediction of user preferences based on their historical data.

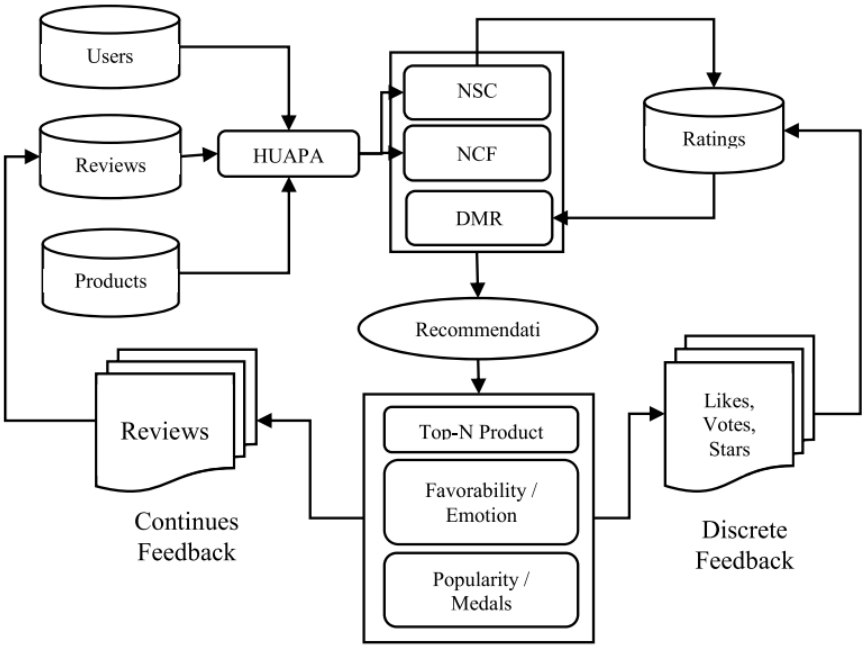


Figure 2.11: Overview of the hybrid system

Similar to the recommendation model by Qalbyassalam et al. (2022), this proposed system also conducted sentiment analysis. In this recommendation system, the Neural Sentiment Classifier (NSC) is utilized to extract semantic information from user reviews, classifying sentiments associated with products. It incorporates both user preferences and product characteristics to provide a comprehensive understanding of user sentiments.

An experiment is conducted to evaluate the performance of this model. It achieves an accuracy of 0.646 which is slightly higher than the benchmark models, including SVM (0.557) and CNN (0.585).

Cindhamani et al. (2024) proposed a hybrid recommendation system using NCF with Association Rule Mining (ARM). ARM helps to discover interesting relationships between variables in large datasets. It can identify patterns in user behavior and preferences more accurately, which can inform better recommendations by understanding how different movies relate to one another based on user ratings and interactions.

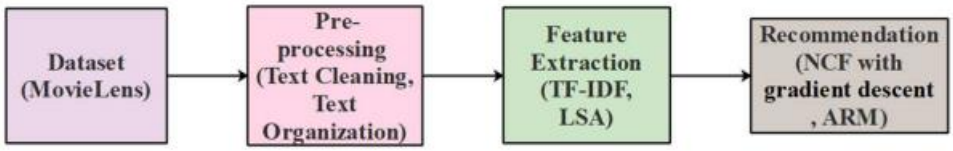


Figure 2.12: Overview of the proposed system

The figure above shows that NCF is optimized with Gradient Descent. This optimization technique is used within the NCF framework to minimize prediction errors during training, ensuring that the model learns effectively from the data.

To evaluate the performance of this model, an experiment is conducted. This model is able to achieve an accuracy of 0.99, slightly higher than the benchmark models including CNN (0.97) and KNN (93.87).

2.3.3 Convolutional Neural Network (CNN)

Putri et al. (2022) proposed a recommendation system using a specifically designed CNN to predict ratings accurately. The Input layer accepts two types of data namely user information and item information. In the embedding layer, User and item IDs are one-hot encoded into sparse binary vectors. These vectors are then projected into lower-dimensional continuous embeddings, which act as latent feature representations for users and items. The convolutional layer performs feature extraction by combining embeddings of users and items, which detects contextual patterns and relationships between users and items. The dropout layer prevents overfitting by ensuring the model doesn't overly rely on specific features, improving generalization. The dense layer combines the learned patterns from previous layers to create a comprehensive representation for prediction. The output layer generates the final predicted rating score for the user-item pair.

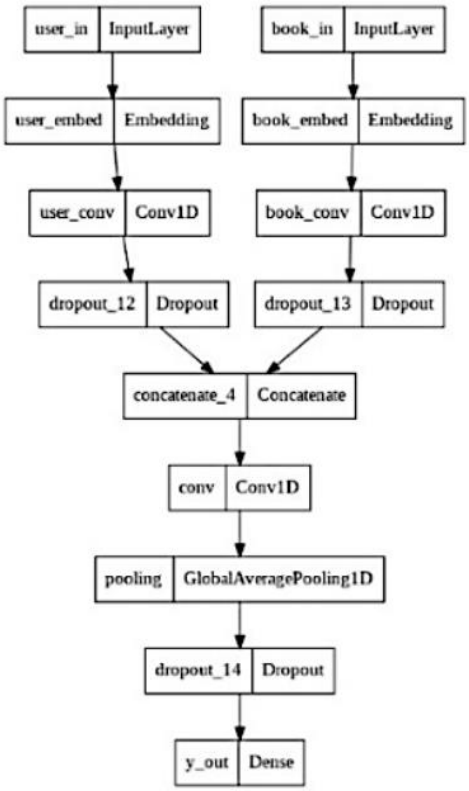


Figure 2.13: Structure of proposed CNN model

The model's performance was evaluated by comparing it with the Singular Value Decomposition (SVD) algorithm. The proposed model achieves an MAE of 1.345 which is slightly higher than SVD (1.083). Putri et al. (2022) explained that noise removal in SVD helps improve performance but may inadvertently discard useful information. The authors noted that the model may require improved overfitting handling and could involve hybrid models to enhance performance.

Ge (2022) proposed a personalized recommendation system using a model called Local Similarity Prediction CNN (LSPCNN). The goal of this CNN-based model is to address challenges like data sparsity and cold start. According to Ge (2022), existing methods such as basic CNN models and collaborative filtering cannot effectively handle these issues.

The proposed LSPCNN model adds a regulation layer to enhance the convolution layer by dynamically adjusting the local similarity of features extracted. It focuses on localizing user interests by constructing an item-scoring matrix for individual users.

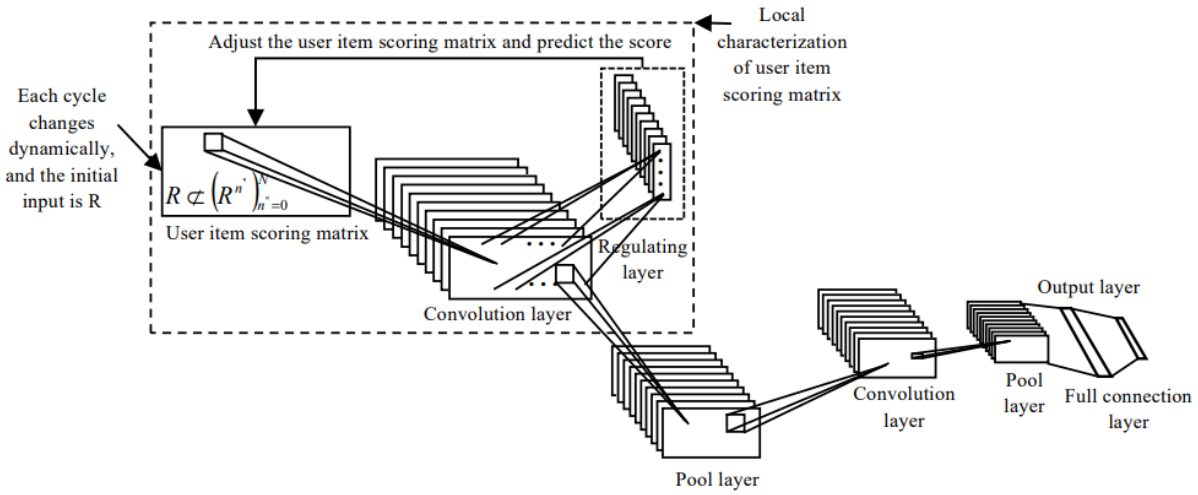


Figure 2.14: General structure of LSPCNN

Input data (user-item scoring matrix) is processed through convolution layers to extract user preferences. A two-dimensional matrix is created based on the highest similarity of user and item features. To optimise the LSPCNN model, it learns user preference features through multiple iterations of training. The final prediction score is computed as an average of predictions across multiple cycles.

The model was evaluated against the basic CNN model to compare the performance. This model achieves an MAE of 0.77, which is 0.05 lower than that of the basic CNN model (0.82).

Hong et al. (2024) proposed a review-based recommender system using the Outer Product on CNN (ROP-CNN) model, which effectively extracts and integrates semantic features from reviews. This model consists of three fundamental components. First, the interaction encoder generates an interaction map utilizing outer products to capture latent interactions between users and items, upon which a 2-D CNN is applied to produce interaction vectors. Secondly, the review encoder is designed to extract linguistic expressions from reviews. In this context, a one-dimensional CNN (1-D CNN), which is widely recognized in the realm of natural language processing, is employed to effectively extract the high-level semantic elements inherent in the reviews. Finally, the rating prediction stage utilizes a Multi-Layer Perceptron (MLP) to predict the feature vectors derived from both the interaction encoder and the review encoder, allowing for the concatenation of these vectors to learn the non-linear relationships between user-item interactions and the semantic aspects of reviews.

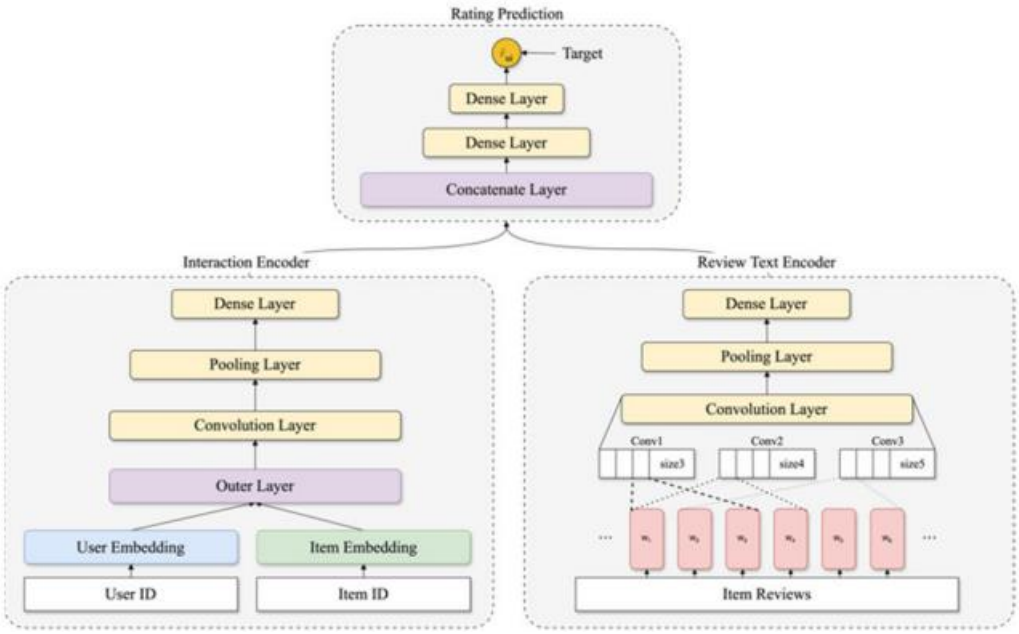


Figure 2.15: Overview of ROP-CNN architecture

Both element-wise operations and concatenation are included in the outer product. This allows the outer product to effectively capture interactions not only among elements within the embedding dimension but also across different dimensions.

The extracted user-item interaction vector is integrated with the user's latent preference vector. Then, a Multi-Layer Perceptron (MLP) is implemented to capture the nonlinear relationship between the two sets of information and generate the rating prediction. The predicted rating is then normalized to guarantee that it remains within the minimum and maximum boundaries established by the existing ratings (usually 0-5).

To assess the performance of the proposed ROP-CNN model, Hong et al. (2024) conducted a comparison experiment between the proposed model and benchmark models. The proposed ROP-CNN model achieves an MAE of 0.376, which is significantly lower than the benchmark models including DeepHCF (0.635) and NeuMF (0.814).

The ROP-CNN model utilizes outer products to capture interactions across different dimensions, allowing the model to extract richer interaction data. Similar to the proposed recommendation system by Qalbyassalam et al. (2022) and Ibrahim et al. (2023), this model also incorporates reviews to enhance the semantic understanding of user preferences and item characteristics. This approach improves recommendation performance by utilizing both user-item interaction data and the semantic features derived from reviews.

A CNN-based hybrid recommendation system was proposed by Tran et al. (2020). In the model, CNN is combined with neural collaborative filtering (NCF) to form the StackedMF framework. NCF helps capture user-item interactions through element-wise products of user and item latent vectors. Meanwhile, CNN processes a 2-D interaction feature map to extract deeper features. To create the 2-D interaction feature map, the methodology employs a stacking operation on the latent vectors of users and items. This approach helps to reduce noise compared to other methods, such as the outer products technique used in the model proposed by Hong et al. (2024).

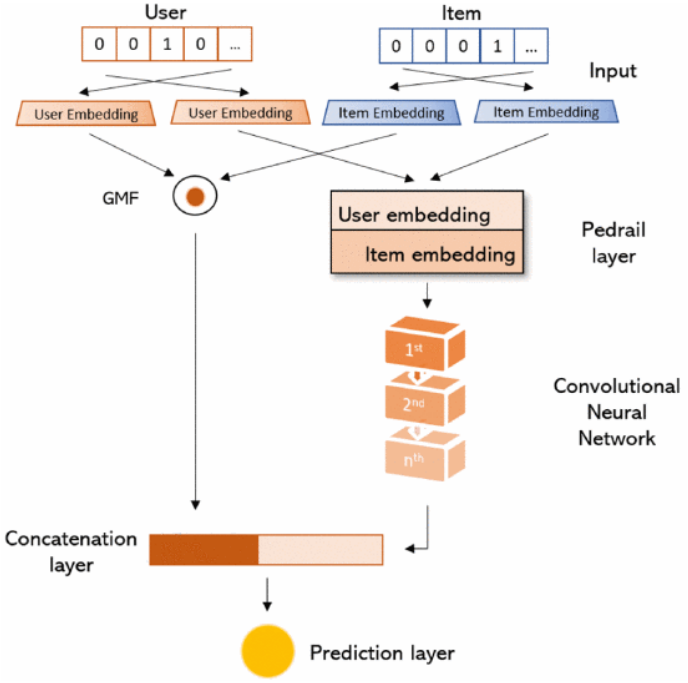


Figure 2.16: General architecture of the proposed system

The outputs from both the GMF and CNN layers are concatenated before passing through a loss function (Binary Cross Entropy with logits). This fusion allows the model to leverage both linear (from MF) and non-linear (from CNN) aspects of user-item interactions effectively.

The proposed system was evaluated against several benchmarks. It achieves an NDCG of 0.5 which is slightly higher than the benchmark models including NeuMF (0.42) and GMF (0.415).

This model also incorporated implicit feedback data, similar to the approaches used by Hong et al. (2024), Qalbyassalam et al. (2022) and Ibrahim et al. (2023). The difference is that this model gathers implicit data from user interactions like clicks or views rather than reviews.

2.3.4 Long Short Term Memory (LSTM) Algorithm

Wang (2024) proposed a recommendation system using the Long Short Term Memory (LSTM) algorithm for rural areas. LSTM is a type of recurrent neural network designed to effectively manage long-term dependencies in data. It employs a gating mechanism to control the flow of information, making it suitable for handling the temporal and nonlinear characteristics of rural e-commerce data. The LSTM algorithm is primarily structured with three layers: the input layer, the hidden layer, and the output layer. The state of an LSTM unit includes a memory cell and a hidden state. Once the input for each network computation is established, the predicted output value is computed through the forward network process.

According to Wang (2024), the use of the Long Short-Term Memory (LSTM) algorithm in rural e-commerce is gaining significant traction. Rural e-commerce data presents complex characteristics, including temporal regularity and unpredictable behavioral patterns. LSTM networks are adept at addressing these complexities by effectively capturing the time series and nonlinear properties of the data, thereby enhancing user behavior analysis and product trend modeling. The current capabilities of the LSTM algorithm include predicting potential future purchasing behaviors of users by examining their past interactions and product features.

To compare the effectiveness of the Long Short-Term Memory (LSTM) algorithm with the Traditional Collaborative Filtering (CF) algorithm, a performance evaluation experiment was conducted by Wang (2024). The LSTM algorithm achieved an accuracy of 75%, 10% higher than CF (65%).

Dey et al. (2022) proposed a recommendation system for online learners using a hybrid deep learning model. It incorporates both LSTM and Gated Recurrent Unit (GRU) layers. LSTM provides a memory architecture that stores and utilizes user behavior data, discarding irrelevant information. Meanwhile, GRU Extracts relevant user profile features, improving data retention from previous analyses. The system generates recommendations based on stored probabilities in LSTM memory units. Then, it combines outputs from LSTM and GRU layers to provide final recommendations. Essentially, the model combines LSTM's ability to encode user behavior with GRU's feature extraction capabilities to enhance recommendation accuracy.

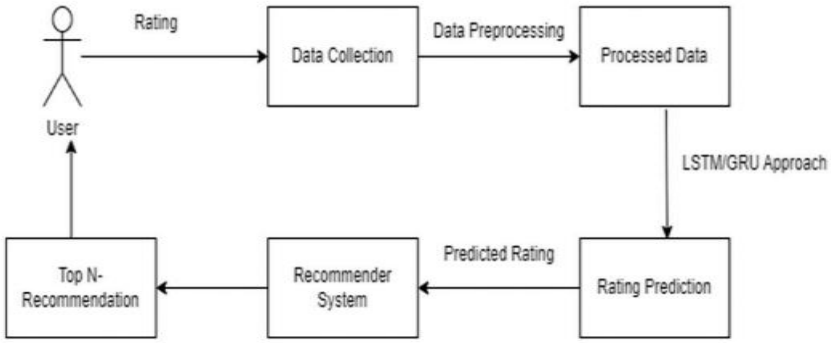


Figure 2.17: Overview of the proposed system

The performance of the proposed system was evaluated. It achieved an MAE of 0.819 and RMSE of 1.031.

According to Dey et al. (2022), LSTM effectively handled the gradient disappearance issue, which often affects deep learning models. Using a hybrid LSTM-GRU architecture can achieve optimal feature extraction.

In 2023, Li & Zhang proposed a hybrid recommendation system that integrates Bidirectional Long Short-Term Memory (BiLSTM) and Neural Collaborative Filtering (NCF) algorithms. The goal is to utilize the strengths of Bi-LSTM in sequential data analysis and NCF in collaborative filtering. Bi-LSTM processes sequential user interaction data to capture patterns and extract meaningful embeddings that represent user preferences and behavior. NCF utilizes matrix factorization and deep learning layers to learn latent features from the interaction matrix. Then, the embeddings from both components are concatenated to form a single comprehensive feature vector. The combined feature vector is fed into a fully connected feedforward neural network. This network learns to weigh the relative contributions of sequential and collaborative filtering features to optimize the final recommendations. The output of the feedforward network is passed through a prediction layer, producing the probability of a user interacting with a particular item.

The performance of the proposed model was evaluated with the benchmark models. The model achieves an F1 score of 0.93 which is higher than NCF (0.91) and CF (0.87).

The proposed recommendation model by Ibrahim et al. (2023) also utilized BiLSTM and NCF, but it used BiLSTM in the HUAPA module, which captures user preferences and product

characteristics through a hierarchical attention mechanism. In this model by Li & Zhang (2023), BiLSTM is utilized to generate feature embeddings that represent the temporal patterns in user behavior. The functionality of BiLSTM in both models is different but the goal is the same (to capture user preferences).

In 2020, W. Wang et al. proposed a hybrid recommendation model that integrates Long Short-Term Memory (LSTM) and Convolutional Neural Networks (CNN). This model aims to improve recommendation accuracy by capturing context dependency in user ratings with LSTM and extracting locally relevant features from items using CNN. The model utilizes user information, rating data, and item data as inputs to extract user features and item features. It then computes predicted ratings and generates recommendations based on these features.

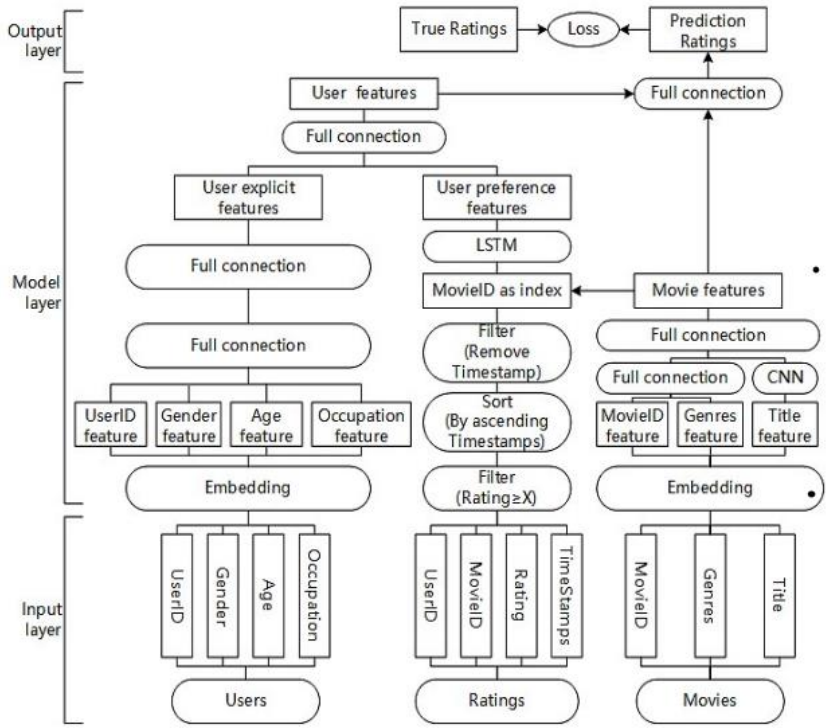


Figure 2.18: Structure of the proposed system

Embedded vectors are created for user information. These vectors are passed through a two-layer fully connected network to compute explicit user features. Meanwhile, item information was processed through CNN layers and fully connected networks, forming the item feature vector. To form user preference features, item feature vectors corresponding to user-rated items are input into a two-layer LSTM to capture sequential patterns, yielding user preference

features. This output is combined with user features to form the comprehensive user feature vector. Then, this value is combined with the item feature vector, producing the rating prediction.

The performance of the proposed model is evaluated. The model achieved an MAE of 0.751 and an MSE of 0.876.

2.4 Comparison between existing techniques

Table 2.1 Comparison among existing techniques

	Content-based filtering (CBF)	**Neural collaborative filtering (NCF)**	**Convolutional Neural Network (CNN)**	**Long Short Term Memory (LSTM)**
Performance Metrics used	- Accuracy - Precision - F-measure	- Accuracy - RMSE - MSE	- MAE - NDCG	- Accuracy - MAE - RMSE - MSE - F-measure
Advantages	- No cold start problem for new items, recommendations are based on similarity, not historical data (ratings, views).	- Captures complex user-item interactions through deep learning. - Can incorporate both explicit feedback (ratings)	- Efficient in extracting hierarchical patterns, making them suitable for processing images, text, or other structured data.	- Excel at handling sequential data, which is useful if recommendations are based on time-series data or sequential user behavior (e.g., viewing history).

	- Simple and easy to implement.	and implicit feedback (reviews, clicks, views).	- Minimum cold start issue since CNN can recommend new items by analyzing item visual content without requiring interaction data.	- Remember past interactions over long periods, providing more context to user preferences.
Disadvantages	- Only recommends items similar to what the user has already interacted with, potentially leading to a lack of diversity.	- Need a significant amount of data for effective training, and performance can degrade with smaller datasets.	- Suffer from data sparsity if item features (like images or text) are limited, as CNN performance depends on rich, high-quality input data. - Computationally intensive, training CNN typically requires significant computational resources.	- Complex to implement and require precise tuning of hyperparameters. - Scale poorly on large datasets due to their sequential nature and the need for substantial computational power.

2.5 Summary

In summary, this chapter reviewed and analyzed existing academic research on recommendation systems conducted between 2020 and 2024, providing valuable insights into existing methodologies and their strengths and limitations. These findings have laid a strong foundation for the development of the proposed system, offering essential concepts to guide the next chapter on Methodology. A hybrid approach will be proposed due to the clear advantages identified throughout this literature review. Multiple academic research in this literature review have employed hybrid systems that combines multiple techniques. Liliana et al. (2024) and Singla et al. (2020) also suggested that combining content-based filtering (CBF) and collaborative filtering (CF) to create a hybrid system that combines the strengths of both approaches while addressing their respective limitations.

Chapter 3: Methodology

3.1 Introduction

This chapter will discuss the methodology of the project. It outlines the systematic approach taken to achieve the objectives of the project. This will guide the research process to ensure that the research is systematic and relevant. The general architecture and process flow of the proposed system will be explained with the help of a flowchart diagram. The final part of this chapter will present a basic user interface of the proposed system.

3.2 Research Methodology

This research involves five key phases, which include data collection, data preprocessing, model development, model evaluation, and web application development.

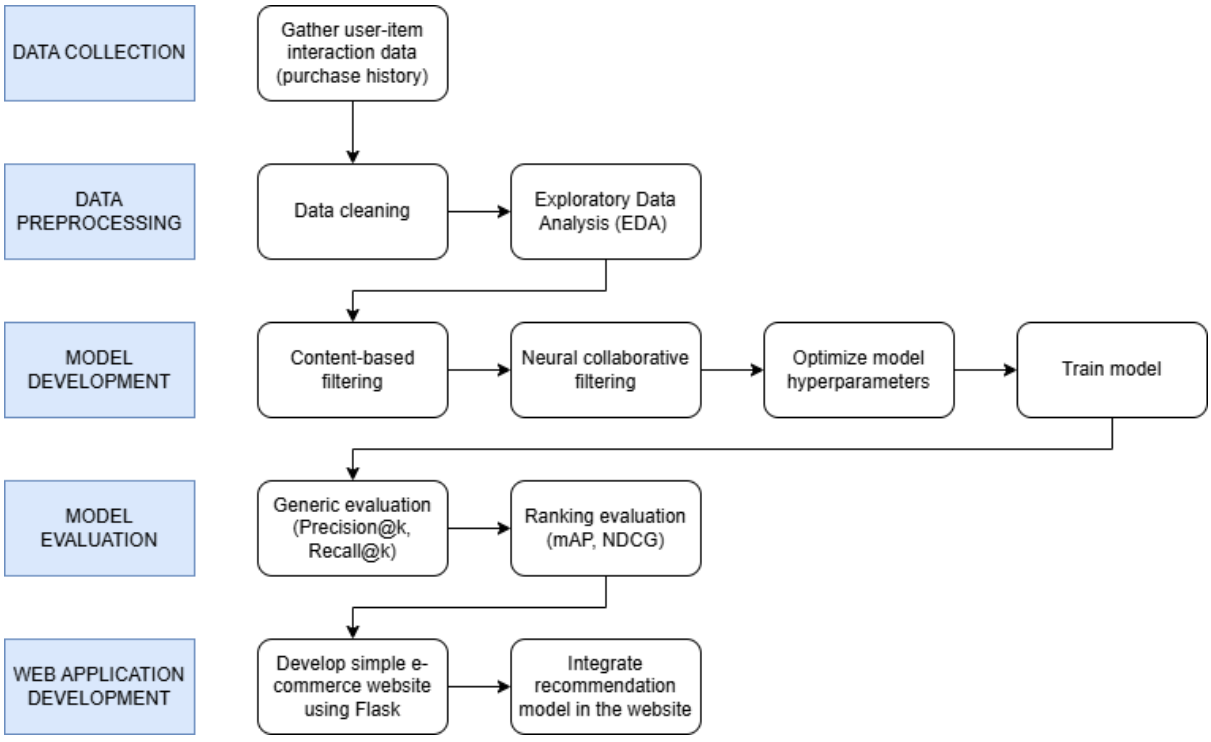


Figure 3.1: Research Methodology

3.2.1 Phase 1: Data Collection

There are primarily three categories of data to gather. The first category is user data. This includes the user's historical interaction data such as purchase history and ratings. Understanding user behavior is essential for tailoring recommendations. The second category is product data. This includes detailed information about products such as categories and brands. This data helps in content-based filtering approaches and enhances the context for recommendations. The third category is interaction data. This includes user ID, product ID, and ratings. This interaction data helps capture user interactions with products, which is crucial for collaborative filtering methods that rely on user behavior patterns.

There are quite a few sales datasets provided by Amazon which can be used to build product recommendation systems. These datasets can be commonly found online at Kaggle and GitHub. The figure below shows electronic product sales by Amazon on kaggle.com.

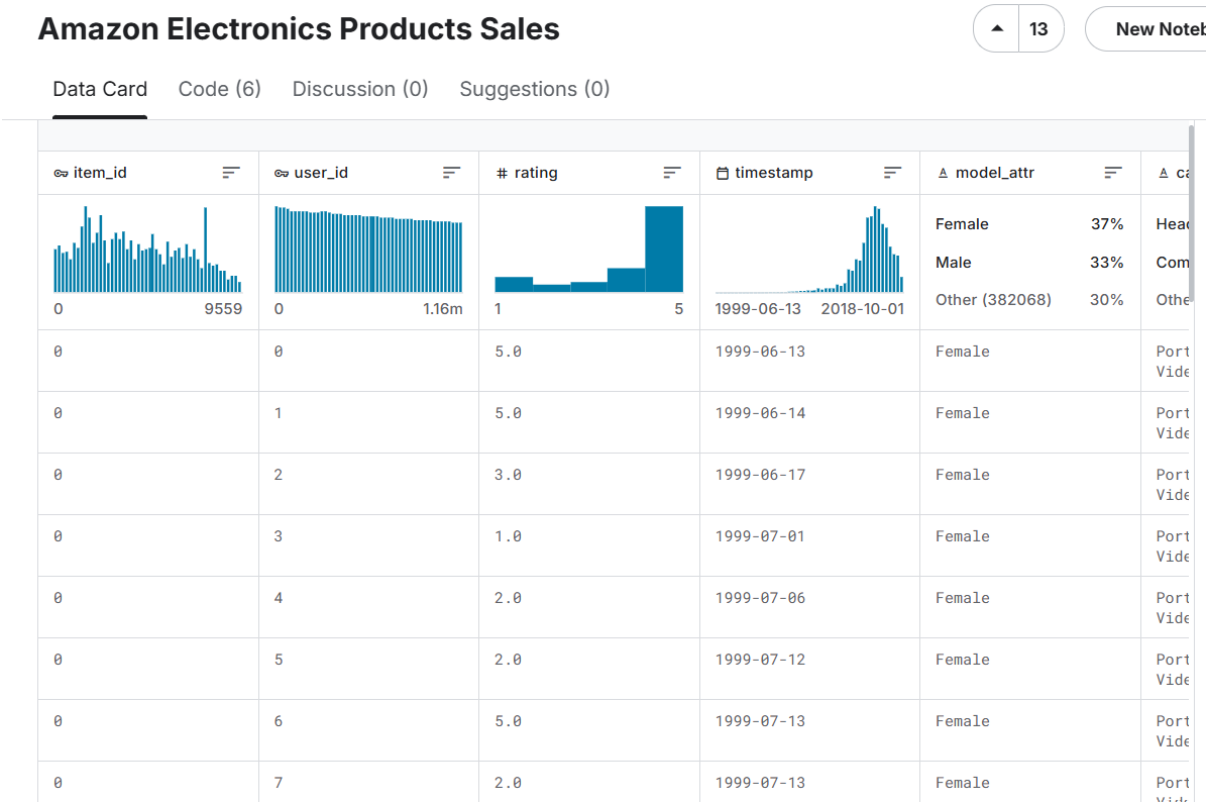
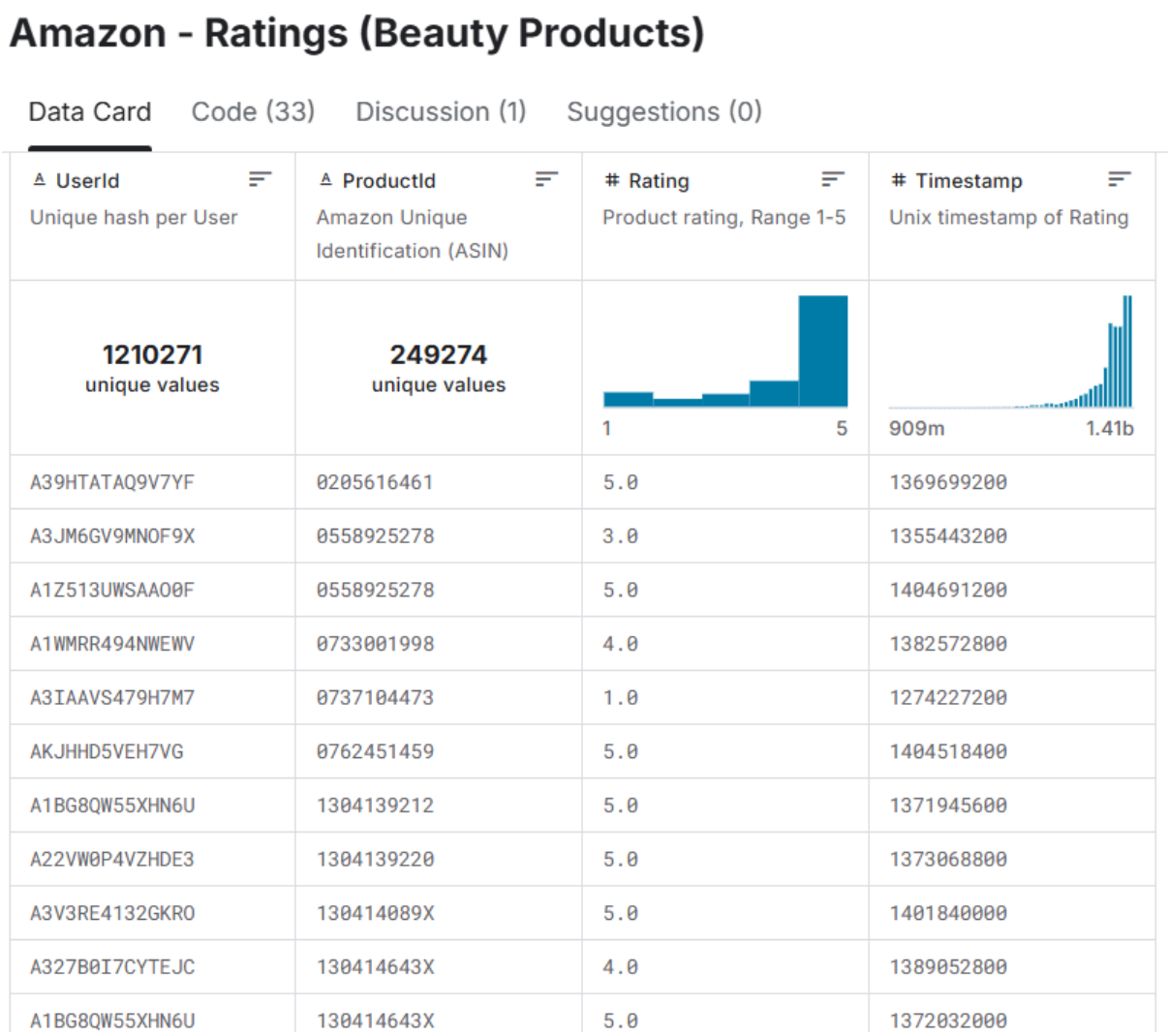


Figure 3.2: Amazon electronic product sales (Sanket Dinesh Khadse, 2021)

Here are some other datasets that can be used for building recommendation system models.



Amazon - Ratings (Beauty Products)

Data Card | Code (33) | Discussion (1) | Suggestions (0)

UserId Unique hash per User	ProductId Amazon Unique Identification (ASIN)	# Rating Product rating, Range 1-5	# Timestamp Unix timestamp of Rating
1210271 unique values	**249274** unique values	1 — 5	909m — 1.41b
A39HTATAQ9V7YF	0205616461	5.0	1369699200
A3JM6GV9MNOF9X	0558925278	3.0	1355443200
A1Z513UWSAAO0F	0558925278	5.0	1404691200
A1WMRR494NWEWV	0733001998	4.0	1382572800
A3IAAVS479H7M7	0737104473	1.0	1274227200
AKJHHD5VEH7VG	0762451459	5.0	1404518400
A1BG8QW55XHN6U	1304139212	5.0	1371945600
A22VW0P4VZHDE3	1304139220	5.0	1373068800
A3V3RE4132GKRO	130414089X	5.0	1401840000
A327B0I7CYTEJC	130414643X	4.0	1389052800
A1BG8QW55XHN6U	130414643X	5.0	1372032000

Figure 3.3: Amazon beauty products (Amazon - Ratings (Beauty Products), n.d.)

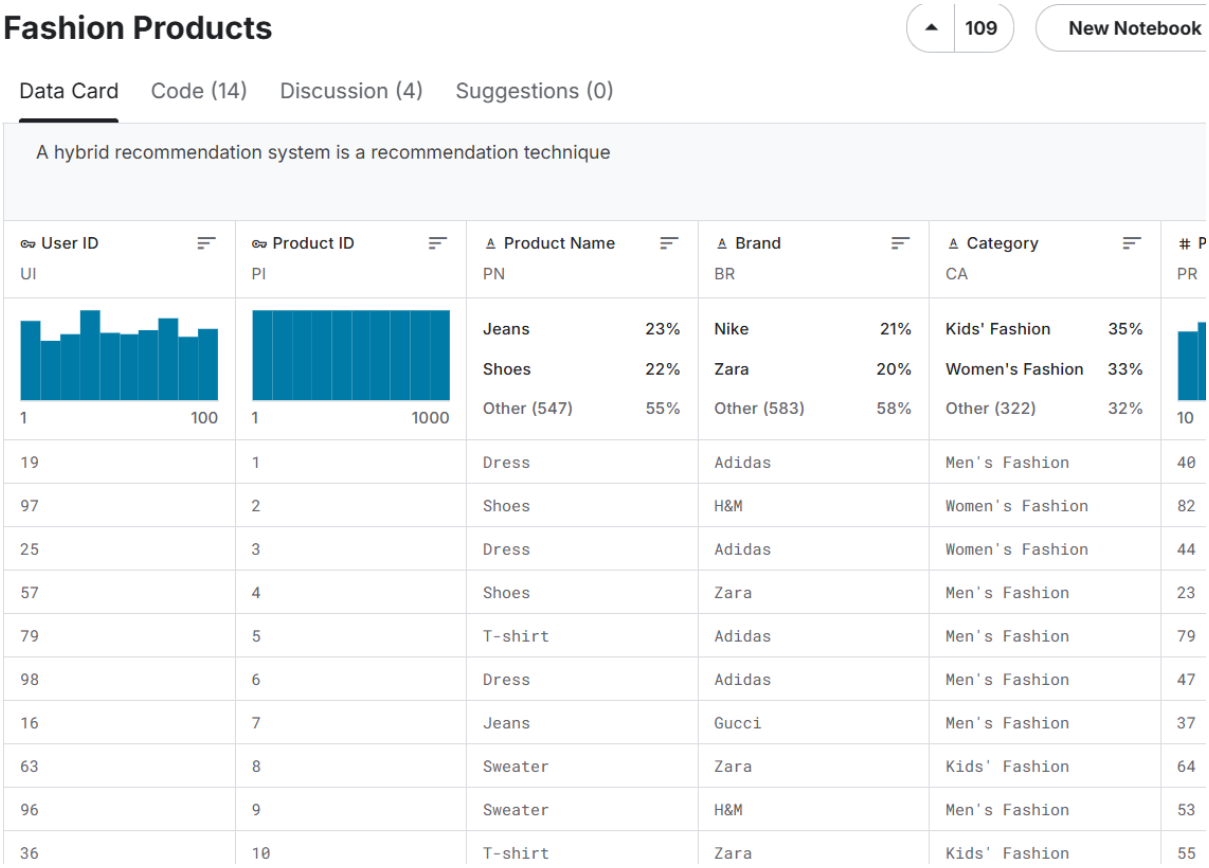


Figure 3.4: Fashion products (Fashion Products, n.d.)

3.2.2 Phase 2: Data Preprocessing

The first step is removing unnecessary columns. This simplifies the dataset by keeping only the features relevant to the models which prevent noisy data from interfering with the learning. The second step is data cleaning. This includes removing duplicates and handling missing values in the datasets. Techniques such as imputation or removal of incomplete records are applied. Other than that, it is also crucial to conduct Exploratory Data Analysis (EDA) to understand the distribution of data. This helps to identify trends and visualize relationships between features, which is important to gain insights into user behavior and product characteristics.

3.2.3 Phase 3: Model Development

The proposed system implements a **hybrid** recommendation model that integrates content-based filtering (CBF) and neural collaborative filtering (NCF). Figure 5 below shows the flowchart of the proposed system for the proposed system.

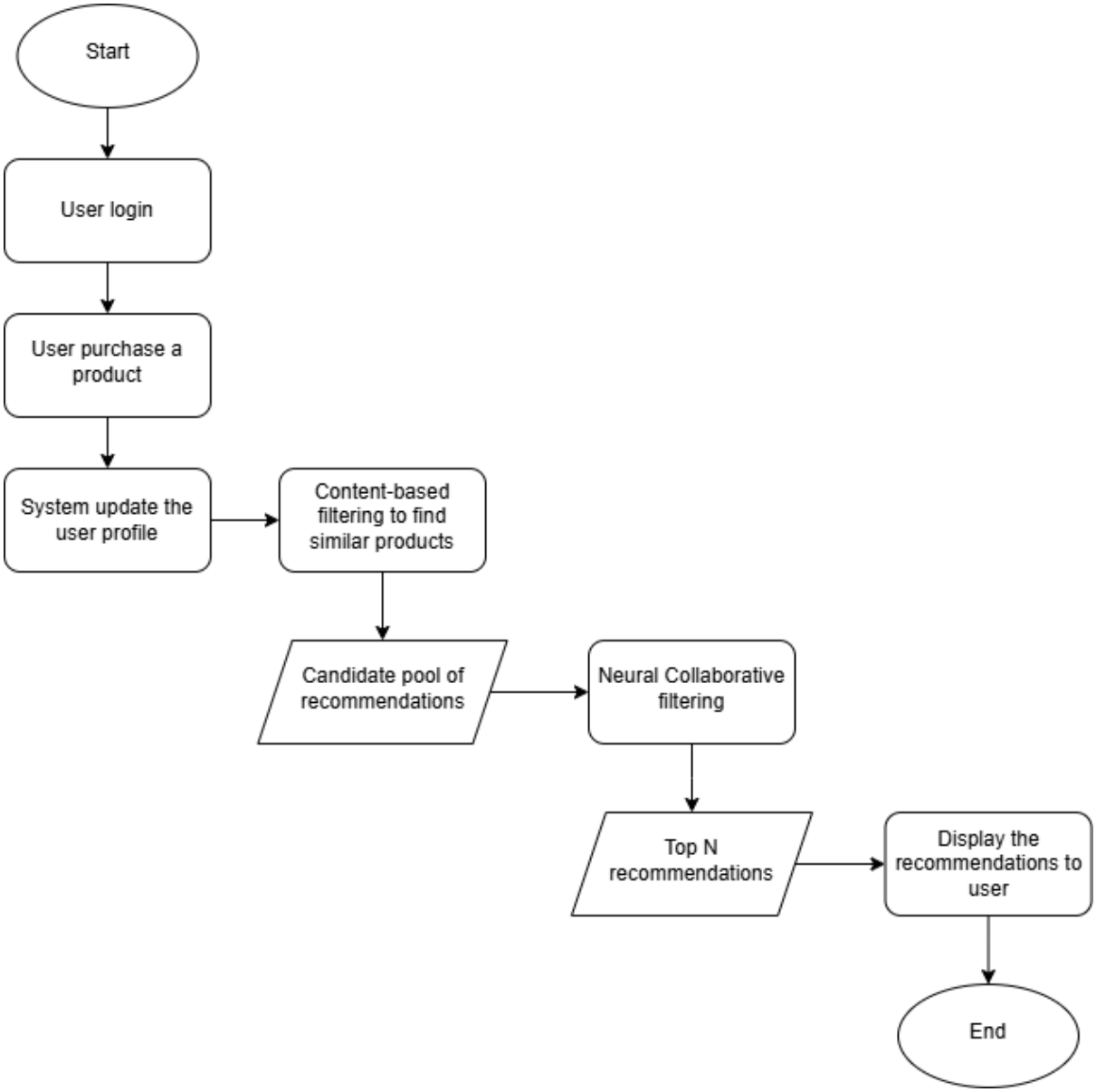


Figure 3.5: Flowchart of proposed system

When a user purchases an item, the system updates their profile to include the features of that item. Then, CBF calculates the similarity between the user's profile and the feature vectors of other items in the catalog. This will generate a list of recommended items that are most similar to those that the user has previously interacted with or expressed interest in. The list will act as a candidate pool which will be further ranked by NCF.

NCF can effectively capture the latent structures of user–item interactions. In the proposed model, NCF receives input from the candidate pool of recommendations produced by CBF. For each item in the candidate pool, a predicted rating is given by NCF. Then, the items in the candidate pool will be ranked by this predicted rating, which outputs the final top N recommendations for the user.

CBF was implemented before NCF for a few reasons. Firstly, it addresses the cold start problem for new products. For new products with little to no interaction history, CBF recommends them to users if they have similar metadata (categories, brand) that match the user's purchase history. This ensures the system can provide relevant suggestions even before sufficient user-item interaction data is available for NCF. The second reason is that it reduces data sparsity for NCF. NCF is designed to learn complex patterns from user-item interactions. This means NCF requires a rich dataset of user-item interactions to learn embeddings and make predictions effectively. By using CBF to generate initial user-item interactions (based on content similarity), the interaction matrix is enriched, thus reducing sparsity. The third reason is that it improves scalability. CBF is used to create a candidate pool of recommendations that align with a user's preferences, which will be used in NCF. NCF can then rank these candidates based on learned embeddings. Ranking a smaller candidate pool generated by CBF is computationally cheaper for NCF than training the model on the entire dataset. This means the proposed system will scale better as stored data expands.

The model's architecture was developed using TensorFlow, a widely used machine learning framework, enabling efficient implementation and training. Key hyperparameters, including the number of layers, learning rate, and embedding dimensions, were carefully optimized to enhance the model's performance. The dataset was split into two parts, with 80% allocated for training to allow the model to learn patterns and adjust its weights, and 20% reserved for validation to evaluate its ability to generalize to unseen data. This approach ensured a balanced and systematic process for building and testing the model effectively.

3.2.4 Phase 4: Model Evaluation

Model evaluation assesses the general accuracy and quality of the recommendations made by the system, and how well they are ranked. The evaluation metrics used include Precision@K, Recall@K, Mean Average Precision (mAP), and Normalized Discounted Cumulative Gain (NDCG).

$$\text{Precision@K} = \frac{\text{Number of relevant items in top K recommendations}}{K}$$

$$\text{Recall@K} = \frac{\text{Number of relevant items in top K recommendations}}{\text{Total number of relevant items}}$$

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i$$

$$\text{DCG@K} = \sum_{i=1}^{K} \frac{\text{relevance}_i}{\log_2(i+1)}, \quad \text{NDCG@K} = \frac{\text{DCG@K}}{\text{IDCG@K}}$$

Precision@K and Recall@K provide insights into the overall accuracy and completeness of the recommendation system. mAP ensures that the ranking of relevant items is taken into account, rewarding systems that position relevant items higher. NDCG evaluates the ranking quality, emphasizing the importance of placing the most relevant items early in the list. By combining these metrics, a comprehensive evaluation that balances relevance, ranking quality, and coverage is achieved.

The performance of the proposed hybrid model was evaluated against benchmark models, specifically Singular Value Decomposition (SVD) and Light Graph Convolution Network (LightGCN). LightGCN uses graph network to encode relationship between user-item while SVD applies matrix factorization to uncover latent factors that influence user preferences. Comparing these models provides a comprehensive evaluation of the hybrid approach's ability to generate accurate and personalized recommendations.

3.2.5 Phase 5: Web Application Development

To demonstrate the functionality of the proposed recommendation system, a prototype e-commerce website will be developed using Flask. Flask was chosen for its simplicity and modularity, which allow for quick prototyping and seamless integration with machine learning models. The website will serve as a user-facing platform, simulating a real-world e-commerce environment.

3.2.5.1 Requirement Analysis

3.2.5.1.1 Use Case Diagram

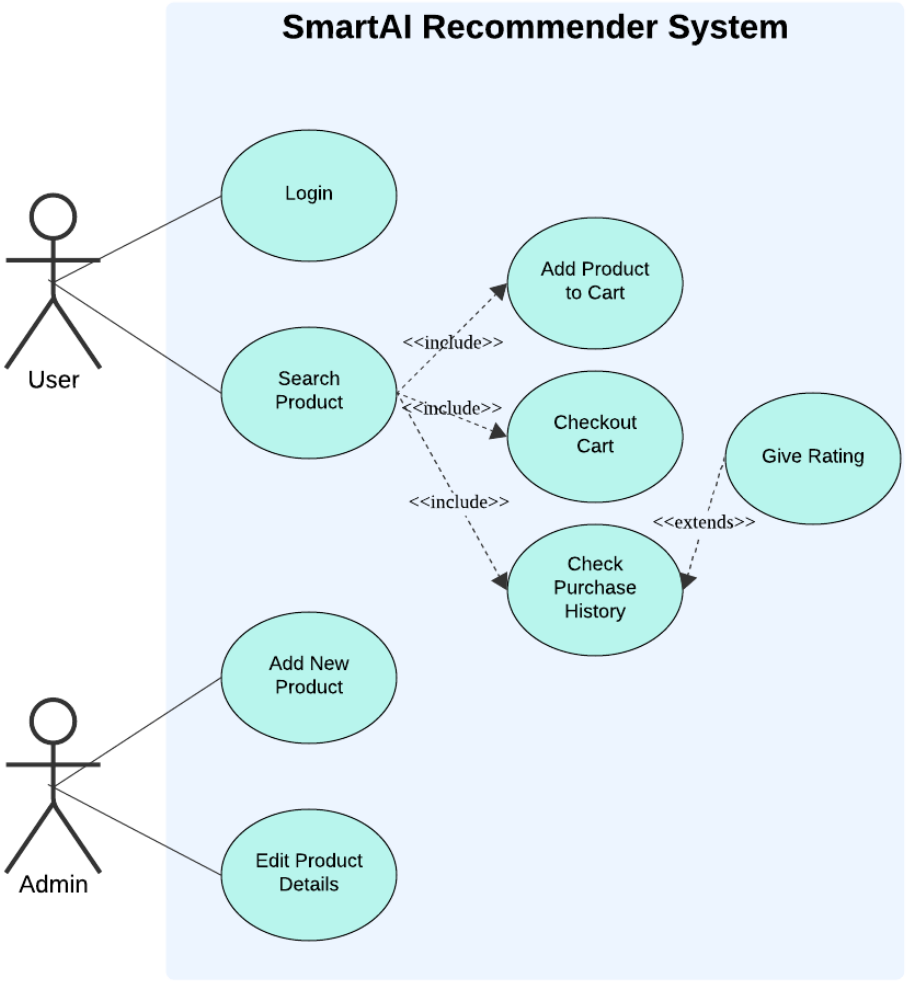


Figure 3.6: Use Case Diagram

The user can perform actions such as logging in, searching for products, adding products to the cart, checking out, giving ratings, and reviewing purchase history. Meanwhile, the admin can manage the product catalog by adding new products and editing product details.

3.2.5.1.2 Sequence Diagram

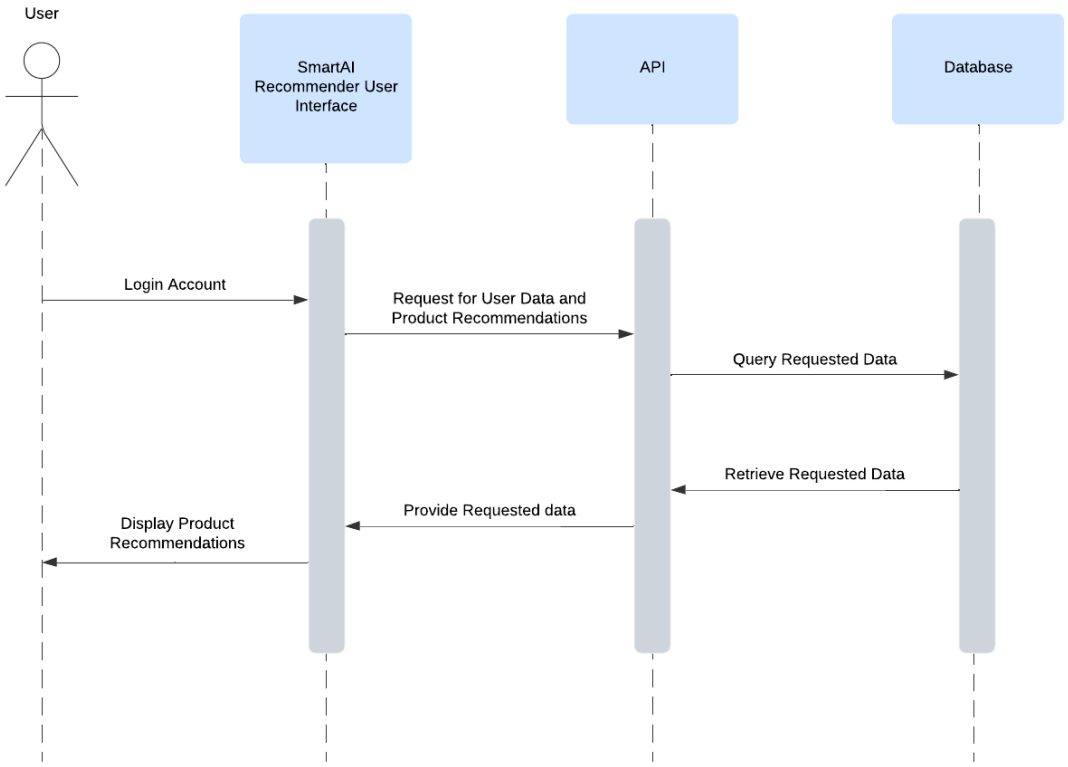


Figure 3.7: Sequence Diagram of User Logging into Account

The user logs in through the interface, which sends a request to the API for user data and recommendations. The API retrieves data from the database, processes it, and returns personalized recommendations to be displayed on the interface.

3.2.5.1.3 Activity Diagram

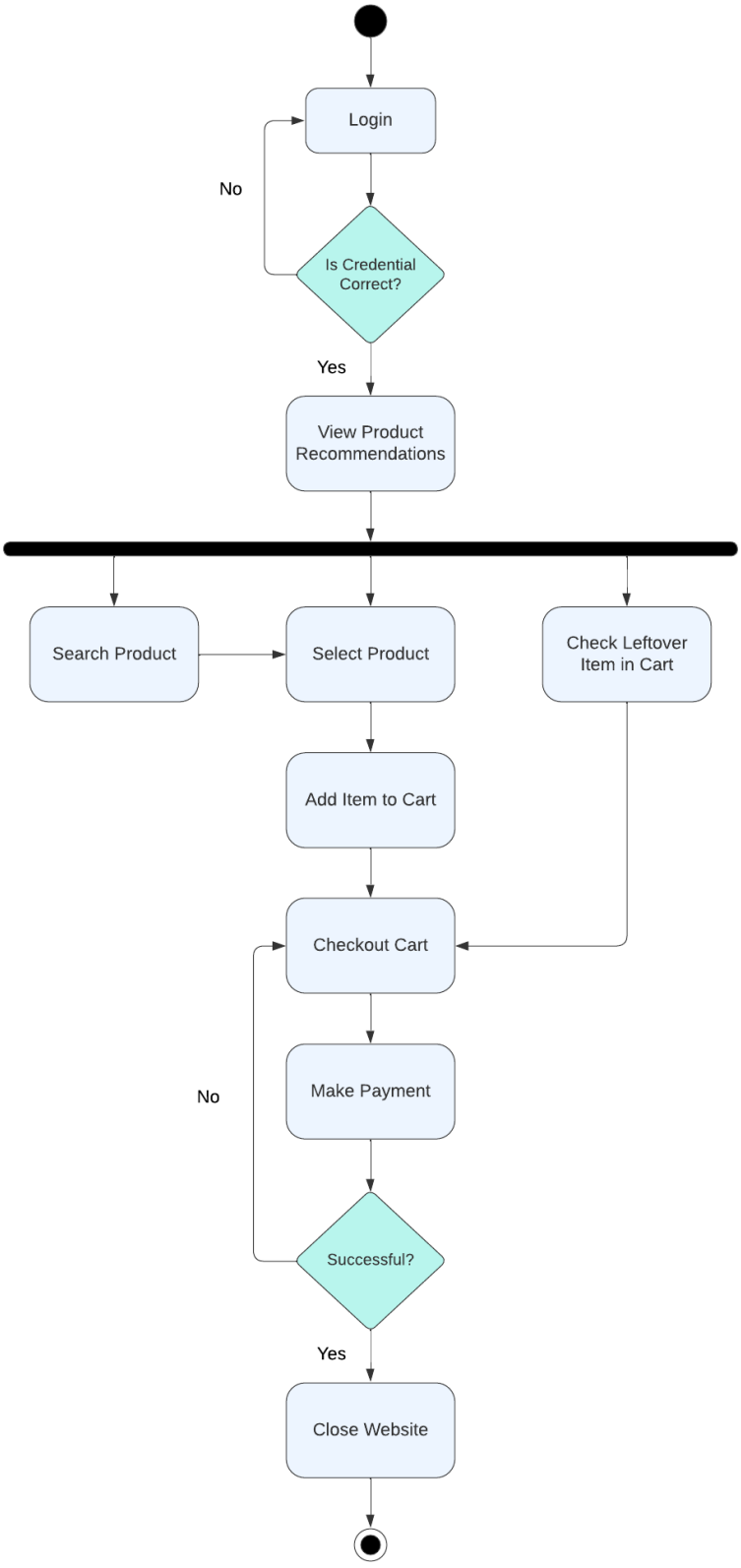


Figure 3.8 Activity Diagram

After login, the user can view product recommendations, search for products, check leftover items in the cart, or select and add items to the cart. The process proceeds to checkout and payment, and if the transaction is successful, the session concludes with the website closure.

3.2.5.1.4 Class Diagram

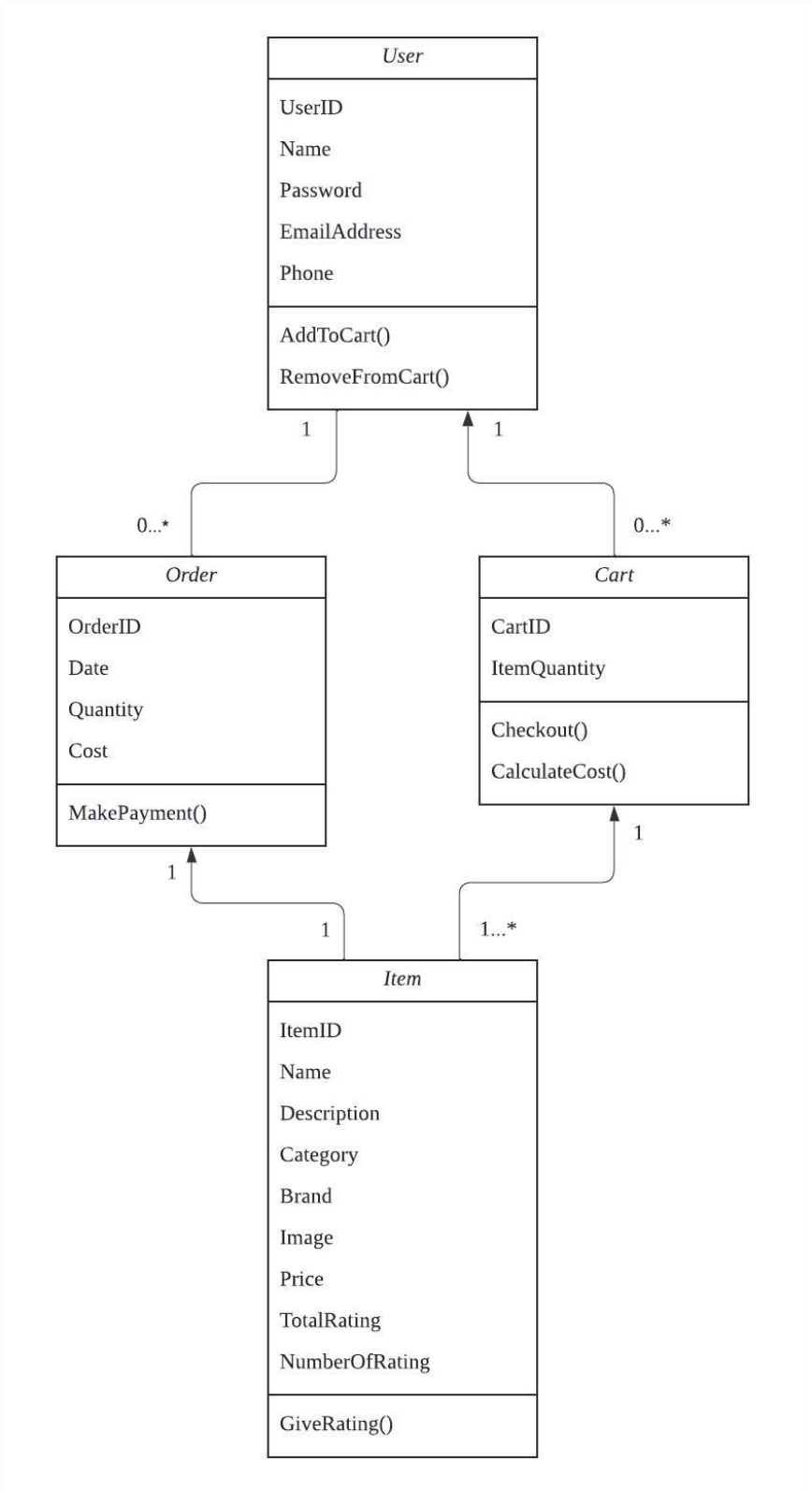


Figure 3.9 Class Diagram

The system includes classes for User, Order, Cart, and Item, with their respective attributes and methods. Users can interact with the cart (e.g., add or remove items), place orders, and provide

ratings for items, while the cart and order handle functionalities such as checkout, cost calculation, and payment processing.

3.2.5.1.5 Software requirements

Table 3.1 Software requirements

Software	Description
Python 3.8 or higher	Required for building the recommendation model and web application
Flask	A web framework to create the backend of the prototype e-commerce website
Jupyter Notebook	Environment for data preprocessing and testing the recommendation model
TensorFlow	Frameworks for implementing and training Neural Collaborative Filtering (NCF)
SQLite	A lightweight database for storing user and product data
Gunicorn	Web Server Gateway Interface (WSGI) to run and manage the Flask app

3.2.5.1.6 Hardware requirements

Table 3.2 Hardware requirements

Hardware component	Description
Intel i5 or equivalent CPU	Smooth coding and model testing
8 GB RAM	Handle processes
256 GB SSD	Store data files, project code, and results
CUDA-compatible GPU	Accelerate model training

3.2.5.2 Survey Analysis

A questionnaire survey was conducted to gather insights into user preferences and behaviors regarding online shopping and product recommendations. 32 respondents, consisting of students from UNIMAS, participated in the survey and provided valuable data on how users interact with e-commerce platforms, including their opinions on personalized product recommendations. The findings will help guide the development of an effective recommendation system tailored to user needs and preferences.

3.2.5.2.1 Analysis 1

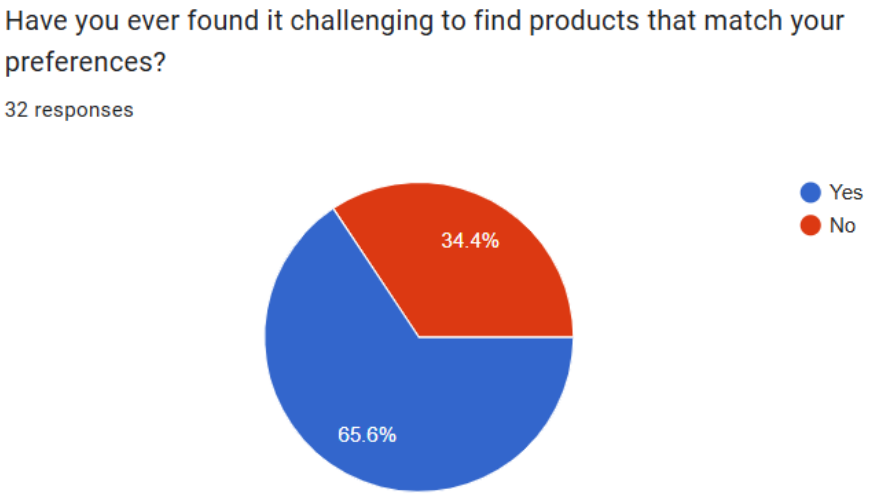


Figure 3.10 Survey Analysis 1

The result likely reflects that many people do find it challenging to find products that match their preferences, which is a common issue when there is a large variety of items available online. This suggests that users might struggle with personalization or relevant recommendations, which aligns with the concept of data sparsity and the difficulty in filtering through large inventories effectively.

3.2.5.2.2 Analysis 2

Figure 3.11 Survey Analysis 2

The result suggests that most users appreciate personalized recommendations because they help them find products more relevant to their preferences and save time. However, some users may prefer browsing and discovering products on their own or feel that recommendations aren't always accurate, which explains the 25% who answered "no."

3.2.5.2.3 Analysis 3

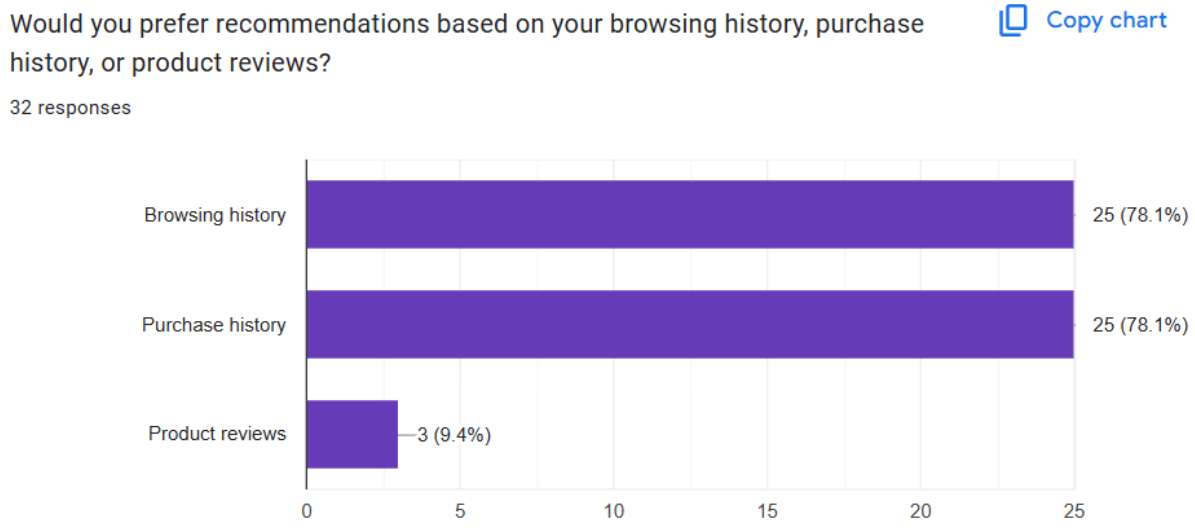


Figure 3.12 Survey Analysis 3

The result suggests that users prioritize recommendations based on their browsing and purchase history because these factors reflect their actual behavior and preferences. Users likely feel that

these sources are more relevant compared to product reviews, which may be less personalized or less directly related to their past interactions.

3.2.5.2.4 Analysis 4

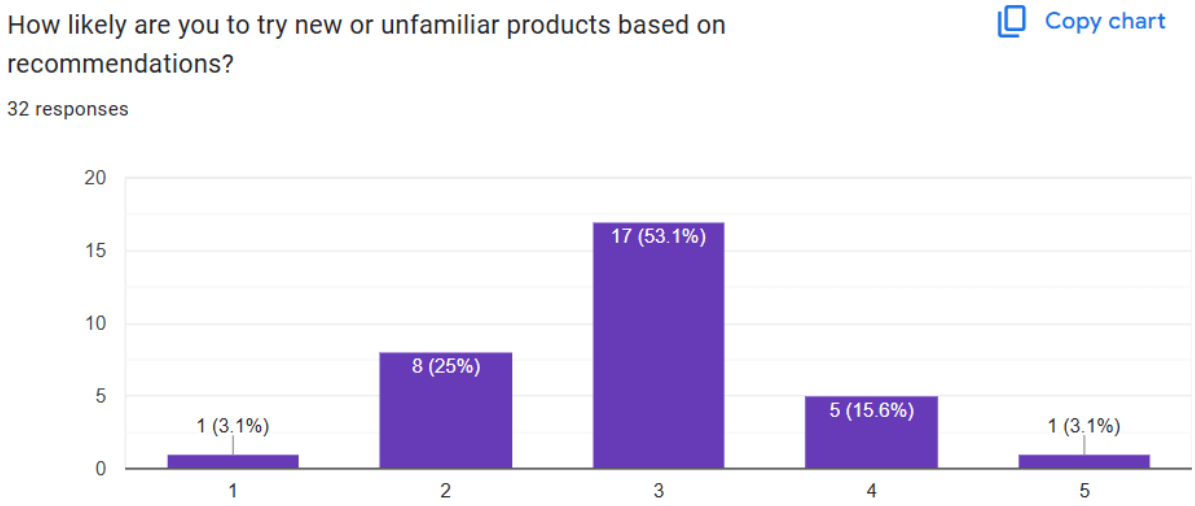


Figure 3.13 Survey Analysis 4

The result indicates that most users are neutral about trying new or unfamiliar products based on recommendations. This could be because they are cautious and prefer familiar products but are open to exploring new ones if the recommendations seem relevant.

3.2.5.3 Prototype

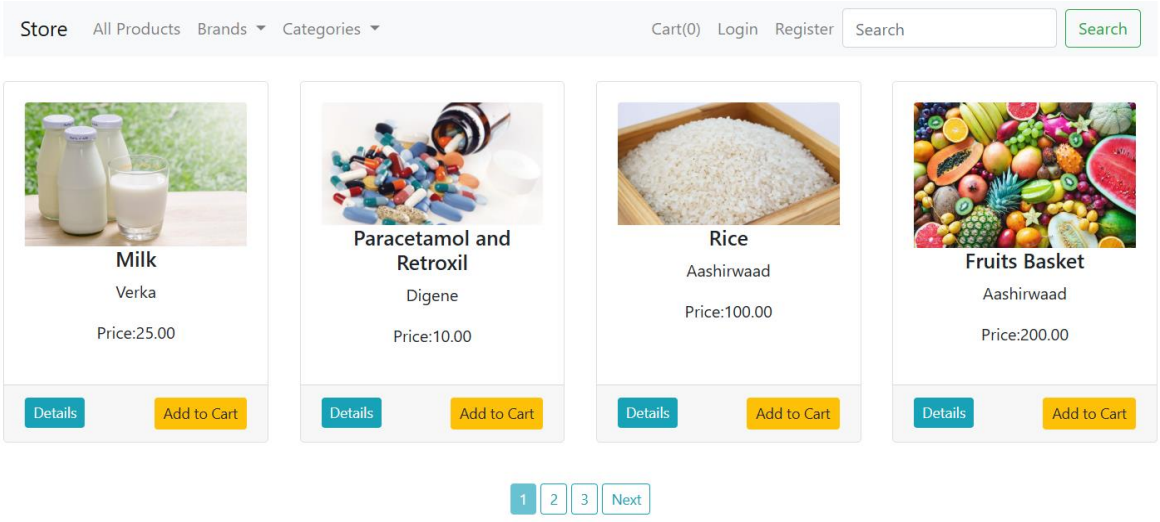


Figure 3.14: Proposed UI for the homepage

The proposed recommendation model will be integrated into the e-commerce website to provide personalized product suggestions to users. The model will interact with the website's database to retrieve user and product information required for generating recommendations. It will mainly utilize user interaction data such as purchase history to generate recommendations. The recommendations will be updated dynamically based on the user's latest interaction. The figures below show some other UI designs for the prototype.

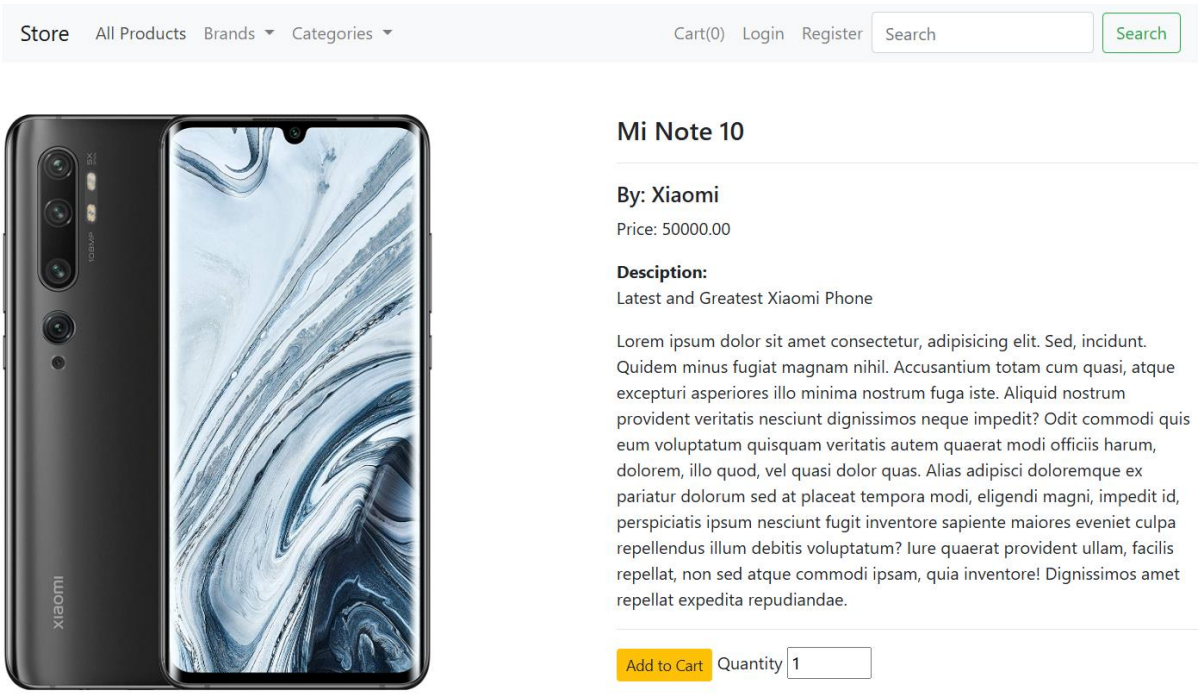


Figure 3.15: Item page

The figure above shows the UI design for the item page, displaying the name, price, and description. It allows users to add items to carts.

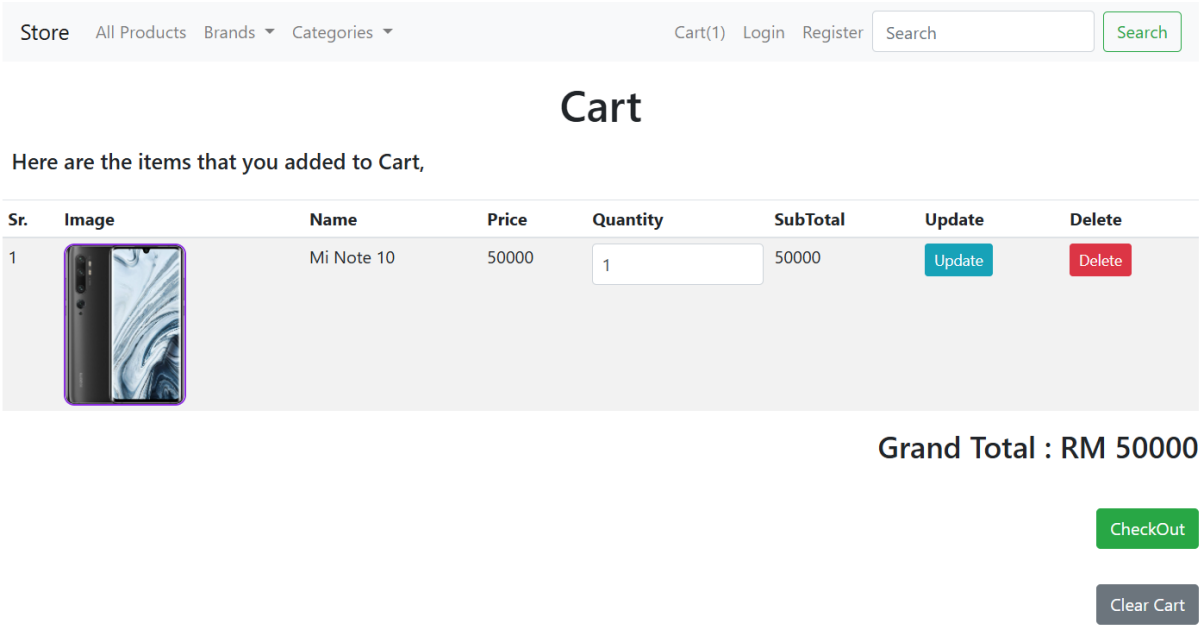


Figure 3.16: Cart page

The figure above shows the UI design for the cart page. It allows users to complete the purchase.

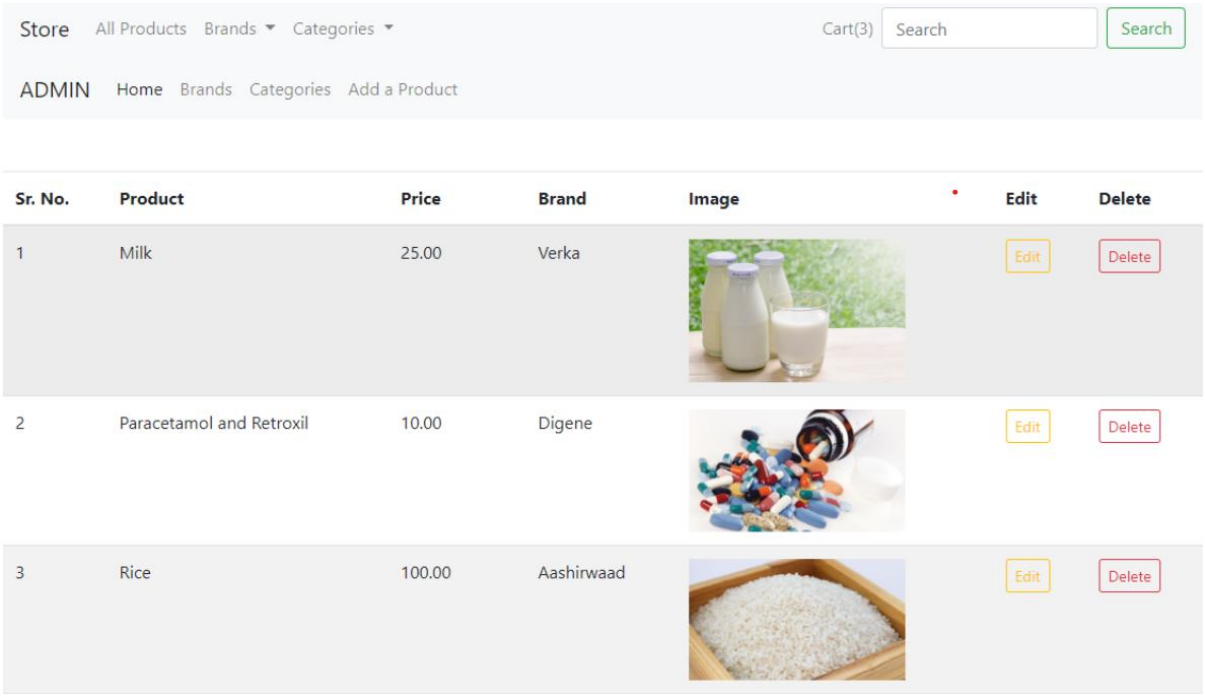


Figure 3.17 Admin Page

The figure above shows the UI design for the admin homepage. It allows the admin to update product details and delete products.

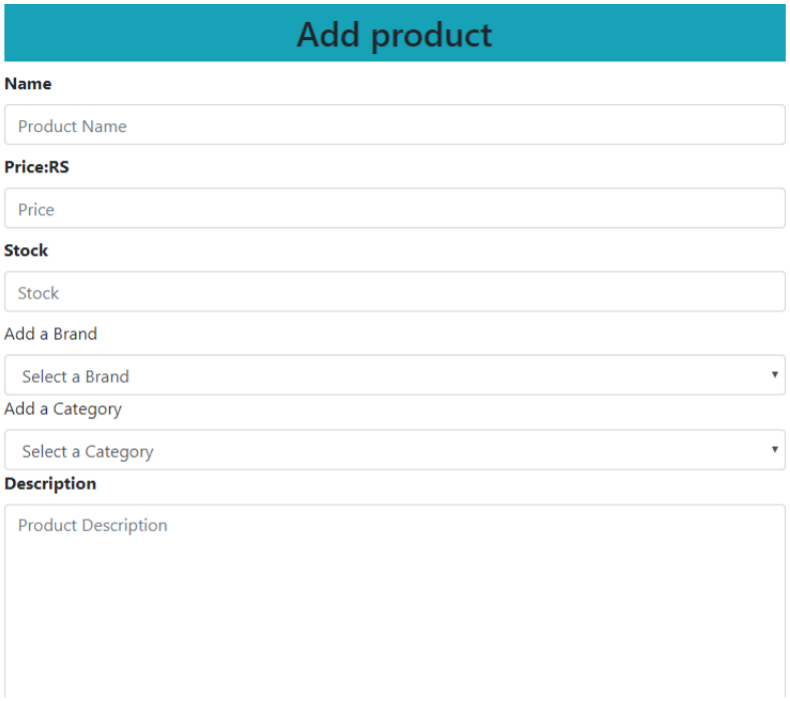


Figure 3.18 Add Product Page for Admin

The figure above shows the UI design of add product page for admin. It allows admin to edit new product details.

3.3 Summary

This chapter outlines the development of a hybrid recommendation system. The system combines content-based filtering and neural collaborative filtering to generate personalized recommendations for users. By leveraging the strengths of both approaches, the system effectively addresses key challenges such as the cold-start problem and data sparsity. A prototype e-commerce website will be developed to demonstrate the proposed recommendation system. The next chapter will focus on the technical implementation of the key elements emphasized in this chapter.

Chapter 4: Implementation and Result

4.1 Introduction

This chapter explains the implementation process of the hybrid recommendation system using content-based filtering and neural collaborative filtering. The chapter covers the tools and techniques used in developing the system, including the e-commerce prototype built using Python and Flask. Furthermore, this chapter provides an in-depth analysis of the evaluation and results of the recommendation models.

4.2 Environment Setup

The development of software project requires setting up an environment equipped with the essential tools and frameworks for building and deploying the system. This section outlines the environment setup for Anaconda and Python Jupyter notebook, which was used to train and test the recommendation models. Additionally, it includes an overview of Visual Studio Code, which functions as the integrated development environment (IDE) to develop the e-commerce prototype using Flask.

4.2.1 Anaconda

Anaconda installation includes a package manager (conda) that makes it easy to install and manage Python libraries. It can also create isolated environments for the proposed system, preventing conflicts with other projects. Furthermore, it comes with Jupyter Notebook, which is an interactive web-based tool that is used to develop and train/test the recommendation models.

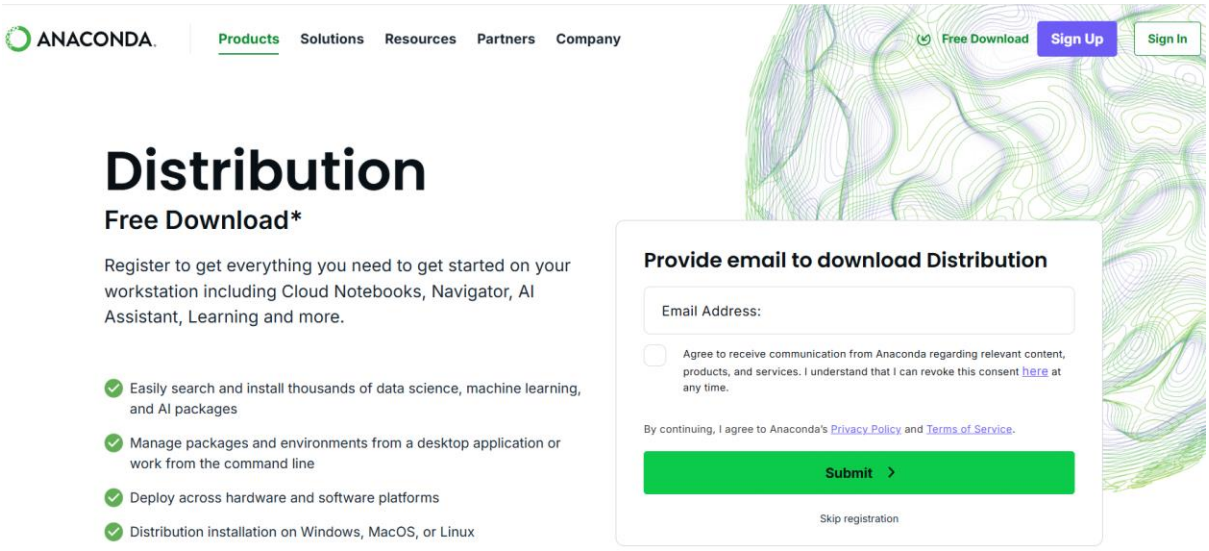


Figure 4.1 The official website of Anaconda

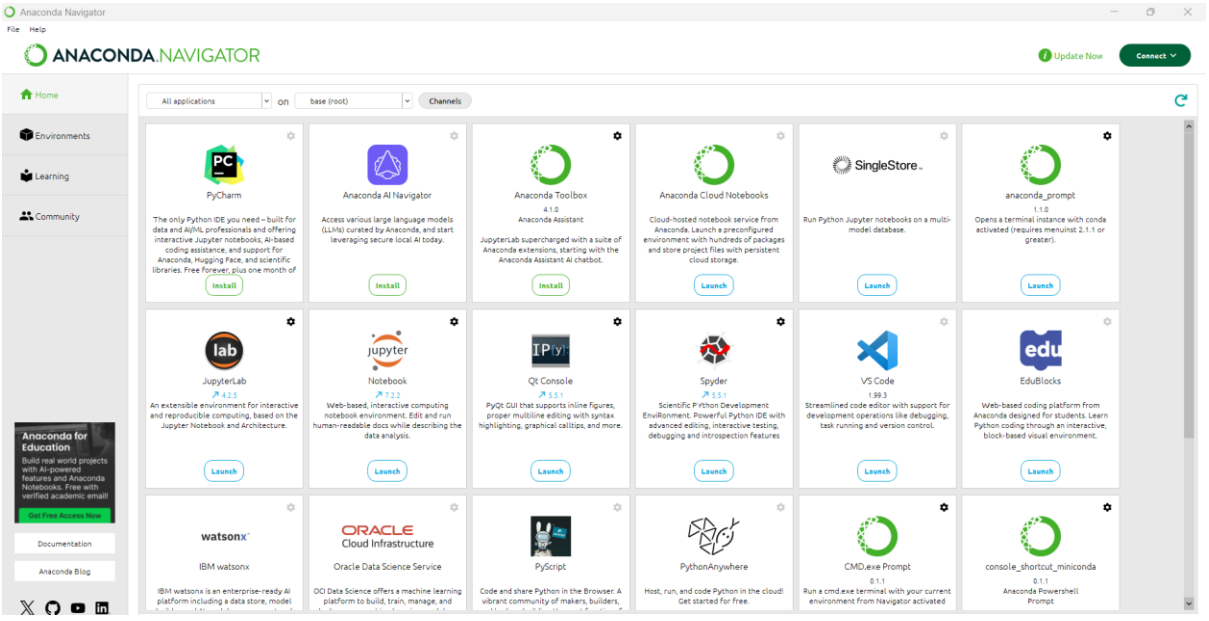


Figure 4.2 Successful installation of Anaconda

Anaconda Navigator is a graphical user interface (GUI) included with the Anaconda distribution. Anaconda Navigator provides a quick launch of Jupyter Notebook without using the command line.

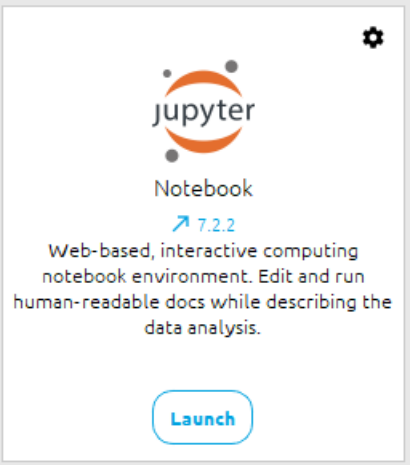


Figure 4.3 Jupyter Notebook in Anaconda Navigator

Apart from developing and train/test machine learning models, Jupyter Notebook also allows conducting Exploratory Data Analysis (EDA) to visualize data distribution using plots. This is important as it helps to spot missing values, imbalanced classes, or unusual distributions that may affect model accuracy.

4.2.2 Visual Studio Code

VS Code is a lightweight yet powerful code editor with support for Python and Flask. To set up Visual Studio Code for this project, the latest version was downloaded from the official website and installed.

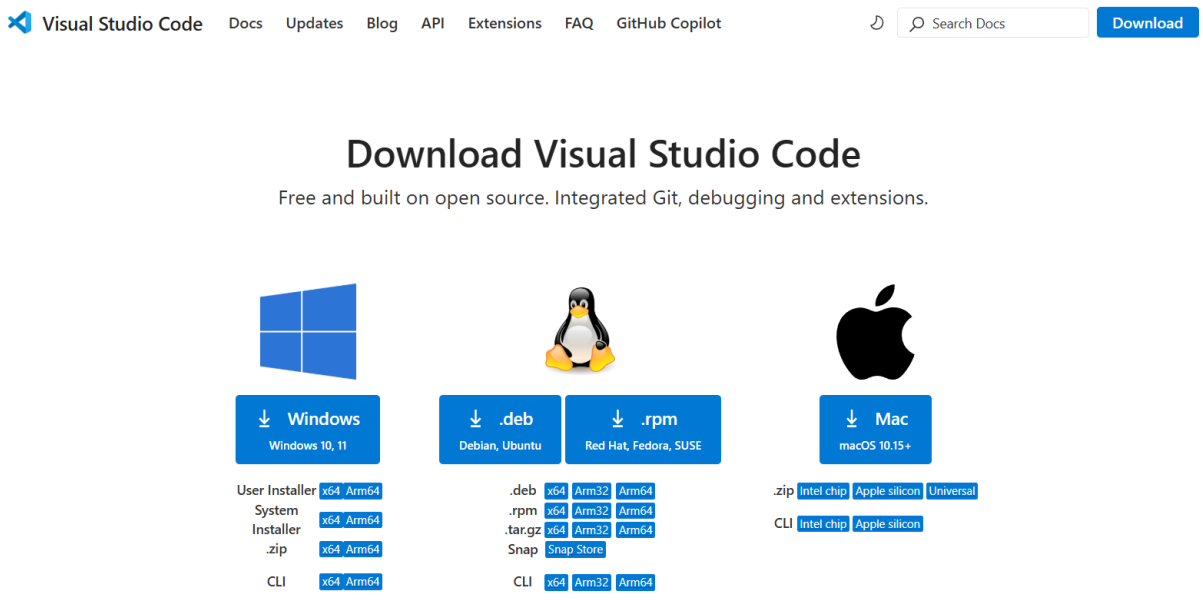


Figure 4.4 The official website of VS Code

After the installation, essential extensions for Python development, such as Python, Pylance, and Pip Manager, were added to Visual Studio Code to facilitate efficient coding.

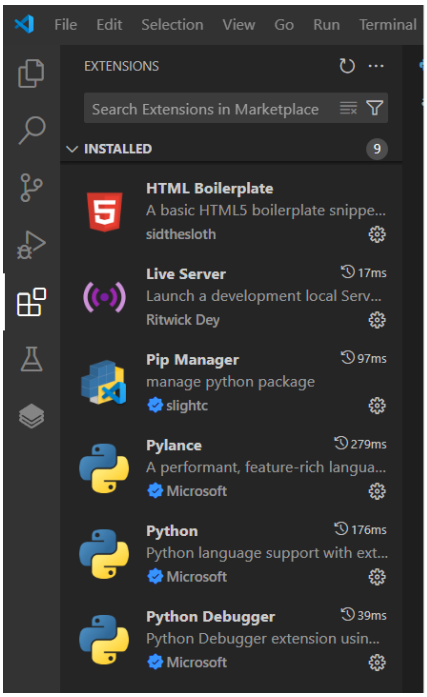


Figure 4.5 Installed extension in VS Code

To access the libraries installed in Anaconda, the Python code interpreter was set to Anaconda 3, using Python version 3.12.7.

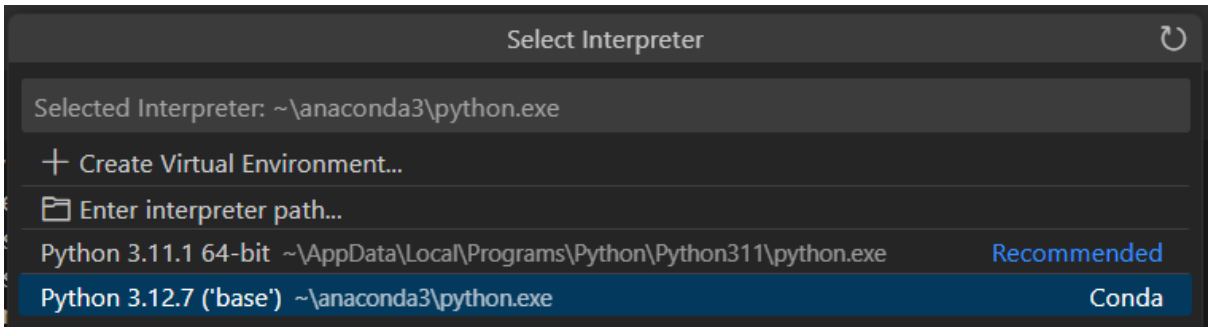


Figure 4.6 Anaconda3 selected as interpreter

4.3 Model development

The implementation of the hybrid recommendation system was carried out by following the design framework established in Chapter 3, ensuring consistency between the planned architecture and the developed system. This section will discuss the Exploratory Data Analysis

(EDA) of the dataset, the pre-processing steps, and the development of the recommendation models using Python Jupyter Notebook.

4.3.1 Exploratory Data Analysis (EDA)

The first step is to look at the general structure of the dataset. df.info() gives a concise summary of the DataFrame. This includes data types of each column, which helps to identify whether data types are appropriate for analysis/modelling. It also displays the number of non-null entries per column to identify which columns might contain missing values.

```
[15]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5381 entries, 0 to 5380
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   itemID     5381 non-null   int64
 1   userID     5381 non-null   int64
 2   rating     5381 non-null   int64
 3   timestamp  5381 non-null   object
 4   category   5381 non-null   object
 5   brand      5381 non-null   object
dtypes: int64(3), object(3)
memory usage: 252.4+ KB
```

Figure 4.7 df.info() showing general structure

The second step is to check the range of rating values (1 to 5) to confirm that the rating scale is correct and there are no unexpected outliers.

```
[16]: #Find the minimum and maximum ratings
      print('Minimum rating is: %d' %(df.rating.min()))
      print('Maximum rating is: %d' %(df.rating.max()))

Minimum rating is: 1
Maximum rating is: 5
```

Figure 4.8 Checking the scale of rating

The third step is to recheck how many missing (null/NaN) values exist in each column.

```
[17]: #Check for missing values
      print('Number of missing values across columns: \n',df.isnull().sum())

Number of missing values across columns:
 itemID       0
userID       0
rating       0
timestamp    0
category     0
brand        0
dtype: int64
```

Figure 4.9 Check missing values across columns

The next step is to identify the number of rows in the dataset, along with the number of unique users and items. This helps provide general insight into the dataset's coverage.

```
[18]: print("\nTotal no of ratings :",df.shape[0])
      print("Total No of Users   :", len(np.unique(df.userID)))
      print("Total No of products  :", len(np.unique(df.itemID)))

Total no of ratings : 5381
Total No of Users   : 5284
Total No of products  : 34
```

Figure 4.10 Identify dataset's coverage

The final step is to visualize the distribution of ratings using a plot. This helps identify any skew or imbalance in the data, such as an overrepresentation of high or low ratings, which can influence model training.

```
[19]: # Check the distribution of the rating
with sns.axes_style('white'):
    g = sns.catplot(x="rating", data=df, aspect=2.0, kind='count')
    g.set_ylabels("Total number of ratings")

plt.show()  # Ensure the plot is displayed
```

```
INFO:matplotlib.category:Using categorical units to plot a list of strings that are all parsable as floats or dates. If these strings should be plotted as numbers, cast to the appropriate data type before plotting.
INFO:matplotlib.category:Using categorical units to plot a list of strings that are all parsable as floats or dates. If these strings should be plotted as numbers, cast to the appropriate data type before plotting.
```

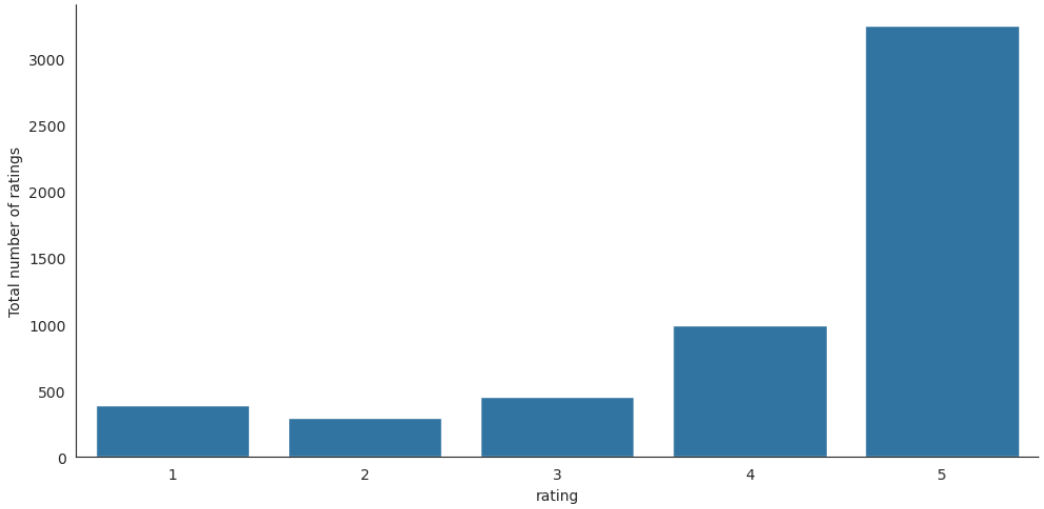


Figure 4.11 Check rating distribution

The distribution of ratings in the dataset is heavily skewed toward higher values, with rating 5 being the most frequent, followed by rating 4. This is a common pattern in recommendation systems, where users are more likely to rate items they enjoy and are less inclined to leave negative feedback unless strongly motivated.

4.3.2 Dataset Pre-processing

The first step is removing unnecessary columns like model_attr, year, user_attr, and split. This leaves only item_id, user_id, rating, timestamp, category, and brand. The first four columns are used for the NCF model, while category and brand are used for the CBF approach.

	A	B	C	D	E	F
1	itemID	userID	rating	timestamp	category	brand
2	2	1297	1	4/4/2003	Computers	Linksys

Figure 4.12 Important columns are kept

The second step is removing any rows with missing (empty) data. This ensures the dataset is clean and complete, which helps prevent errors during training and improves the reliability of the recommendation models.

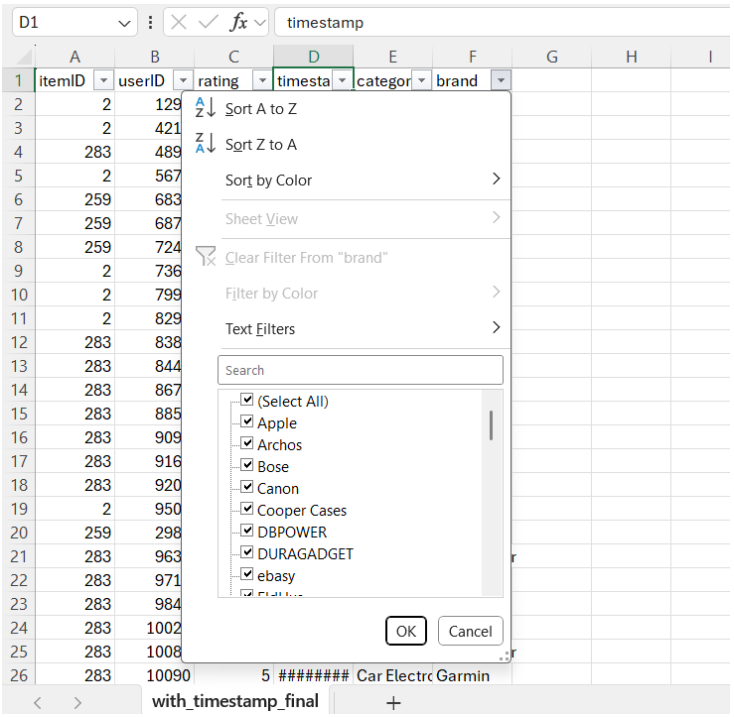


Figure 4.13 Removing empty rows in Excel

The third step is filtering out itemIDs that appear fewer than 100 times. This helps improve model performance by focusing on learning stronger patterns from popular items with richer interaction histories. This also reduces noise, as rarely interacted items may include outliers that don't reflect typical user behavior.

```
[3]: item_counts = df['itemID'].value_counts()
     df_filtered = df[df['itemID'].map(item_counts) >= 100]
```

Figure 4.14 Filter out itemID appearing less than 100 times

4.3.3 Neural Collaborative Filtering (NCF)

NCF combines Generalized Matrix Factorization (GMF) component, which captures linear relationships, with a Multi-Layer Perceptron (MLP) component, designed to model non-linear patterns, unified in the architecture known as NeuMF. The model uses the Microsoft Recommenders library (recommenders-team, 2018) for training and evaluation.

To prepare data for model training, the dataset is loaded from a CSV file into a DataFrame *df*. Then, the dataset is split by timestamp in chronological order using *train, test = python_chrono_split(df, 0.75)*. 75% of each user's interactions go into *train*, and the remaining

25% go into *test*. This preserves the temporal order of interactions, simulating a more realistic training and evaluation scenario. Next step is filtering out unknown users in test using *test = test[test["userID"].isin(train["userID"].unique())]*. This ensures the test set only includes users who also appear in the training set which prevents issues where the model has never seen the users during training. Then, a leave-one-out test set is created using *leave_one_out_test = test.groupby("userID").last().reset_index()*. This is a common evaluation setup that only last interaction of users is kept, using it as ground truth and the rest for training. The final step is to save the train, test, and leave-one-out test sets to CSV files for use in model training and evaluation. To prepare the data for use with the NCF model, an NCFDataset object is initialized to load the training and test files.

```
[2]: # Load dataset
     df = pd.read_csv("final_dataset.csv")

     SEED = DEFAULT_SEED #42

[3]:
     train, test = python_chrono_split(df, 0.75)
     # train.shape

[4]: #Filter out any users or items in the test set that do not appear in the training set.
     test = test[test["userID"].isin(train["userID"].unique())]
     # test.head()

[5]: test = test[test["itemID"].isin(train["itemID"].unique())]
     # test.head()

[6]: #Create a test set containing the last interaction for each user as for the leave-one-out evaluation.
     leave_one_out_test = test.groupby("userID").last().reset_index()

[7]: train_file = "./train.csv"
     test_file = "./test.csv"
     leave_one_out_test_file = "./leave_one_out_test.csv"
     train.to_csv(train_file, index=False)
     test.to_csv(test_file, index=False)
     leave_one_out_test.to_csv(leave_one_out_test_file, index=False)

[8]: data = NCFDataset(train_file=train_file, test_file=leave_one_out_test_file, seed=SEED, overwrite_test_file_full=True)
     print(data.n_users)
     print(data.n_items)
```

Figure 4.15 Prepare data for model training/testing

```
[9]: model = NCF(
         n_users=data.n_users,
         n_items=data.n_items,
         model_type="NeuMF",
         n_factors=12,
         layer_sizes=[8],
         n_epochs=20,
         batch_size=64,
         learning_rate=1e-3,
         verbose=10,
         seed=SEED
     )
```

```
C:\Users\User\anaconda3\Lib\site-packages\tensorflow\python\keras\engine\base_layer_v1.
ved in a future version. Please use `layer.__call__` method instead.
  warnings.warn('`layer.apply` is deprecated and '
```

```
[10]: with Timer() as train_time:
          model.fit(data)

      print("Took {} seconds for training.".format(train_time.interval))
      model.save('ncf_model')
```

```
INFO:recommenders.models.ncf.ncf_singlenode:Epoch 10 [64.37s]: train_loss = 0.071197
INFO:recommenders.models.ncf.ncf_singlenode:Epoch 20 [49.03s]: train_loss = 0.006800
Took 1091.300115399994 seconds for training.
```

Figure 4.16 Model hyperparameter and training

In collaborative filtering models like NCF, the *n_factors* parameter determines the dimensionality of the embedding vectors used to represent users and items in a latent factor space. Since the dataset is about 5k rows, a *n_factors* of 12 is suitable for the model, as increasing the embedding size further can significantly increase the risk of overfitting.

The *layer_sizes* parameter in the NCF model defines the number and size of the hidden layers in the Multi-Layer Perceptron (MLP) component. A single hidden layer with 8 neurons should match the model complexity because a deeper MLP might have too much capacity and could easily overfit the training data.

The number of training epochs determines how many times the entire training dataset is passed through the model during the training process. A *n_epochs* of 20 is suitable for the model as it decreases the time for training, and stopping the training early is recommended for a 5k dataset.

The batch size defines the number of training samples used in one forward and backwards pass of the neural network during training. A *batch_size* of 64 helps the model escape local minima and generalize better but can potentially introduce more noise into the gradient estimation

during training. The choice of batch size often involves a trade-off between training speed and model performance.

The learning rate is a crucial hyperparameter that controls the step size at which the model's weights are updated during training. A *learning_rate* of 0.001 is a common default value for many neural network optimizers. In the proposed model, it matches properly with the other hyperparameters. To find out whether the learning rate is appropriate, a grid search is implemented.

```
# Define hyperparameter grid
learning_rates = [1e-3, 1e-4]
best_ndcg = -1
best_lr = None

# Hyperparameter tuning with manual grid search
for lr in learning_rates:
    print(f"Training with learning rate: {lr}")

    # Prepare data
    data = NCFDataset(train_file=train_file, test_file=test_file, seed=42, overwrite_test_file_full=True)

    # Initialize model
    model = NCF(
        n_users=data.n_users,
        n_items=data.n_items,
        model_type="NeuMF",
        n_factors=12,
        layer_sizes=[8],
        n_epochs=20,
        batch_size=64,
        learning_rate=lr,
        verbose=10,
        seed=42
    )

    # Train model
    model.fit(data)
```

Figure 4.17 Hyperparameter tuning with grid search

Based on Figure 4.18 below, the best learning rate is 0.001 with NDCG of 0.3516 at top 30 recommendations.

```
Best learning rate: 0.001 with NDCG@30: 0.3516489704508821
```

Figure 4.18 Result of hyperparameter tuning

4.3.4 Content-based Filtering (CBF)

The Content-Based Filtering (CBF) method implemented in the *cbf_filter* function aims to recommend items to users based on the similarity of item metadata, specifically the category and brand attributes. The process begins by identifying the user's interactions from the test dataset. If the user has interacted with any item, the function extracts the category and brand of the first item the user interacted with. These two features are used as a representation of the user's current preference.

To model item similarities, the function creates a new text feature by concatenating each item's category and brand into a single string. These combined text features are then transformed into numerical vectors using the Term Frequency–Inverse Document Frequency (TF-IDF) vectorizer. TF-IDF captures the importance of terms (in this case, category and brand names) across all items, providing a meaningful representation of the item metadata.

After vectorization, cosine similarity is computed between all item vectors to measure how closely related each item is to others based on their textual features. The function then identifies the index of items that match the user's current category and brand and calculates their similarity to other items. From these results, the top K most similar items are selected and returned as recommendations.

```python
def cbf_filter(user_id, all_items):
    """Filter items using CBF before passing them to NCF."""
    user_data = test[test["userID"] == user_id]  # Get user's test interactions
    if user_data.empty:
        return []  # No data for this user

    category, brand = user_data.iloc[0]["category"], user_data.iloc[0]["brand"]

    # Apply CBF: Get items similar to category & brand
    combined_features = df["category"] + " " + df["brand"]
    vectorizer = TfidfVectorizer()
    feature_vectors = vectorizer.fit_transform(combined_features)
    similarity = cosine_similarity(feature_vectors, feature_vectors)

    idx = df[(df["category"] == category) & (df["brand"] == brand)].index
    if idx.empty:
        return []  # No similar items found

    sim_scores = list(enumerate(similarity[idx[0]]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)[1:TOP_K+1]
    cbf_filtered_items = [df.iloc[i[0]]["itemID"] for i in sim_scores]

    return cbf_filtered_items
```

Figure 4.19 Content-based filtering

4.3.5 Hybrid recommendation model

The hybrid recommendation model combines Content-Based Filtering (CBF) and Neural Collaborative Filtering (NCF) to generate more accurate and personalized recommendations. The process starts by iterating over all unique users in the training dataset. For each user, the model first applies the CBF filter to select a subset of items that are similar in content (category and brand) to those previously interacted with by the user. This serves as a pre-filtering step that narrows down the candidate items, improving both performance and relevance.

If no content-similar items are found for a given user, the model skips to the next user. Otherwise, it prepares a list of user–item pairs by repeating the current user ID for each of the filtered items. These user–item pairs are then passed to the pre-trained NCF model, which predicts a score for each pair, representing the likelihood that the user will interact positively with the item.

All predictions, along with the corresponding user and item IDs, are stored in a DataFrame. To evaluate the model or prepare for further processing, the predictions are merged with the original training dataset using an outer join on both user and item IDs. Finally, only rows that were not part of the original training set (i.e., where the rating is null) are retained. This ensures that recommendations are only made for unseen user–item pairs, avoiding redundancy and enabling proper evaluation.

```
users, items, preds = [], [], []
for user in train.userID.unique():
    filtered_items = cbf_filter(user, train.itemID.unique())  # Apply CBF filter
    if not filtered_items:
        continue

    user_list = [user] * len(filtered_items)
    users.extend(user_list)
    items.extend(filtered_items)
    preds.extend(list(ncf_model.predict(user_list, filtered_items, is_list=True)))

all_predictions = pd.DataFrame(data={"userID": users, "itemID":items, "prediction":preds})

merged = pd.merge(train, all_predictions, on=["userID", "itemID"], how="outer")
all_predictions = merged[merged.rating.isnull()].drop('rating', axis=1)
```

Figure 4.20 Hybrid recommendation model

4.4 Evaluation and Result

Evaluation is essential in determining the performance and practical effectiveness of recommendation models. It involves assessing how well a model predicts items that are relevant to users based on past behavior or preferences. To perform this evaluation, four widely used metrics were employed: Mean Average Precision (mAP), Normalized Discounted Cumulative Gain (NDCG), Precision@K, and Recall@K. These metrics provide a holistic view of recommendation quality, balancing relevance, ranking quality, and retrieval effectiveness.

4.4.1 Benchmark Models

The first benchmark model is Surprise Singular Value Decomposition (SVD). SVD is a fundamental matrix decomposition technique originating from the field of linear algebra. SVD can be understood as a process of identifying the principal components or latent factors that best explain the variance in the data. Considering a movie rating matrix, SVD can uncover hidden factors that influence user preferences, such as movie genres (action, comedy, drama), the presence of specific actors, or even abstract qualities like the "mood" of a film. These latent factors are not explicitly present in the original rating data but are inferred from the patterns of how users have rated different movies. At its core, SVD provides a way to factorize any rectangular matrix into a product of three other matrices, revealing the underlying structure and patterns within the data. In the context of recommender systems, SVD is applied to the user-item rating matrix. This matrix is constructed with rows representing individual users, columns representing the items, and the entries containing the ratings given by users to those items.

```
[5]: svd = surprise.SVD(random_state=0, n_factors=200, n_epochs=30, verbose=True)

with Timer() as train_time:
    svd.fit(train_set)

print(f"Took {train_time.interval} seconds for training.")
```

Figure 4.21 Training SVD as benchmark model

The other benchmark model is Light Graph Convolution Network (LightGCN). LightGCN is a simplified variant of Neural Graph Collaborative Filtering (NGCF), which incorporates

Graph Convolutional Networks (GCNs) into recommendation systems. GCNs are neural networks designed to learn patterns from graph-structured data. While applicable across various domains, GCNs are especially effective in recommendation systems due to their strength in capturing and representing relationships. In recommendation contexts, GCNs help embed interaction signals directly into the model. For example, items a user has interacted with can serve as features that reflect the user's preferences. Likewise, the users associated with a particular item can act as features for that item, helping to assess the collaborative similarity between items.

```
[3]: data = ImplicitCF(train=train, test=test, seed=SEED)
     hparams = prepare_hparams(yaml_file,
                               n_layers=3,
                               batch_size=BATCH_SIZE,
                               epochs=EPOCHS,
                               learning_rate=0.005,
                               eval_epoch=5,
                               top_k=TOP_K,
                              )
```

```
[4]: model = LightGCN(hparams, data, seed=SEED)

     with Timer() as train_time:
         model.fit()

     print("Took {} seconds for training.".format(train_time.interval))
```

Figure 4.22 Training LightGCN as benchmark model

4.4.2 Evaluation

The figure below shows the evaluation result of the proposed model and the benchmark models.

Table 4.1 Evaluation result

Technique	Mean Average Precision (mAP)	Normalized Discounted Cumulative Gain (NDCG)	Precision@K	Recall@K
Proposed CBF + NCF	0.439	0.324	0.044	1.333

SVD	0.102	0.003	0.028	0.281
LightGCN	0.055	0.233	0.033	1.000

Mean Average Precision (mAP) measures the overall precision across all users by taking into account the order of recommended items and their relevance. A higher mAP indicates that the model is more effective at ranking relevant items higher. In the results, the hybrid model (CBF + NCF) achieved a mAP of 0.439, significantly outperforming SVD (0.102) and LightGCN (0.055). This demonstrates that the hybrid model is substantially better at delivering consistently accurate recommendations.

Normalized Discounted Cumulative Gain (NDCG) evaluates the ranking quality of the recommendations by giving higher importance to relevant items that appear earlier in the recommendation list. The hybrid model recorded an NDCG of 0.324, which is notably higher than LightGCN (0.233) and far superior to SVD (0.003). Subhan et al. (2025) achieved NDCG of 0.35 in a recommender system using hybrid Neural Collaborative Filtering (NCF) for E-commerce cosmetic product. The performance aligns with the proposed hybrid model with only minor differences.

Precision@K measures the proportion of relevant items among the top-K recommendations. Although the hybrid model's Precision@K is 0.044, only slightly higher than LightGCN (0.033) and SVD (0.028), it still indicates a stronger ability to include relevant items in the top-K list.

Recall@K, on the other hand, evaluates how many of the relevant items were successfully retrieved among the top-K recommendations. The hybrid model achieved a Recall@K of 1.333, which is substantially higher than LightGCN (1.000) and SVD (0.281). This means the hybrid model is more effective at capturing a larger proportion of items that are actually relevant to users, making it especially useful in contexts where comprehensive recommendations are important.

In conclusion, the hybrid model combining Content-Based Filtering with Neural Collaborative Filtering demonstrates superior performance across all key evaluation metrics. This indicates that integrating user-specific content features with deep learning-based collaborative filtering results in more accurate, personalized, and effectively ranked recommendations. The results clearly validate the hybrid model as a more reliable and robust approach compared to traditional matrix factorization (SVD) and graph-based (LightGCN) methods.

Comparison with previous study

Research paper	Model used	Data splitting	Dataset	Evaluation metric
Subhan et al., 2025	Neural Collaborative Filtering (NCF), Bidirectional Long Short-Term Memory (BiLSTM)	70% training, 30% testing	eCommerce Event History in Cosmetics Shop	NDCG (Normalized Discounted Cumulative Gain): 0.35 MRR (Mean Reciprocal Rank): 0.56
Proposed model	Neural Collaborative Filtering (NCF), Content Based Filtering (CBF)	75% training, 25% testing	Amazon electronic product sales	NDCG (Normalized Discounted Cumulative Gain): 0.324 Mean Average Precision (mAP): 0.439

4.4.3 Time Efficiency Evaluation

Time efficiency evaluation assesses how quickly a recommendation system can generate item suggestions for users. This metric is crucial, especially in real-time applications like e-commerce, where fast response times directly impact user experience and system scalability. A model that offers high accuracy but takes too long to produce results may not be suitable for practical deployment, making the balance between performance and speed an important consideration. Therefore, evaluating the processing time allows assessing the trade-offs between model accuracy and computational cost, ensuring the selected model meets the desired balance of quality and speed for the application context.

Table 4.2 Processing Time for Each Technique

Technique	**Processing Time**
CBF + NCF	0m 14s
SVD	0m 11s

LightGCN	0m 1s

Based on the results, LightGCN is the fastest among the three models, requiring only 1 second to generate recommendations. Its graph-based architecture eliminates unnecessary operations found in traditional GCNs (like feature transformations and nonlinear activations), significantly speeding up inference. This makes it highly suitable for real-time systems, although it trades off some recommendation accuracy.

SVD follows closely with a processing time of 11 seconds. While it is slightly faster than the CBF + NCF hybrid, it still requires over 10 seconds to produce recommendations. The speed improvement likely comes from the lack of content-based filtering and the use of linear algebra techniques rather than deep learning. However, it also has lower accuracy compared to the hybrid model, meaning the time saved may not justify the loss in recommendation quality depending on application goals.

The CBF + NCF hybrid model, while the most accurate in terms of recommendation quality, has the longest processing time at 14 seconds. The added latency is primarily due to the initial filtering of items using content features (category and brand), followed by the prediction stage using a deep neural network model. This two-stage process increases computational complexity, especially when it involves both cosine similarities and deep learning computations.

By evaluating time efficiency, it helps to determine whether a model is suitable for real-time usage, whether it requires optimization, or whether a lighter alternative might be necessary. It also helps in comparing multiple models to find the best trade-off between recommendation quality and computational cost. Ultimately, it ensures that the recommendation system not only performs well in accuracy but also meets performance and scalability requirements for deployment.

4.5 Summary

This chapter discussed the implementation of the proposed hybrid recommendation system combining Content-Based Filtering (CBF) and Neural Collaborative Filtering (NCF). The development environment included Python, Flask, SQLAlchemy, and machine learning libraries like Scikit-learn and TensorFlow. Data preprocessing involved removing irrelevant columns, handling missing values, and filtering out items with low interaction to improve

model performance. The CBF model used item features like category and brand to find similar products, while the NCF model used user-item interactions to predict preferences. The hybrid model leverages CBF to narrow down item candidates and NCF to rank them based on predicted relevance. Evaluation metrics such as Precision@K, Recall@K, and NDCG@K demonstrated that the hybrid model delivered more accurate recommendations than individual models.

Chapter 5: Web Prototype Development

5.1 Introduction

This chapter discusses how the hybrid recommendation model is integrated into the web prototype using Flask. The chapter also covers the testing performed and presents user scenarios demonstrating how the prototype functions.

5.2 Integrating Proposed Model in E-commerce Prototype

Flask is a lightweight and flexible web framework ideal for building e-commerce prototypes, especially those involving machine learning. It allows rapid development and easy integration with Python-based recommendation models. Flask also supports essential features like database handling, user authentication, and API creation, making it suitable for deploying functional prototypes efficiently.

To integrate the neural collaborative filtering (NCF) model into the website, the model is first loaded as shown in the figure below.

```
48  # Loading model
49  ncfmodel = NCF(
50      n_users= 5284,
51      n_items= 34,
52      model_type="NeuMF",
53      n_factors=12,
54      layer_sizes=[8],
55      n_epochs=20,
56      batch_size=64,
57      learning_rate=1e-3,
58      verbose=10,
59      seed=42
60  )
61
62  ncfmodel.load(neumf_dir = 'app/ncf_model')
```

Figure 5.1 Load ncf model in web prototype

To generate recommendations for users, the *recommend* function will utilize both CBF and NCF as shown in the figure below.

```python
def recommend(user_id, all_items):
    TOP_K = 30
    """Filter items using CBF before passing them to NCF."""
    past_orders = Order.query.filter_by(uid=user_id).all()
    ordered_items = [ordered_item.item for order in past_orders for ordered_item in order.items]
    if not ordered_items:
        return []

    reference_item = ordered_items[-1]  # Last purchased item
    category, brand = reference_item.category, reference_item.brand

    # Apply CBF (Cosine Similarity)
    cbf_filtered_item_ids = [item.id for item in all_items]
    df = pd.DataFrame([(item.id, item.category, item.brand) for item in all_items],
                      columns=["itemID", "category", "brand"])

    combined_features = df["category"] + " " + df["brand"]
    vectorizer = TfidfVectorizer()
    feature_vectors = vectorizer.fit_transform(combined_features)
    similarity = cosine_similarity(feature_vectors, feature_vectors)

    target_idx = df[(df["category"] == category) & (df["brand"] == brand)].index
    if target_idx.empty:
        target_idx = df[(df["category"] == category) | (df["brand"] == brand)].index

    if len(target_idx):
        sim_scores = list(enumerate(similarity[target_idx[0]]))
        sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

        # Exclude the target item itself and get the top K similar items
        top_k_similar_indices = [i[0] for i in sim_scores[1:TOP_K+1]]
        cbf_filtered_item_ids = df.iloc[top_k_similar_indices]["itemID"].tolist()
        # cbf_filtered_item_ids = [df.iloc[i[0]]["itemID"] for i in sim_scores]

    cbf_filtered_items = Item.query.filter(Item.id.in_(cbf_filtered_item_ids)).all()

    # Apply ncf
    ncfmodel.user2id = create_user_to_id_mapping()
    ncfmodel.item2id = create_item_to_id_mapping()

    predictions = [(item, ncfmodel.predict(user_id, item.id)) for item in cbf_filtered_items]
    sorted_recommendations = [item for item, _ in sorted(predictions, key=lambda x: x[1], reverse=True)]

    sorted_predictions = sorted(predictions, key=lambda x: x[1], reverse=True)
    for i, (item, score) in enumerate(sorted_predictions, start=1):
        print(f"{i}. {item.name} (ID: {item.id}) - Predicted Rating: {score:.4f}")

    return sorted_recommendations
```

Figure 5.2 function to generate recommendations

The *recommend* function implements a hybrid recommendation system by combining Content-Based Filtering (CBF) with a Neural Collaborative Filtering (NCF) model. It is designed to generate personalized recommendations for a given user by utilizing both the content attributes of items (specifically category and brand) and collaborative signals learned from user-item interactions.

The process begins by retrieving the user's past orders from the database. If the user has no purchase history, the function returns an empty list since no reference can be made for similarity.

If the user does have past orders, the most recent purchase is identified and used as the reference point. The category and brand of this reference item are extracted, as they will be used to find similar items.

Next, the function applies the content-based filtering component. It constructs a DataFrame from all available items, extracting their item ID, category, and brand. It then creates a combined textual feature by concatenating the category and brand of each item. Using a TF-IDF vectorizer, these combined features are transformed into numerical vectors, which are then compared using cosine similarity to measure how similar each item is to the reference item. The function identifies the reference item's index and computes the similarity scores against all other items. The top-k most similar items (excluding the reference item itself) are selected based on these scores.

These filtered item IDs represent the content-based candidate pool of items that are most relevant to the user based on their last purchase. The function queries the database to retrieve the actual item objects corresponding to these IDs. These items are then passed to the collaborative filtering model for rating prediction.

In the collaborative filtering phase, the function maps the user ID and item IDs into the numerical indices expected by the trained NCF model. The model then predicts how much the user would potentially like each of the content-filtered items. These predictions are sorted in descending order of predicted rating to prioritize the most relevant recommendations.

Finally, the function returns a sorted list of item recommendations along with their predicted scores.

The Flask route defined for the home page is responsible for displaying items to users. If a user is logged in (authenticated), the system calls the *recommend()* function using the user's ID and the list of available items. The home.html template is rendered with the personalized recommendations, providing a user-specific shopping experience.

```python
@app.route("/")
def home():
    items = Item.query.all()
    recommendations = items

    if current_user.is_authenticated:
        recommended_items = recommend(current_user.id, items)
        if recommended_items:
            recommendations = recommended_items  # Show personalized recommendations

    return render_template("home.html", items=recommendations)
```

Figure 5.3 Displaying recommendations on homepage

The figure below illustrates the system architecture of the Flask web prototype. Users interact with the web interface by browsing and sending requests, which are handled by the Flask application. These interactions are logged and passed through the application's static and model components. The hybrid recommendation model, trained using a collected dataset, analyzes user behavior and product features to generate personalized suggestions. This information is then used to query the database and deliver relevant product recommendations back to the user.

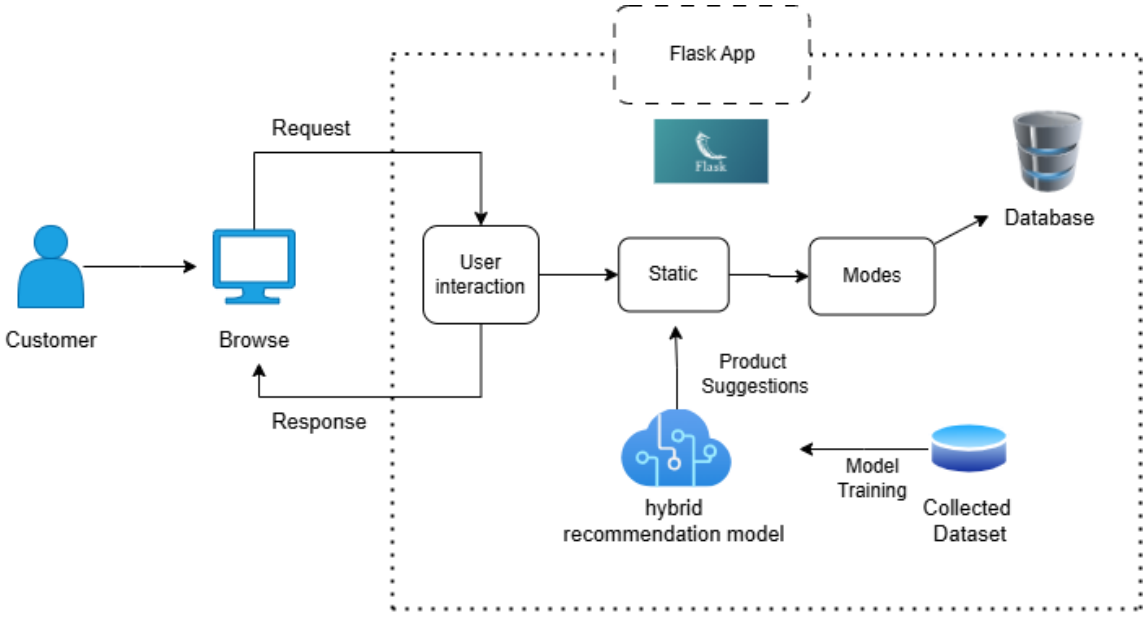


Figure 5.4 Overview of the system architecture

5.3 User scenarios

This section illustrates a typical interaction flow between a user and the developed e-commerce platform, demonstrating how the proposed model is integrated into the user experience.

A user begins their journey by accessing the AI-E-shop platform. To proceed, they are required to log in using their registered email address and password, as shown in Figure 5.4. This login process ensures secure access to personalized features, including recommendations and order history.

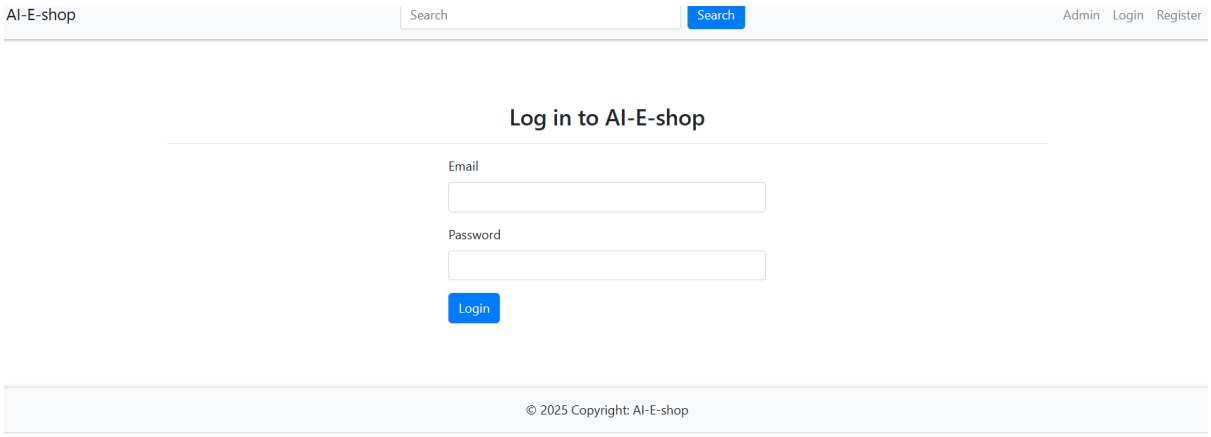


Figure 5.5 User login

Upon successful authentication, the user is redirected to the homepage, where a list of available products is displayed.

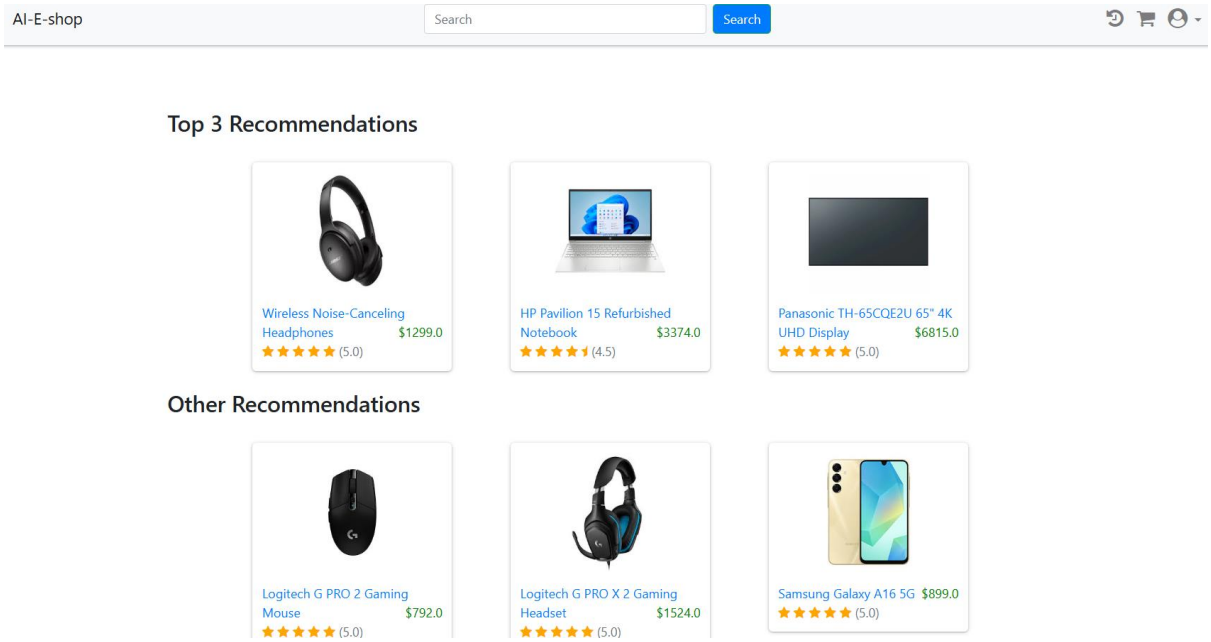


Figure 5.6 Homepage

The user browses through the catalog and selects an item of interest. After reviewing the item details, the user clicks the **"Add to Cart"** button to place the product in their shopping cart.

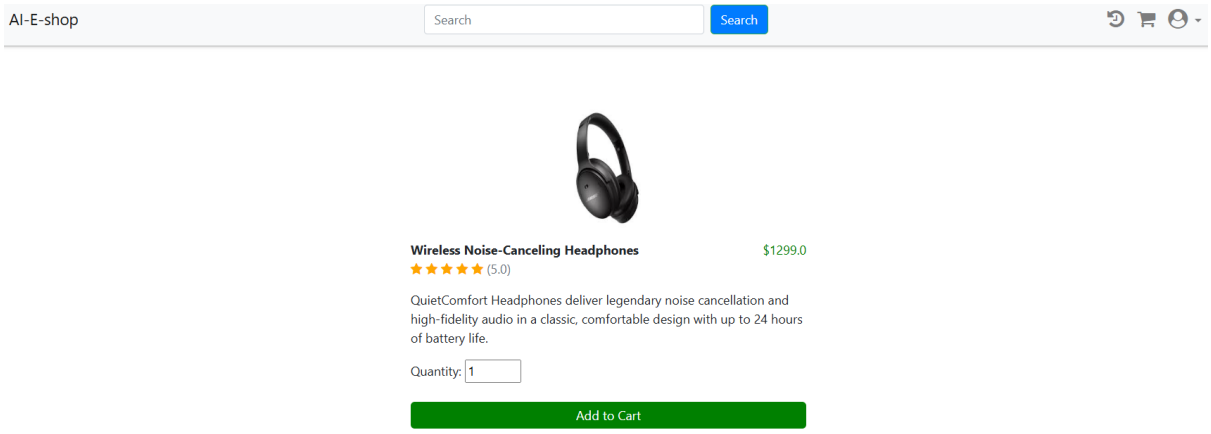


Figure 5.7 Item page

The user proceeds by clicking "Checkout" after reviewing the items.

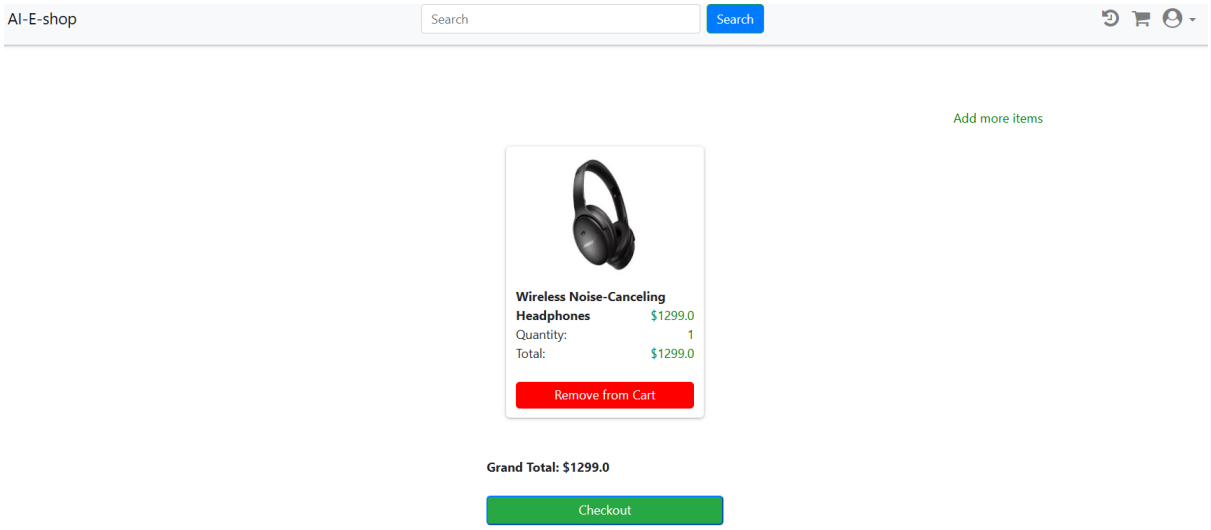


Figure 5.8 User checkout

After the checkout process, the system generates a new set of personalized recommendations using the hybrid recommendation model. Figure below shows the source code for the recommendation process.

```python
def recommend(user_id, all_items):
    TOP_K = 30
    """Filter items using CBF before passing them to NCF."""
    past_orders = Order.query.filter_by(uid=user_id).all()
    ordered_items = [ordered_item.item for order in past_orders for ordered_item in order.items]
    if not ordered_items:
        return []

    reference_item = ordered_items[-1]  # Last purchased item
    category, brand = reference_item.category, reference_item.brand

    # Apply CBF (Cosine Similarity)
    cbf_filtered_item_ids = [item.id for item in all_items]
    df = pd.DataFrame([(item.id, item.category, item.brand) for item in all_items],
                      columns=["itemID", "category", "brand"])

    combined_features = df["category"] + " " + df["brand"]
    vectorizer = TfidfVectorizer()
    feature_vectors = vectorizer.fit_transform(combined_features)
    similarity = cosine_similarity(feature_vectors, feature_vectors)

    target_idx = df[(df["category"] == category) & (df["brand"] == brand)].index
    if target_idx.empty:
        target_idx = df[(df["category"] == category) | (df["brand"] == brand)].index

    if len(target_idx):
        sim_scores = list(enumerate(similarity[target_idx[0]]))
        sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

        # Exclude the target item itself and get the top K similar items
        top_k_similar_indices = [i[0] for i in sim_scores[1:TOP_K+1]]
        cbf_filtered_item_ids = df.iloc[top_k_similar_indices]["itemID"].tolist()
        # cbf_filtered_item_ids = [df.iloc[i[0]]["itemID"] for i in sim_scores]

    cbf_filtered_items = Item.query.filter(Item.id.in_(cbf_filtered_item_ids)).all()

    # Apply ncf
    ncfmodel.user2id = create_user_to_id_mapping()
    ncfmodel.item2id = create_item_to_id_mapping()

    predictions = [(item, ncfmodel.predict(user_id, item.id)) for item in cbf_filtered_items]
    sorted_recommendations = [item for item, _ in sorted(predictions, key=lambda x: x[1], reverse=True)]

    sorted_predictions = sorted(predictions, key=lambda x: x[1], reverse=True)
    for i, (item, score) in enumerate(sorted_predictions, start=1):
        print(f"{i}. {item.name} (ID: {item.id}) - Predicted Rating: {score:.4f}")

    return sorted_recommendations
```

Figure 5.9 function to generate recommendations

In the *recommend()* function, the Content-Based Filtering (CBF) component first analyzes the category and brand of the purchased items to identify similar products and generate a pool of candidate recommendations. This candidate list is then passed to the Neural Collaborative Filtering (NCF) model, which uses the user's interaction history and learned latent features to predict preferences and rank the items based on their predicted ratings.

Figure below shows the recommended items.

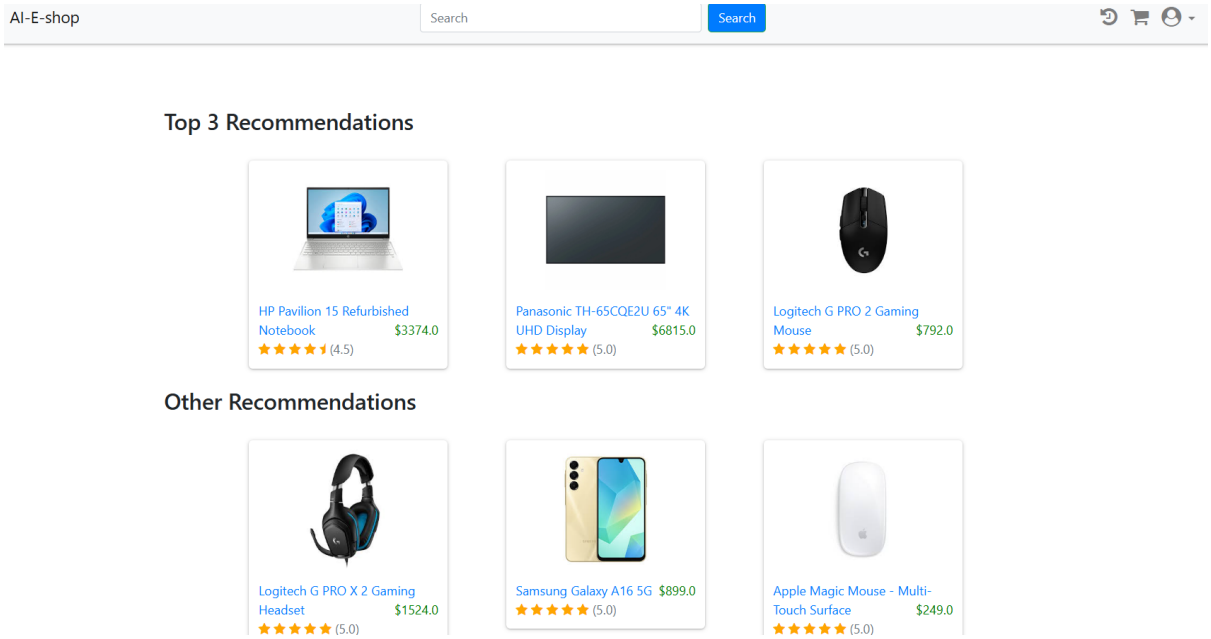


Figure 5.10 Recommended items

Flowchart below illustrates the user scenario.

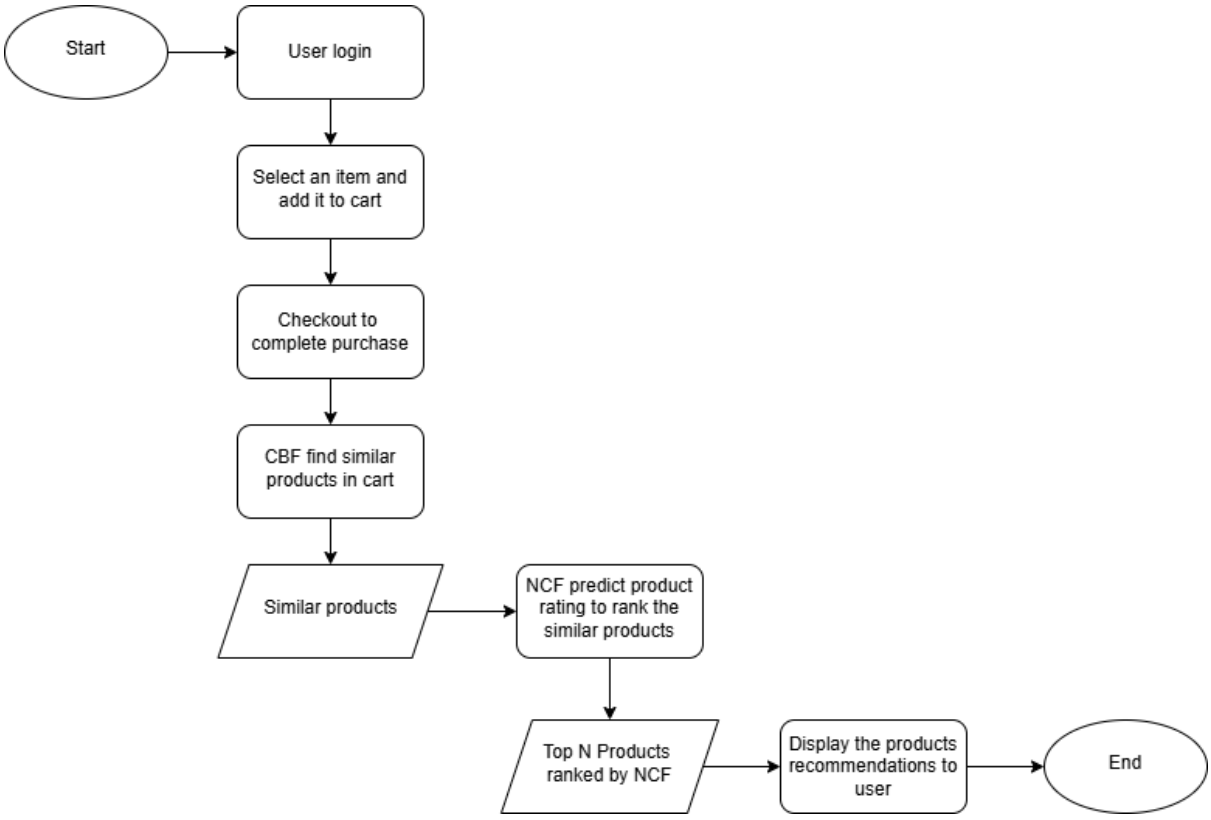


Figure 5.11 User scenario flowchart

5.4 Testing

The main purpose of testing is to ensure that all features and user interactions within the application work as expected. Functionality testing helps confirm that core processes such as user registration, login, adding items to the cart, product searching, placing orders, and receiving recommendations are working correctly.

To conduct functionality testing, all the functional requirements of the application are identified based on the system specifications or user stories. Each feature is tested using predefined inputs, and the resulting outputs are compared against expected results. The process involves using both valid and invalid input data to ensure the system responds appropriately in different scenarios.

The testing tool Selenium is used to send HTTP requests to the backend, simulate user actions, and capture the system's response. The outcomes are documented in test cases that specify the input, expected output, actual output, and whether the test passed or failed. This testing process helps identify functional bugs early and ensures that the system provides a smooth and reliable user experience.

Table 5.1 Test cases

Test Case ID	**Test Case Description**	**Steps to Execute**	**Expected Result**	**Actual Result**	**Status**
TC01	User Login	1. Go to login page 2. Enter correct email and	User successfully login	As expected	Pass

		password 3. Click "Login			
TC02	Add Item to Cart	1. Go to item listing 2. Select an item 3. Click "Add to Cart"	Item appears in user's cart	As expected	Pass
TC03	Successful Checkout	1. Go to cart 2. Click "Checkout"	Order appears in purchase history	As expected	Pass
TC04	View Personalized Recommendations	1. Login 2. Visit homepage	Personalized recommendations are shown based on history	As expected	Pass
TC05	Add Rating to Product	1. Visit purchase history 2. Select an item 3. Enter rating and Click "Submit Rating"	Product rating is updated	As expected	Pass

5.5 Deployment

To deploy the e-commerce prototype built with Flask, Railway was used as the deployment platform due to its simplicity and developer-friendly interface. It allows quick demonstrations of the recommendation model to users without the need to set up complex cloud infrastructure manually. The deployment process began by pushing the Flask project to a GitHub repository.

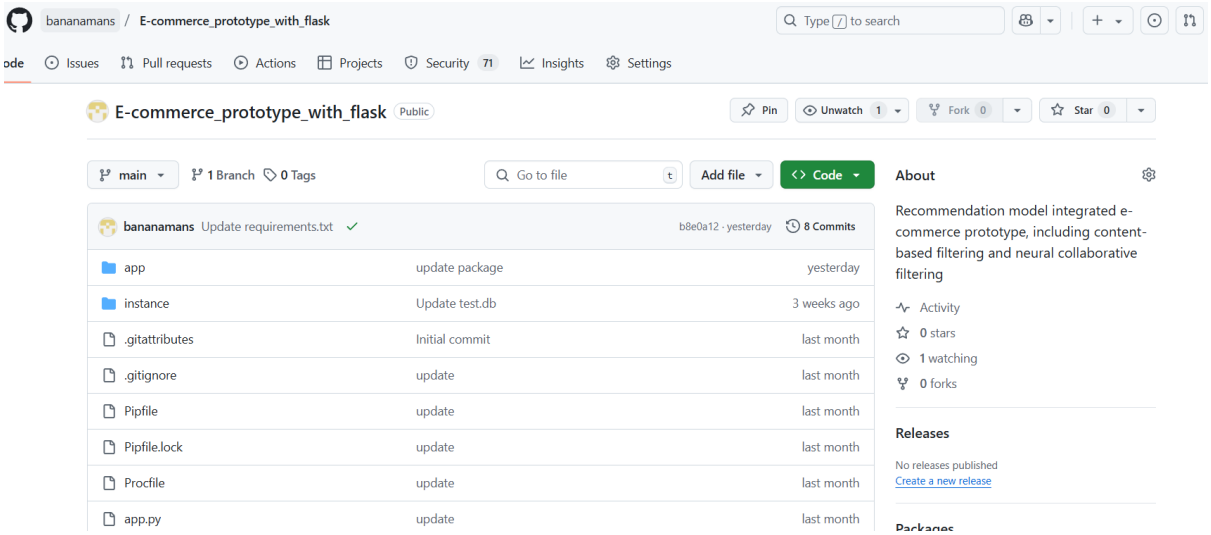


Figure 5.12 Pushing project to GitHub

On the Railway platform, a new project was created and connected to the GitHub repository. Railway automatically detected the Flask application and attempted to build and deploy it using the specified configurations (requirements.txt) for dependencies to define how the app should run. Environment variables such as the database URI and secret keys were configured through Railway's environment settings to ensure secure and proper runtime behavior.

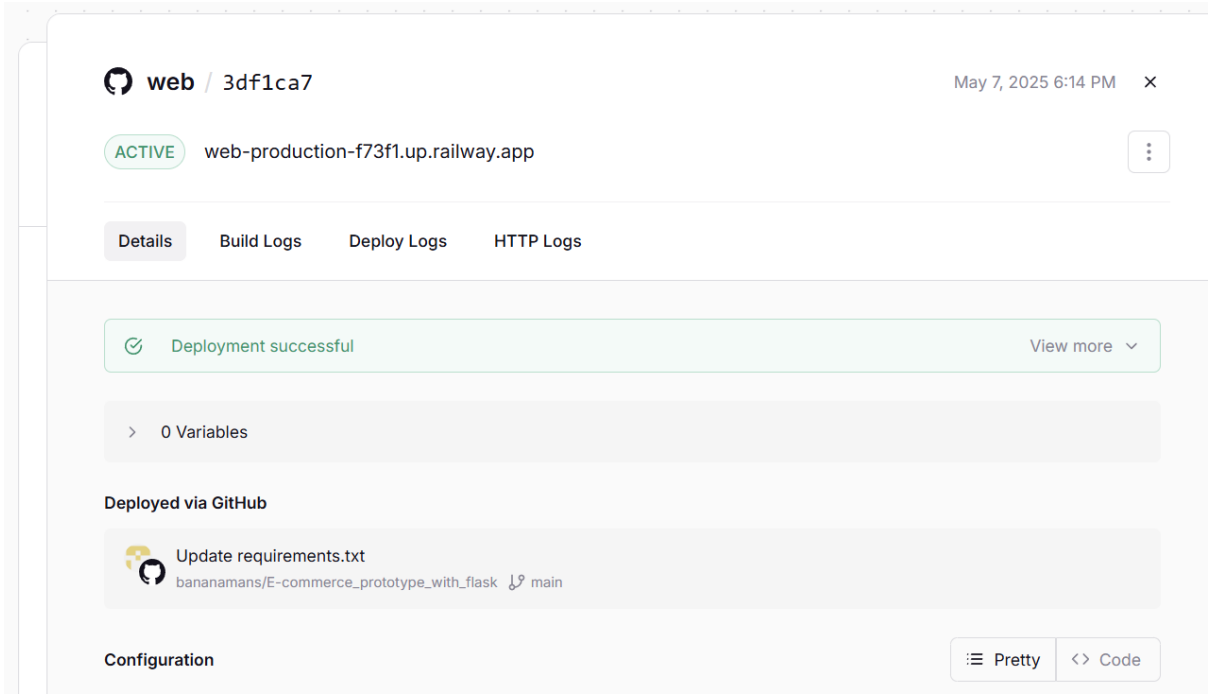


Figure 5.13 Deploy project on Railway

Once deployed, Railway provided a public URL through which the Flask-based e-commerce application became accessible online.

Railway also offered real-time logs and a console view, which made it easy to debug issues or monitor traffic.

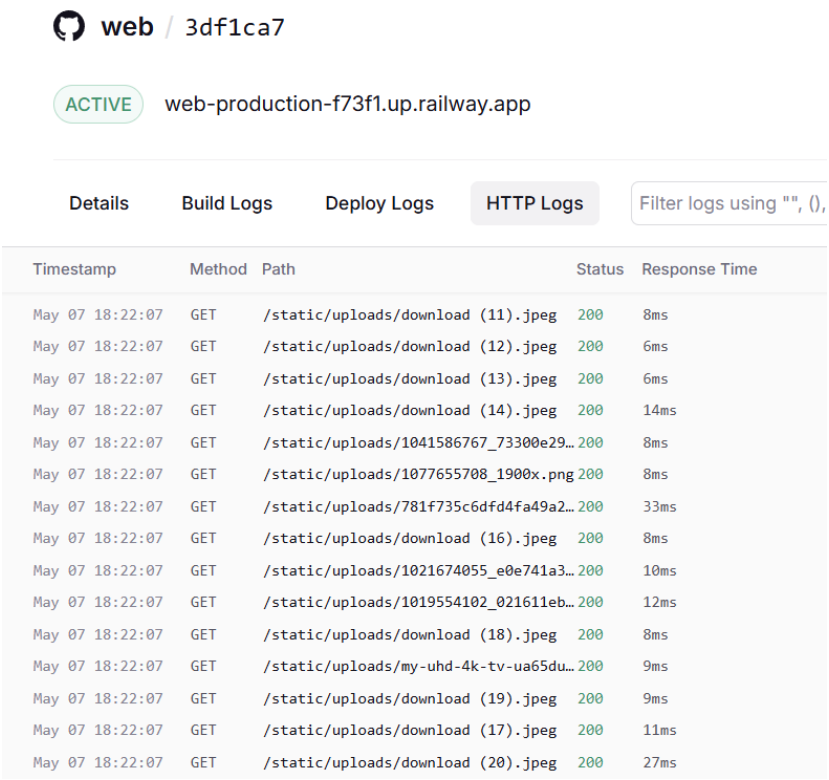


Figure 5.14 Log monitoring on Railway

With GitHub integration enabled, any changes pushed to the connected repository automatically trigger a new deployment. This ensured that the latest version of the application was always live without the need for manual redeployment.

5.6 Summary

The recommendation model was integrated into an e-commerce web application built with Flask. To make the application accessible, it was deployed using Railway, a cloud platform known for its ease of use and efficient deployment workflow. Overall, the project successfully demonstrated how hybrid recommendation systems can enhance user experience in e-commerce platforms.

Chapter 6: Conclusion and Future Work

6.1 Introduction

The final chapter wraps up the project by reviewing the main outcomes, limitations experienced, and potential future work. It also reflects on the integration of the hybrid recommendation model with the e-commerce prototype.

6.2 Achievements

Table 6.1 Achievements

Objective	**Achievement**
To develop a hybrid recommendation system using CBF and NCF.	Successfully implemented a hybrid recommendation model combining Content-Based Filtering (CBF) and Neural Collaborative Filtering (NCF), leveraging the strengths of both techniques to deliver more accurate and personalized product recommendations.
To evaluate the model using appropriate metrics.	Trained and tested the model using a real-world dataset. Evaluation was conducted using metrics such as Precision@K, Recall@K, and NDCG@K, demonstrating the model's superior performance compared to traditional methods like SVD.
To integrate the model within an e-commerce prototype.	The recommendation system was successfully integrated into an e-commerce prototype built with Flask, allowing users to receive personalized recommendations based on their past purchase behavior and item similarity.

6.3 Limitations

The primary limitation lies in the quality and accuracy of the dataset used. The dataset may contain missing, outdated, or imbalanced data, which can affect the model's ability to learn

meaningful patterns. Another constraint is the relatively small number of unique items and users in the dataset, which may restrict the diversity and scalability of recommendations. Furthermore, the system has not yet been tested with live user feedback, meaning its real-world performance and adaptability remain to be fully validated.

6.4 Future Works

Future works can focus on enhancing the dataset by collecting more diverse and higher-quality user interaction data. Incorporating additional contextual features such as time of interaction, user preferences, or browsing history may also help generate more personalized and relevant suggestions. Additionally, optimizing the system for scalability and performance in a real-world production environment would be beneficial, especially if deployed to serve a large user base. Finally, extending the e-commerce prototype with features like user reviews, item popularity trends, and interactive recommendation explanations could enhance user engagement and trust in the system.

6.5 Summary

This chapter concludes the project by highlighting the key achievements, such as the successful development and integration of a hybrid recommendation system into an e-commerce prototype. It also outlines the limitations faced, particularly regarding dataset quality and scale. Lastly, it suggests directions for future work, including improving data quality, enhancing model capabilities, and expanding the system’s functionality for better real-world applicability.

References

Amazon - Ratings (Beauty Products). (n.d.). Www.kaggle.com. https://www.kaggle.com/datasets/skillsmuggler/amazon-ratings

Amazon. (2024). Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs & more. Amazon.com; Amazon. https://www.amazon.com/

Barragáns-Martínez, A. B., Costa-Montenegro, E., Burguillo, J. C., Rey-López, M., Mikic-Fonte, F. A., & Peleteiro, A. (2010). A hybrid content-based and item-based collaborative filtering approach to recommend TV programs enhanced with singular value decomposition. *Information Sciences*, *180*(22), 4290–4311. https://doi.org/10.1016/j.ins.2010.07.024

Bhagat, M. D., & Chatur, P. N. (2023). A Study on Product Recommendation System based on Deep Learning and Collaborative Filtering. *2023 Third International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT)*, 1–5. https://doi.org/10.1109/ICAECT57570.2023.10117628

Cantador, I., Fernández, M., Vallet, D., Castells, P., Picault, J., & Ribière, M. (2008). A Multi-Purpose Ontology-Based Approach for Personalised Content Filtering and Retrieval. In M. Wallace, M. C. Angelides, & P. Mylonas (Eds.), *Advances in Semantic Media Adaptation and Personalization* (pp. 25–51). Springer. https://doi.org/10.1007/978-3-540-76361_2

Chirravuri, S. N. S. L., & Immidi, K. P. (2022). Major Challenges of Recommender System and Related Solutions. *International Journal of Innovative Research in Computer Science & Technology*, 10–18. https://doi.org/10.55524/ijircst.2022.10.2.3

Choi, S.-M., Lee, D., Jang, K., Park, C., & Lee, S. (2023). Improving Data Sparsity in Recommender Systems Using Matrix Regeneration with Item Features. *Mathematics*, *11*(2), Article 2. https://doi.org/10.3390/math11020292

Cindhamani, J., Pogaku, R., Sahana, M. P., Adnan, M. M., & Suganya, M. (2024). Hybrid Movie Recommendation System using Neural Collaborative Filtering with Gradient Decent and Association Rule Mining. *2024 First International Conference on Software, Systems and Information Technology (SSITCON)*, 1–5. https://doi.org/10.1109/SSITCON62437.2024.10796709

Dey, A. K., Chauhan, V. K., Singh, P. K., & Choudhury, P. (2022). LSTM-Based Top N Recommendation System using Cognitive Data. *2022 11th International Conference on System Modeling & Advancement in Research Trends (SMART)*, 93–98. https://doi.org/10.1109/SMART55829.2022.10046838

Ge, Q. (2022). E-Commerce Personalized Recommendation Based on Convolutional Neural Network. *2022 6th Asian Conference on Artificial Intelligence Technology (ACAIT)*, 1–5. https://doi.org/10.1109/ACAIT56212.2022.10137920

Ibrahim, M., Bajwa, I. S., Sarwar, N., Hajjej, F., & Sakr, H. A. (2023). An Intelligent Hybrid Neural Collaborative Filtering Approach for True Recommendations. *IEEE Access*, *11*, 64831–64849. IEEE Access. https://doi.org/10.1109/ACCESS.2023.3289751

Im, I., & Hars, A. (2007). Does a one-size recommendation system fit all? The effectiveness of collaborative filtering based recommendation systems across different domains and search modes. *ACM Trans. Inf. Syst.*, *26*(1), 4-es. https://doi.org/10.1145/1292591.1292595

Ko, H., Lee, S., Park, Y., & Choi, A. (2022). A Survey of Recommendation Systems: Recommendation Models, Techniques, and Application Fields. *Electronics*, *11*(1), Article 1. https://doi.org/10.3390/electronics11010141

Li, C., Ishak, I., Ibrahim, H., Zolkepli, M., Sidi, F., & Li, C. (2023). Deep Learning-Based Recommendation System: Systematic Review and Classification. *IEEE Access*, *11*, 113790–113835. IEEE Access. https://doi.org/10.1109/ACCESS.2023.3323353

Li, G., & Zhang, X. (2023). Research on the Recommendation System Combining Bi-LSTM and NCF Algorithms. *2023 11th International Conference on Information Systems and Computing Technology (ISCTech)*, 402–407. https://doi.org/10.1109/ISCTech60480.2023.00080

Liliana, D. Y., Nalawati, R. E., Iswara, R. W., Aisyah, A. P., & Malo, H. O. T. (2024). Book Recommender System Using Content-Based Filtering for PNJ Press Website. *2024 10th International Conference on Education and Technology (ICET)*, 231–236. https://doi.org/10.1109/ICET64717.2024.10778466

Putri, A., Abdurahman Baizal, Z. K., & Rischasdy, D. (2022). Book Recommender System using Convolutional Neural Network. *2022 International Conference on Advanced Creative Networks and Intelligent Systems (ICACNIS)*, 1–6. https://doi.org/10.1109/ICACNIS57039.2022.10055753

Qalbyassalam, C., Rachmadi, R. F., & Kurniawan, A. (2022). Skincare Recommender System Using Neural Collaborative Filtering with Implicit Rating. *2022 International Conference on Computer Engineering, Network, and Intelligent Multimedia (CENIM)*, 272–277. https://doi.org/10.1109/CENIM56801.2022.10037471

Raheem, K. R., & Ali, I. H. (2020). Content-based Recommender System Improvement using Hybrid Technique. *2020 1st. Information Technology To Enhance e-Learning and Other Application (IT-ELA*, 54–59. https://doi.org/10.1109/IT-ELA50150.2020.9253117

Salter, J., & Antonopoulos, N. (2006). CinemaScreen recommender agent: Combining collaborative and content-based filtering. *IEEE Intelligent Systems*, *21*(1), 35–41. IEEE Intelligent Systems. https://doi.org/10.1109/MIS.2006.4

Singla, R., Gupta, S., Gupta, A., & Vishwakarma, D. K. (2020). FLEX: A Content Based Movie Recommender. *2020 International Conference for Emerging Technology (INCET)*, 1–4. https://doi.org/10.1109/INCET49848.2020.9154163

Subhan, S., Syarif, D. L., Widhihastuti, E., Rakainsa, S. K., Sam'an, M., & Ifriza, Y. N. (2025). Improved recommender system using Neural Network Collaborative Filtering (NNCF) for E-commerce cosmetic product. *SINERGI*, *29*(1), 155–162. https://doi.org/10.22441/sinergi.2025.1.014

Trabelsi, F., Khtira, A., & El Asri, B. (2021). Hybrid Recommendation Systems: A State of Art: *Proceedings of the 16th International Conference on Evaluation of Novel Approaches to Software Engineering*, 281–288. https://doi.org/10.5220/0010452202810288

Tran, P. H., Nguyen, H. T., & Nguyen, N.-T. (2020). A Hybrid Approach for Neural Collaborative Filtering. *2020 7th NAFOSTED Conference on Information and Computer Science (NICS)*, 368–373. https://doi.org/10.1109/NICS51282.2020.9335910

Wang, W., Ye, C., Yang, P., & Miao, Z. (2020). Research on Movie Recommendation Model Based on LSTM and CNN. *2020 5th International Conference on Computational Intelligence and Applications (ICCIA)*, 28–32. https://doi.org/10.1109/ICCIA49625.2020.00013

Wang, Y. (2024). Recommendation Algorithms for Rural E-Commerce Business on Basis of Big Data and Machine Learning Technology. *2024 International Conference on Data Science and Network Security (ICDSNS)*, 1–5. https://doi.org/10.1109/ICDSNS62112.2024.10691257

Yarahmadi Gharaei, N., Dadkhah, C., & Daryoush, L. (2021). Content-based Clothing Recommender System using Deep Neural Network. *2021 26th International Computer Conference, Computer Society of Iran (CSICC)*, 1–6. https://doi.org/10.1109/CSICC52343.2021.9420544

Zhang, L., & Zhang, L. (2021). Top-N recommendation algorithm integrated neural network. *Neural Computing and Applications*, *33*(9), 3881–3889. https://doi.org/10.1007/s00521-020-05452-y