Movie Recommendation System : A Comparative Analysis Of Machine Learning And Deep Learning Models

Dissertation Submitted by
Prashant Neupane

Under Supervision of
Anu Bala

York St Joh University - London Campus
Department Of Computer Science
November 2024

Declaration

I, Prashant Neupane, declare that this report which has been submitted for assessment is my own work, has been written in my own words and no part of it has been previously submittedffor any other assessments. Any use of other authors work, whether that be quotes, images,ideas, statistics etc. are correctly acknowledged and listed in the references section at the end of this report.

Acknowledgements

I would firstly like to thank Anu Bala for her guidance, supervision and support throughout this project. I would also like to thank all the staff on the Computer Science course at York St John University for giving me advice, encouragement, guidance and most importantly for sharing their knowledge throughout the project and throughout my degree.

I would also like to thank my parents for their support and confidence in my decision to return to education and their encouragement throughout the degree.

I would lastly like to thank my partner, Esther and my friends who all helped encourage me to do the best I could and who offered a happy distraction outside of my research and studies.

Table Of Contents

List of Figures

List of Tables

List of Abbreviations

DL	Deep Learning
ML	Machine Learning
SVD	Singular Value Decomposition
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
MAE	Mean Absolute Error
NCF	Neural Collaborative Filtering
CBF	Content-Based Filtering
KNN	K-Nearest Neighbours

Abstract

Recommendation systems are essential for delivering customized content in domains such as social networking, entertainment, and e-commerce. In particular, movie recommendation systems are essential for improving user experience by making pertinent film recommendations based on user preferences and historical behavior. The goal of this study is to balance accuracy and computational efficiency by examining the performance of both sophisticated deep learning (DL) and conventional machine learning (ML) models for movie suggestions. The project analyzes four models—KNNWithMeans, SVD, Neural Collaborative Filtering (NCF), and the Transformer Model—that are selected to reflect both traditional ML approaches and contemporary DL architectures using a well-known dataset of user-movie interactions. An 80:20 train-test split was used to train the models after the dataset was preprocessed by encoding movie genres using one-hot vectors. Metrics including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and training time were used to assess the models. Although KNNWithMeans had the quickest training time (0.56 seconds), its accuracy was lower, as seen by its greatest MAE of 0.695. In contrast, the SVD model, which had an MSE of 0.753, RMSE of 0.868, and MAE of 0.666, demonstrated a balance between accuracy and computing efficiency. Superior accuracy was shown by deep learning models such as Transformer and NCF; the Transformer model achieved an MSE of 0.835 and an MAE of 0.686, but at the expense of a noticeably longer training time (92.82 seconds). With a training time of 35.99 seconds and an MAE of 0.694, the NCF model likewise demonstrated remarkable accuracy. These findings demonstrate the trade-off between computing demands and accuracy. Despite their increased accuracy, deep learning models are computationally demanding and less useful for resource-constrained real-time applications. The significance of choosing the right model based on particular use case requirements and available computing power is emphasized by this effort. In order to improve accuracy and efficiency, future research should investigate hybrid models that combine the best features of ML and DL approaches.

Keywords: Recommendation systems, Deep Learning (DL), Machine Learning (ML), Neural Collaborative Filtering (NCF), Transformer, Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE)

Chapter 1: Introduction

1.1 Background

For over a century, movies have been a vital source of entertainment, with a vast array of genres and storylines to suit a diverse spectrum of interests and tastes. Movies are now more widely available than ever thanks to the introduction of digital streaming services, giving moviegoers an unheard-of selection. Finding films that suit your tastes among the plethora of possibilities is a unique difficulty that comes with this wealth of information. In order to solve this issue, recommendation systems employ user information, such as search terms, reviews, and watching preferences, to offer tailored movie recommendations. Through the assessment and prediction of user preferences, these systems enhance the viewing experience by helping individuals find new films that align with their interests (Jannach and Adomavicius, 2016).On websites like Netflix, Amazon Prime, and Hulu, where algorithms are essential to differentiating their offerings from competitors and attracting new users, recommendation systems are particularly apparent. In addition to maintaining customer satisfaction, these technologies are critical for promoting platform loyalty and user retention. Advanced algorithms analyze user behavior and interaction patterns to produce tailored suggestions that cater to individual preferences. Given that customer expectations are always shifting in this cutthroat market, offering a personalized and interesting experience is crucial for retaining subscribers (Ricci et al., 2015). According to Dean (2024), Netflix's subscriber base increased significantly from 400,000 in 2001 to 260.28 million in 2023. From $3.1 billion in 2011 to $31.6 billion in 2022, the company's revenue increased dramatically, demonstrating the value of customized recommendation algorithms and the necessity for customized content distribution (Gill, 2024).

Recommendation systems combine user profiles, item data, and sophisticated algorithms to match users with potential films (Anwar et al., 2020). Each person's profile contains their interests and past interactions, and each item, such as a film, has characteristics such as genre, director, and cast. These systems then use a range of ml and dl techniques, like as matrix factorization or collaborative filtering, to link users to products in order to predict what a user might like. For example, Netflix's recommendation engine predicts which films or TV shows a user is likely to watch based on their ratings and viewing patterns.

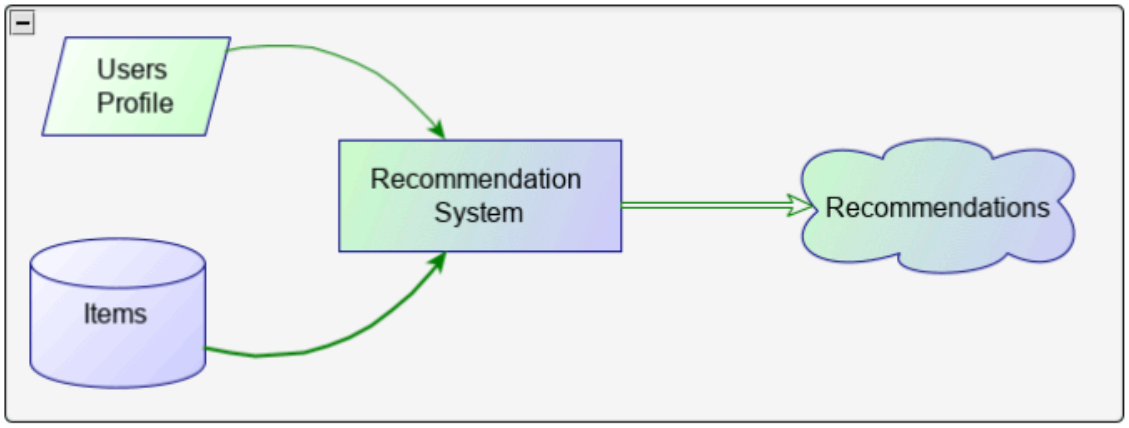


Fig 1: Recommender system process (Anwar et al., 2020)

The combination of ML and DL technology has brought about a huge transformation in the field of recommendation systems. Traditional machine learning techniques, such as matrix factorization and collaborative filtering, have been essential in identifying user preferences and generating accurate predictions. However, the introduction of DL methods has elevated these skills to new levels. Transformer-based architectures and Neural Collaborative Filtering (NCF) models offer advanced methods for simulating and interpreting intricate correlations found in user data. These techniques provide more precise and user-specific recommendations by spotting complex patterns and interactions that conventional approaches could overlook. As the amount and complexity of data continue to increase, using such cutting-edge technology is becoming increasingly important for producing well-informed suggestions and raising user satisfaction (He et al., 2017). This advancement highlights the need for recommendation systems to leverage state-of-the-art technologies in order to remain competitive in the quickly changing digital market.

1.2 Motivation

The proliferation of digital content, especially movies and TV shows, is making it more and more difficult for users to locate content. Even while streaming providers offer a wide variety of options, viewers often struggle to find content that fits their particular tastes. Recommendation systems have become essential in this context, offering consumers personalized suggestions that enhance their experience by sorting through vast amounts of content and presenting choices according to individual tastes. Ricci et al. (2015) claim that personalization boosts user satisfaction and platform engagement.

The advancement of DL and ML technology presents an opportunity to further refine and enhance recommendation systems. Traditional methods sometimes struggle to manage the volume and complexity of modern data, but machine learning and deep learning algorithms can model intricate relationships and patterns in user behavior to generate more relevant and accurate recommendations. By employing these state-of-the-art technologies,

recommendation systems can enhance user experience and retention rates in a fiercely competitive digital environment, better meeting users' evolving expectations (He et al., 2017).

1.3 Aims and Objectives

The primary objective of this dissertation is to advance the field of movie recommendation systems by developing and evaluating novel algorithms that significantly increase precision and personalization. Systems that can handle these challenges and provide personalized recommendations are essential given the abundance of content available on streaming platforms and the growing complexity of user preferences. This research attempts to meet this need by creating increasingly sophisticated and accurate recommendation systems through the application of state-of-the-art machine learning (ML) and deep learning (DL) approaches.

The aims and objectives of the dissertation are

- to use deep learning and machine learning methods to develop sophisticated movie recommendation algorithms.
- to determine the optimal strategy by evaluating and contrasting performance using measures like as MSE and RMSE.
- to assess the most advanced recommendation systems in order to identify issues and offer solutions.

1.4 Scope

The development and optimization of movie recommendation systems are examined in this study using contemporary ml and dl methodologies. In order to maximize accuracy and personalization, the scope of the work includes the creation and use of cutting-edge recommendation techniques, including hybrid models, matrix factorization, and neural collaborative filtering. The study will conduct a comprehensive performance evaluation of these algorithms using metrics such as MSE and RMSE to see how well they perform in offering tailored recommendations. In addition to creating algorithms, the research will employ user-specific and contextual data to make recommendations more relevant. This means looking into how various components, such as user behavior patterns and contextual cues, could be used to provide recommendations for more personalized content. Along with discussing the primary issues and challenges with the current recommendation systems, the dissertation will offer analysis and recommendations for improving them. By applying these methods to a large dataset, the study aims to produce reliable and practical results that can advance the field of recommendation systems and enhance user experiences in the digital entertainment industry.

1.5 Research Method

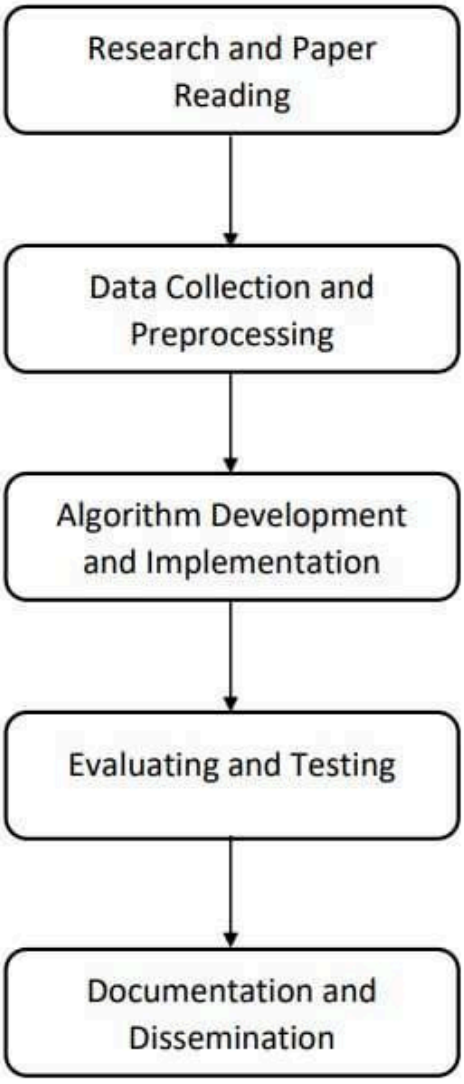


Fig 2: Block Diagram of Research Method

The research methodology used in this dissertation involves the systematic development and evaluation of movie recommendation algorithms. The first phase included a comprehensive literature review that aided in understanding current methodologies and identifying research gaps. Throughout the review process, a range of academic sources were explored to learn more about different recommendation systems and their uses. Following the literature research, data preparation and collection were completed. Relevant datasets and other information were acquired from online sources. To ensure that the data was suitable for developing and assessing recommendation algorithms, it was then cleaned and prepared for analysis.

The creation and application of multiple recommendation systems was the main focus of the study. These explored both traditional and innovative approaches to look into different tactics for improving suggestion accuracy. The effectiveness of these algorithms was compared and

their performance was carefully evaluated using pre-established parameters. The final stage was to document the research findings and prepare them for dissemination. This included summarizing the research findings, presenting them, and sharing them with the academic community. The entire process was designed to ensure a thorough analysis and effective communication of the research results.

1.6 Ethics And Relevance

Every step of this dissertation was conducted with ethical issues in mind, particularly with regard to the handling and use of data. By following ethical guidelines, confidentiality and privacy were protected during the entire data collection and analysis process. Data protection regulations were followed in the management of the MovieLens and other sources' datasets used in this investigation. Measures were taken to anonymize user information to guard against any misuse. Additionally, the research process was conducted in an ethical manner, avoiding any practices that would skew or produce biased data. Before starting the study, ethical approval was sought when necessary, and it was conducted in compliance with recognized academic research protocols.

The potential for this dissertation to advance the subject of movie recommendation systems—which are becoming increasingly important in the digital age—makes it pertinent. Given the rapid growth of streaming services and the wealth of available content, there is an urgent need for efficient recommendation algorithms that enhance user satisfaction and experience. This work develops and evaluates advanced ml and dl algorithms to increase recommendation accuracy and customisation. Future research opportunities in the field of recommendation systems are highlighted, along with the study's practical implications for streaming platforms and content providers. The study not only tackles current problems but also establishes the foundation for further development and innovation in the area.

Chapter 2: Literature Review

2.1 Introduction to Recommender Systems

In today's digital platforms, recommender systems are essential because they affect how users find and engage with content in a variety of settings, such as social media, news aggregation, streaming services, and e-commerce. These systems make personalized recommendations that boost user satisfaction and engagement by looking at user behavior and preferences. The three primary methods employed in recommender systems—Collaborative Filtering (CF), Content-Based Filtering (CBF), and Hybrid Methods—each address the challenge of personalization differently. Based on the assumption that people with similar past behaviors will exhibit similar preferences, collaborative filtering is an effective method for uncovering latent patterns in user interactions.

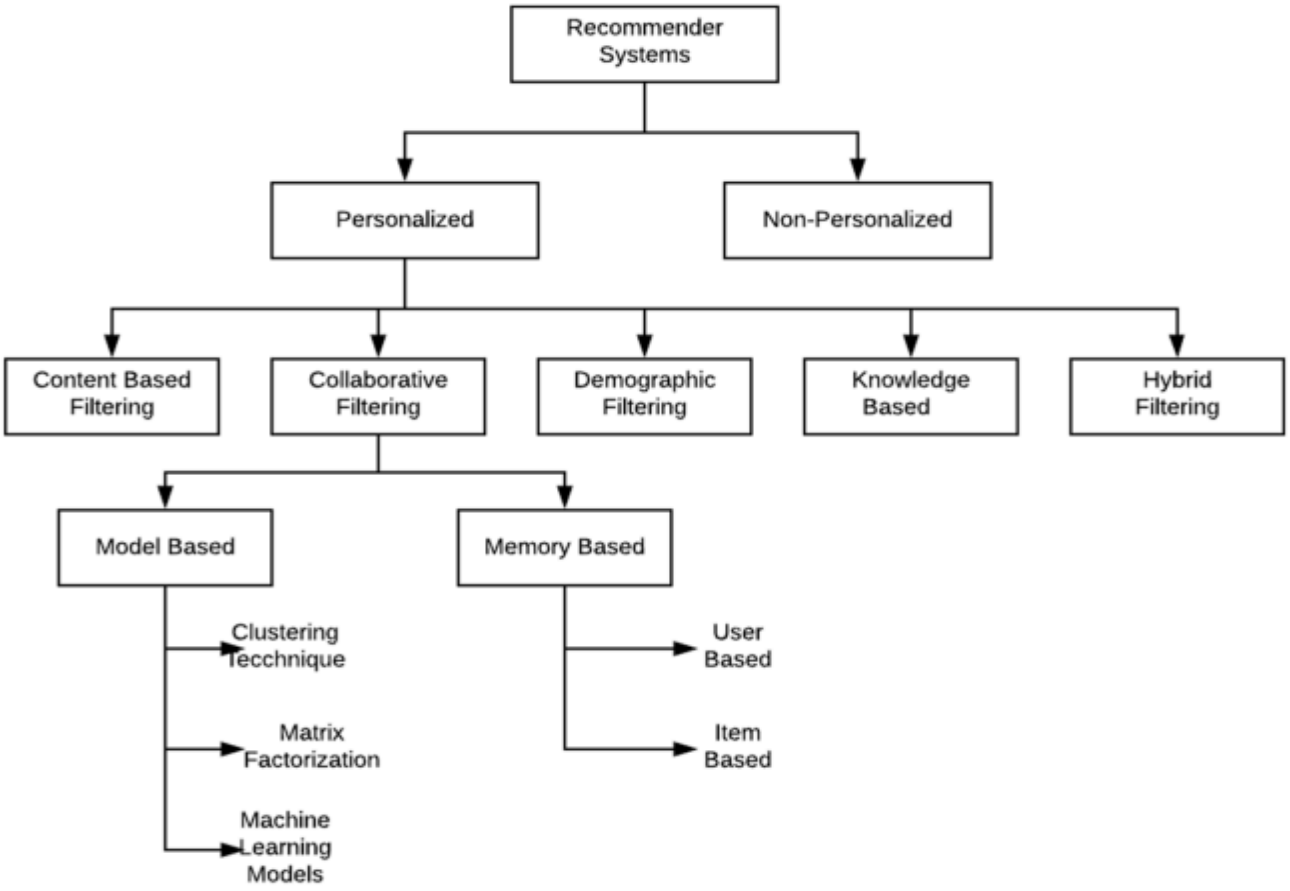


Fig 3: Various Approaches For Recommendation System (Thakker et al., 2021)

In contrast, CBF suggests products based on item attributes and user profiles. By examining the characteristics of products and contrasting them with the user's prior preferences, this method makes suggestions based on specific aspects rather than just user interactions. By

combining collaborative and content-based filtering, hybrid approaches aim to maximize suggestion accuracy while addressing the drawbacks of each strategy. In an attempt to offer more insightful and successful recommendations, hybrid systems combine a variety of data sources and methodologies.

The field of recommender systems has changed significantly since the advent of ML and DL approaches, which have raised the bar for accuracy and complexity. Traditional machine learning methods like k-nearest Neighbors (kNN), Singular Value Decomposition (SVD), and Non-negative Matrix Factorization (NMF) have been crucial to the development of recommendation systems. For instance, while kNN uses distance metrics to find similarities between individuals or items, SVD and NMF deconstruct user-item interaction matrices into latent components to uncover more specific information about user preferences. Notably, SVD++ enhances standard SVD and boosts prediction accuracy by incorporating implicit input. The advent of Factorization Machines (FM), which characterize high-dimensional interactions in sparse datasets, has advanced the area.The introduction of deep learning has brought about revolutionary changes with techniques like Wide and Deep Models, which combine linear and non-linear learning to capture both memorization and generalization, and Neural Collaborative Filtering (NCF), which uses neural networks to simulate complex user-item interactions. Transformer Models have shown remarkable efficacy in handling sequential and contextual data in recommendations, despite their initial development for natural language processing. Despite these advancements, recommender systems still face challenges with scalability, data sparsity, and ethical quandaries involving algorithmic bias and user privacy. The development and current state of recommender systems are examined in this literature review, which also looks at recent discoveries and methods to provide a comprehensive understanding of the field's successes, challenges, and possible future opportunities.

2.2 Collaborative Filtering (CF)

CF is a well-liked method in recommender systems that predicts user preferences by using data from user interactions. The fundamental idea behind CF is that users who have previously shown similar preferences are likely to do so again in the future. CF methods can be divided into two main categories: user-based and item-based procedures. This approach identifies individuals who are similar to the target user based on their prior preferences. Recommendations are generated by combining the tastes of users who are similar to one other. For example, if User A and User B have quite comparable ratings for some items, User B's ratings are used to propose User A's unrated items. Instead of searching for similar users, this approach looks for products that are similar to ones the buyer has already appreciated. Recommendations are based on the similarity between the two commodities rather than the consumers. For instance, depending on historical performance, the algorithm may recommend similar products to item X if a customer finds it intriguing.By uncovering latent correlations in user-item interaction data, recommendation systems commonly use matrix factorization algorithms to predict user preferences. These methods decompose a user-item matrix (rows = users, columns = items, values = ratings) into smaller matrices that reflect

latent attributes for both users and items. Singular Value Decomposition (SVD), one of the basic methods, involves factorizing the original matrix into three matrices that reflect users, things, and their latent attributes. By using these latent components to rebuild the original matrix, it becomes possible to forecast ratings that are not there. FunkSVD uses stochastic gradient descent to iteratively update the latent components, lowering the processing overhead of classical SVD without sacrificing accuracy. SVD++ is an enhancement of SVD that includes implicit feedback, like clicks and views, as well as explicit user ratings. SVD++ uses this additional information to provide more accurate recommendations, particularly in the absence of sufficient explicit evaluations. This approach improves the ability to predict user preferences by altering the latent components based on both observable and unseen interactions. This approach improves the ability to predict user preferences by altering the latent components based on both observable and unseen interactions. The complex interactions between users and things are fundamentally captured by matrix factorization, which makes recommendation creation cost-effective even for enormous datasets.

The user-item interaction matrix is divided into latent elements using a matrix factorization method known as SVD. SVD++ improves this even further by adding implicit feedback, or data about user preferences that isn't assessed explicitly, including views, clicks, and past purchases. According to Koren et al.'s investigation, SVD++ obtained an RMSE of 0.879 on the Netflix dataset. This is a significant improvement over the typical SVD, which yielded an RMSE of 0.897. More accurate predictions are produced by SVD++'s improved capacity to identify latent patterns in user-item interactions thanks to implicit feedback. By enabling the model to leverage a wider range of user behaviors, implicit feedback improves recommendation accuracy and fortifies the system's resilience to sparse data.FunkSVD is a specific SVD application that enhances the matrix factorization process by optimizing the decomposition of the user-item matrix. This approach is particularly well-known for the effectiveness and efficiency of its real-world applications. Using the Netflix dataset, Funk's study shown that FunkSVD could reduce RMSE to 0.955 instead of 1.019 when employing simpler matrix factorization methods. FunkSVD works better than traditional methods in detecting latent factors that influence user preferences, as evidenced by the improved RMSE. FunkSVD's ability to reduce RMSE, a measure of its potential to generate more accurate predictions, makes it a suggested choice for practical recommendation systems.Factorization Machines (FM), which characterize the interaction between variables in sparse, high-dimensional datasets, are an extension of matrix factorization. Unlike traditional matrix factorization methods, FMs can capture complex relationships between characteristics, which makes them perfect for scenarios with sparse data. Rendle's FM model was assessed using the Yahoo! Music dataset, yielding an RMSE of 0.955. This performs better than traditional matrix factorization methods, which often have RMSE values around 1.00. The ability of FMs to handle sparse and complicated interactions allows for a more nuanced knowledge of user preferences. Due to their adaptability in simulating high-dimensional interactions, FMs offer a dependable solution for CF limitations, particularly in datasets with complicated and sparse user-item interactions.

The cold-start issue is a significant barrier to collaborative filtering systems. This issue arises when the system encounters new individuals or products with insufficient interaction data.For instance, when a new film is added to a recommendation system, CF approaches struggle to generate trustworthy recommendations until enough people have engaged with the film and provided ratings or comments. Likewise, new users will not be able to get personalized recommendations unless they have interacted with the system enough to build a preference profile. The effectiveness of CF approaches is limited by the cold-start problem, particularly in the early stages of adding new users or objects. This could result in subpar suggestions and an unsatisfactory user experience. One frequent solution to the cold-start problem is to use hybrid approaches that combine CF and content-based filtering with other tactics. For example, hybrid systems may use other data sources, like item metadata or user profiles, to produce initial recommendations before enough interaction data is collected (Schein et al., 2002). Another tactic to mitigate the impact of cold-start issues is to use demographic information or make suggestions based on popularity.

2.3 Content-Based Filtering (CBF)

The CBF approach is used by recommender systems to suggest items based on the attributes of the objects and the user's preferences. CBF places more emphasis on an item's intrinsic traits and how well it fits the user's profile than Collaborative Filtering, which bases recommendations on data about user interactions. This strategy leverages certain item features, such genre, keywords, or metadata, and compares them to those of things that the customer has previously shown interest in. By integrating these attributes to create a profile of the user's tastes, CBF is able to recommend products that fit with the user's pre-existing interests. For example, if a user frequently interacts with "science fiction" items, a CBF system will prioritize proposing other scientific fiction items based on their features over user interactions or ratings alone.

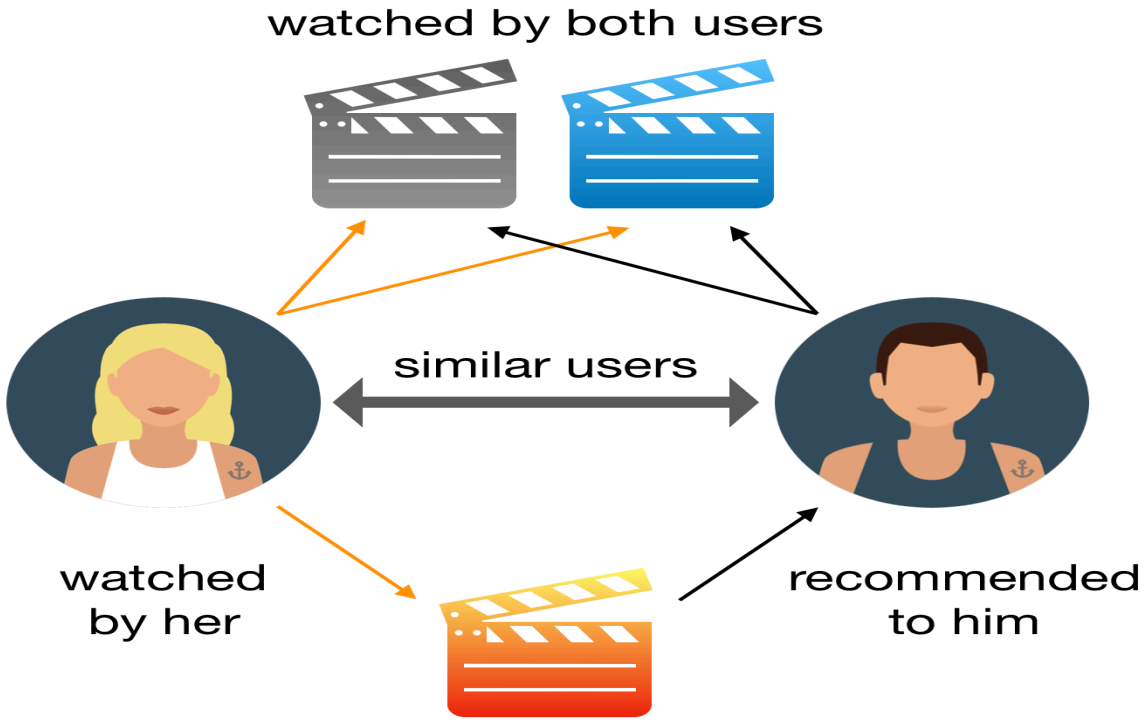


Fig 4: Content-Based Recommendation System (Otten, 2024)

Autoencoders, a type of neural network used for unsupervised learning, are used to learn compressed representations of item information. Chen et al. introduced an autoencoder-based feature extraction method that improved the accuracy of content-based recommendations. By identifying complex patterns and correlations in item qualities, autoencoders can significantly enhance the quality of suggestions. Chen et al.'s research yielded a Mean Absolute Error (MAE) of 0.65 on the MovieLens dataset, a significant improvement over the MAE of 0.72 achieved by conventional methods. Autoencoders allow for more complex feature extraction and better handling of high-dimensional data, which improves the overall performance of content-based filtering. CNNs have been adapted to extract features from textual item descriptions for content-based recommendation systems. CNNs are renowned for their efficiency in tasks involving text and picture processing. Using CNNs to process and analyze item descriptions, Tang et al. achieved an MAE of 0.60 on the Amazon product dataset. This outperformed conventional content-based models, whose MAEs were generally about 0.68. CNNs are particularly adept at handling complex and high-dimensional text input, which improves the relevance and accuracy of recommendations and enables more effective feature extraction.DeepCoNN is a creative approach to fusing deep learning techniques with content-based filtering. Using deep neural networks to learn both object and user representations, this approach enhances recommendations by capturing complex interactions between users and items. Zheng et al. demonstrated that DeepCoNN generated an RMSE of 0.88 on the MovieLens dataset, a notable improvement over the RMSE of 0.95 achieved with traditional content-based techniques. By deftly combining the advantages of content-based and deep-learning approaches, the hybrid model improves the system's ability to understand and predict user preferences more accurately.

The potential for overspecialization is one of the main disadvantages of content-based filtering. Since CBF systems recommend items that are exactly the same as those the customer has already used, it's possible that they inadvertently limit the variety of options. For instance, a fan of action films would observe that the majority of the suggestions are action-oriented, thereby ruling out other genres that the user might enjoy. This lack of diversity could result in fewer recommendations that don't always take into account the user's shifting interests or preferences, claim Pazzani and Billsus (2007). To address this issue, it is often required to add more methods or employ hybrid approaches to balance personalization with exploring new item categories.

2.4 Hybrid Methods

Hybrid recommender systems combine many recommendation techniques to maximize the benefits of each strategy while reducing its disadvantages. By combining CF and CBF, hybrid techniques can produce suggestions that are more accurate and diversified. These systems often integrate item features, user interactions, and sometimes additional data sources to enhance suggestion quality.

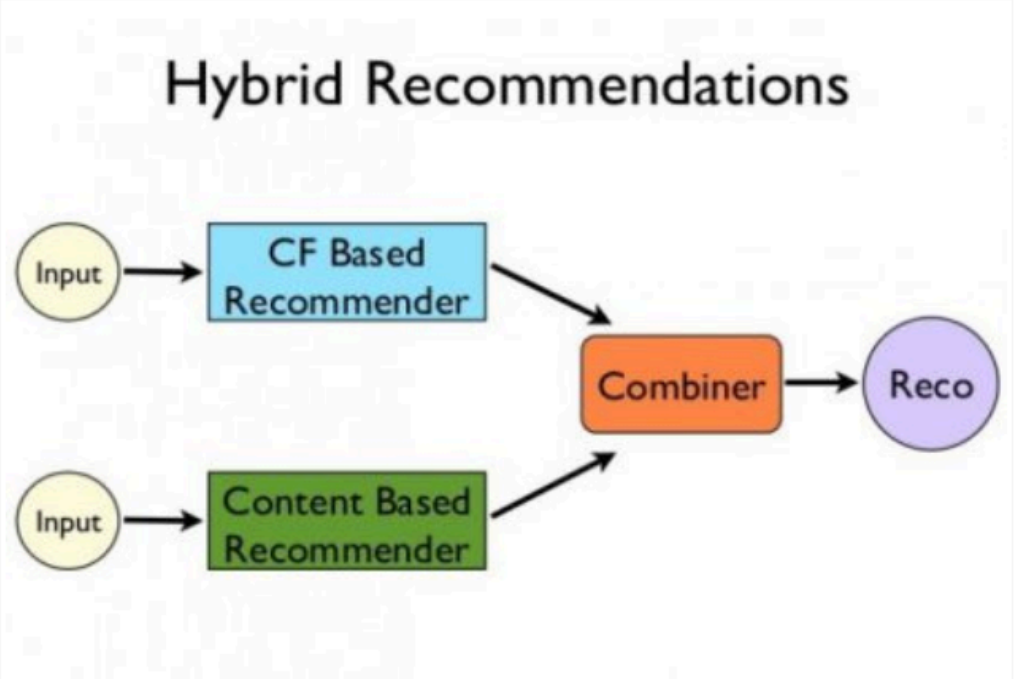


Fig 5: Hybrid Recommendation System

Burke handles hybrid recommendation through layered generalization, which employs a meta-learner to aggregate predictions from many models. This approach improves suggestion accuracy and strengthens the recommendations by combining the output of multiple recommendation systems. Burke demonstrated that stacked generalization performed better than non-hybrid methods on the MovieLens dataset, with an RMSE of 0.87 as opposed to 0.92 for non-hybrid methods. This method effectively integrates predictions from many models, capturing a greater variety of item attributes and user preferences.Zheng et al. introduced a model-based hybrid strategy that combines collaborative and content-based filtering using a blending framework. Their approach fared better than the solo CF (RMSE = 0.89) and CBF (RMSE = 0.88) models on the Netflix dataset, with an RMSE of 0.83. By combining CF and CBF, this hybrid approach balances each system's advantages while addressing its shortcomings. It effectively blends user activity data with item features to produce more accurate and varied suggestions.

Complexity and Computation: Complex models that need a lot of processing power are commonly used in hybrid techniques. Integrating many recommendation systems and managing their interactions may be resource-intensive and challenging to scale. Due to their growing complexity, hybrid systems require sophisticated algorithms and efficient processing techniques to guarantee performance and scalability, particularly when dealing with large datasets and real-time recommendations.

2.5 Machine Learning And Deep Learning Approaches

By increasing accuracy, relevance, and adaptability, machine learning and deep learning approaches have significantly improved recommender systems. User-item patterns are revealed by conventional techniques like KNN, NMF, and SVD; NMF performs best on non-negative datasets, whereas SVD handles sparse data with extensions like SVD++. Similarity measurements are used by kNN, and Item-KNN and User-KNN allow for customized recommendations. Neural Collaborative Filtering (NCF), which records intricate, non-linear user-item interactions for tailored recommendations, is another example of how deep learning has further transformed the area. Combining deep networks and linear models, wide and deep learning strikes a compromise between specialized and broad recommendations. Transformers use self-attention to process contextual and sequential input and provide context-aware, adaptive recommendations. When combined, these methods improve recommendations in large-scale systems, handle data sparsity, and spot complex patterns.

Li et al. focused on Item-KNN and User-KNN approaches while examining kNN variations. The RMSE for item-KNN, which gauges similarities between things, and user-KNN, which gauges similarities between users, on the MovieLens dataset, were 0.89 and 0.92, respectively. The findings demonstrate that Item-KNN is generally more successful at gathering item similarities and providing accurate suggestions due to its capacity to utilize item-specific data more successfully than user-centric approaches.Yang et al. presented regularization strategies to increase NMF's ability to handle sparse data. Their regularized NMF model outperformed traditional NMF approaches, which had RMSE values of approximately 0.92, on the Yahoo! Music dataset, yielding an RMSE of 0.87. Regularization prevents overfitting by imposing constraints on the factorization process, which enhances the model's capacity to generalize and handle sparse datasets more adeptly.BPR's primary objective is to assign ratings to products based on implicit input, such as clicks or purchases. Rendle et al. demonstrated that BPR could achieve a precision of 0.78 and a recall of 0.60 using the MovieLens dataset. The fact that BPR may evaluate objects based on user preferences inferred from behavior rather than vocal ratings, providing a more nuanced knowledge of user interests, demonstrates its effectiveness in implicit feedback contexts.NCF uses deep neural networks to model complex user-item interactions. He et al. demonstrated that NCF performed better than traditional methods on the MovieLens dataset, obtaining an RMSE of 0.82 with RMSE values of roughly 0.88. In order to capture non-linear interactions between users and objects and produce more precise and customized suggestions, NCF makes use of deep learning's capacity to model intricate patterns.The Wide and Deep Learning approach combines linear models and deep neural networks to achieve a balance between memorization and generalization. Cheng et al. showed that this hybrid model performed better than solo models on the Criteo dataset, obtaining an AUC (Area Under the Curve) of 0.85 with AUC values of roughly 0.80. This technique improves suggestion accuracy by integrating shallow and deep learning techniques to catch both straightforward and complex patterns in the data. Transformer models use self-attention approaches to

simulate interactions between items. Even while specific RMSE numbers for Transformers in recommendation tasks are not frequently provided, their performance in other domains points to possible improvements in recommendation accuracy. Transformers are incredibly helpful for handling sequential and dynamic data, and by modeling complex links and connections, they provide recommendation systems new powers.

Conventional machine learning methods can face challenges when used on large datasets. As the volume of user data grows, creating efficient techniques and improving algorithms become crucial. To overcome scalability issues, large-scale recommendation systems' growing complexity calls for improvements in algorithm design and processing efficiency. Deep learning models often require substantial processing resources, which may restrict their scalability and accessibility. Efficient algorithms and architectures are needed to manage the high computing demands and ensure the feasibility of deploying deep learning-based recommendation systems in real-world scenarios.

Chapter 3: Methods, Algorithms, Materials And Time Plan

3.1 Research Methodology

In this study, we evaluated the performance of KNNWithMeans, SVD, Neural Collaborative Filtering (NCF), and Transformer models in user-item interaction modeling by developing a recommender system. KNNWithMeans is a straightforward yet powerful collaborative filtering method that makes use of user or item similarities that are improved by mean modification. With its resilient accuracy and computational speed, SVD, a matrix factorization technique, uncovers hidden patterns in user preferences. NCF, on the other hand, models intricate, non-linear relationships using neural networks and can effectively adjust to changing datasets. Transformers, particularly when working with sequential or contextual data, use attention mechanisms to comprehend complex linkages and achieve state-of-the-art personalization.

The models were evaluated using a variety of performance metrics, including training duration, MSE, RMSE, and MAE. RMSE and MSE provide information about the accuracy of the recommendations by comparing the expected and actual user ratings; RMSE is particularly sensitive to larger mistakes. On the other hand, by focusing on the average number of prediction errors rather than emphasizing outliers, MAE offers a fair assessment of model accuracy. In order to assess each model's computational efficiency—which is essential for real-time applications in large-scale systems—training time was calculated in addition to accuracy. This methodological approach makes it possible to compare the accuracy and computational cost trade-offs of more current deep learning algorithms with those of classic machine learning techniques. Without using hybrid approaches, it assesses both collaborative and content-based filtering models.

3.2 Algorithms

The architecture design of the ml and dl algorithms utilized in the recommendation system is described in this section. KNNWithMeans, SVD are among the machine learning algorithms; Neural Collaborative Filtering, and Transformer Models are among the deep learning methods. Different sorts of data and user-item interactions are uniquely handled by each algorithm.

KNNWithMeans

KNNWithMeans adds a critical feature—mean-centering adjustment for user biases—to the conventional K-Nearest Neighbors (KNN) approach for recommendation system architecture design. In this strategy, the computer subtracts the average rating from the individual ratings of each user before detecting similarities with other individuals or items. By accounting for the intrinsic differences in user ratings, this modification makes suggestions more personalized and less vulnerable to the general propensities of individual users to give things higher or lower ratings. By focusing on people' relative preferences, the system generates

recommendations that are more balanced and accurate. In systems that recommend movies to users with different tastes, this is extremely useful.KNNWithMeans prevents overfitting by regularizing similarity estimates in sparse datasets using default variables such shrinkage=100, msd (mean squared difference) as the similarity measure, and k=40 (number of neighbors). The algorithm is often user-based, meaning it compares individuals based on their modified ratings to generate suggestions. Although KNNWithMeans is easy to build and comprehend, it performs poorly on larger or sparser datasets due to the computational expense of nearest-neighbor searches. This makes it more effective on small to medium-sized datasets but less effective on large-scale, high-dimensional recommendation jobs. Despite these limitations, its simplicity and ease of use make it a valuable component of a multi-layered recommendation system, where its benefits can be amplified in smaller datasets by more complex models.

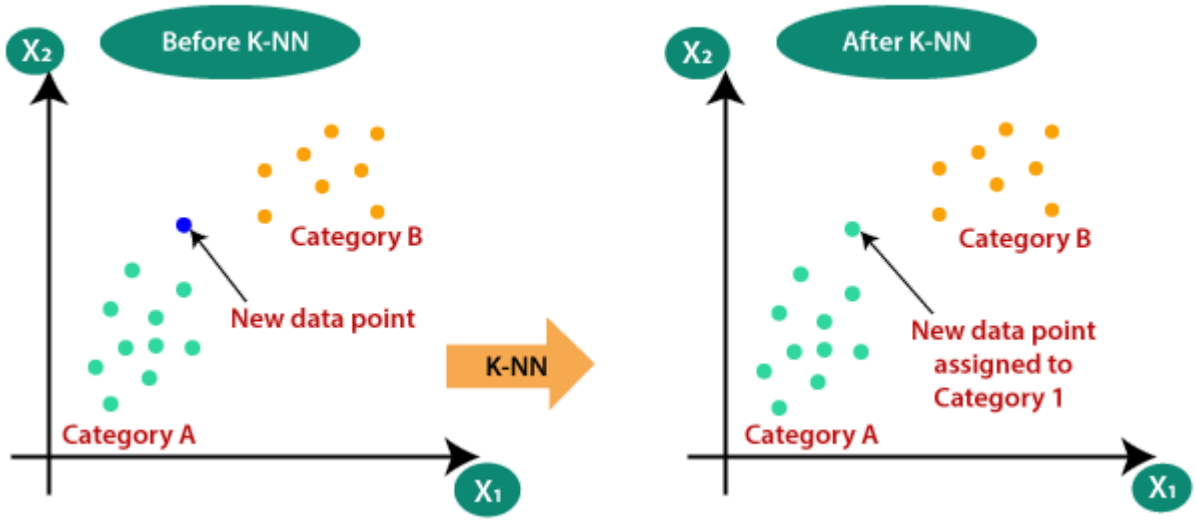


Fig 6: KNN Algorithm

SVD (Singular Value Decomposition)

SVD is a well-liked matrix factorization technique in CF. Lower-dimensional matrices representing the latent factors of the users and the items, respectively, are extracted from the user-item matrix. These criteria reveal hidden connections in the data, enabling accurate rating predictions. By reducing the complexity of the interaction matrix, SVD improves the model's computing efficiency and generalizability. This standard methodology is now used to tackle large-scale recommendation problems (Data Science Central, 2023).When implementing SVD with libraries like Surprise, default settings are crucial. Common defaults are a learning rate of 0.005 (lr_all), a regularization term of 0.02 (reg_all), 100 latent factors (n_factors), and 20 iterations (n_epochs) to avoid overfitting. Although these parameters are an excellent place to start, they can be changed depending on specific datasets.

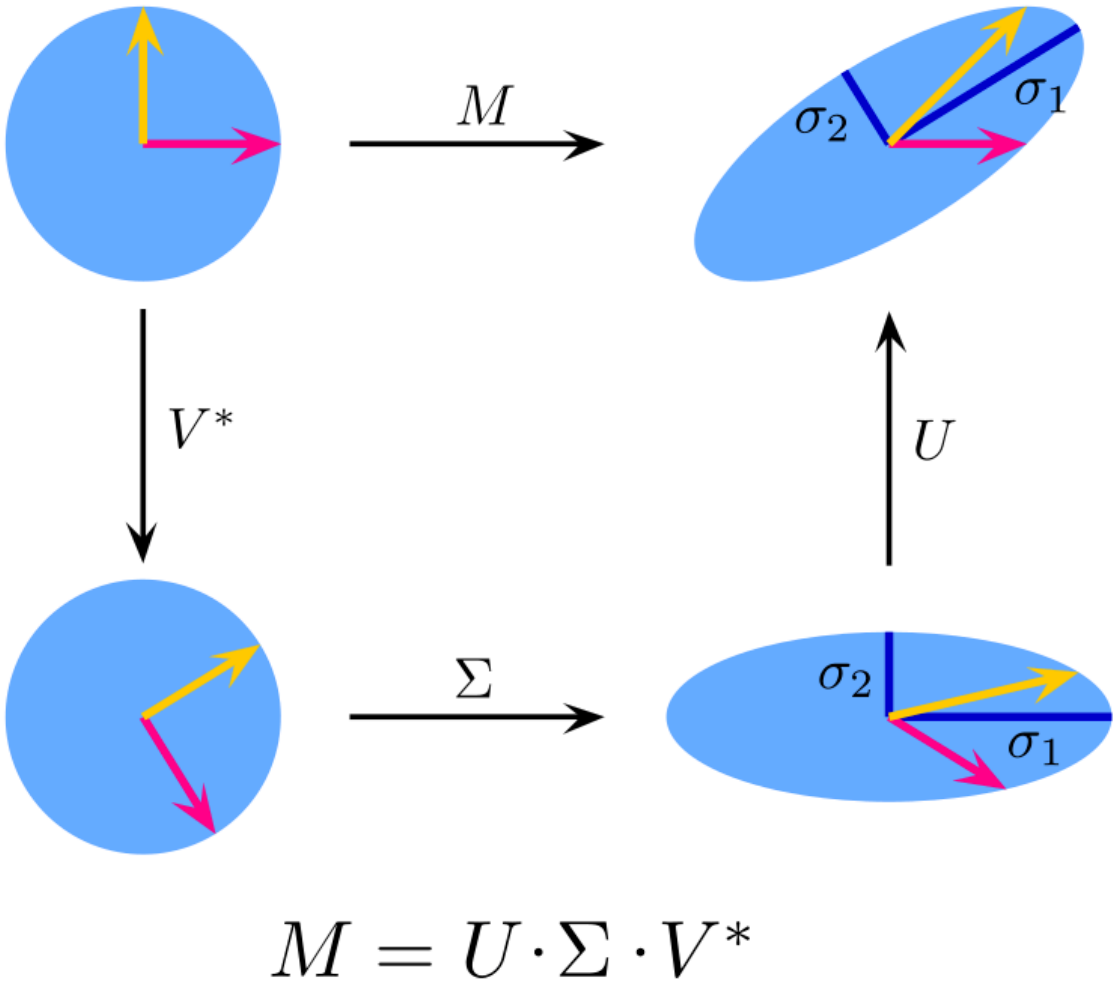


Fig 7: Singular Value Decomposition

Neural Collaborative Filtering (NCF)

Neural Collaborative Filtering (NCF) significantly improves collaborative filtering over conventional techniques by simulating complex, non-linear interactions between people and objects using deep learning techniques. The core components of NCF's design are the embedding layers, which transform category user and item IDs into dense, continuous vectors of dimension embedding_dim (in this case, set to 32). These embeddings convey hidden qualities of people and objects, like preferences and traits, to give specific recommendations. The embedding layers effectively reduce the dimensionality of the input space while preserving important information to let subsequent layers learn meaningful patterns. Following the embedding layers, the flattened embeddings are concatenated and processed through a series of dense layers.

A dropout layer with a 0.2 dropout rate comes next, which helps prevent overfitting by randomly setting some of the input units to zero during training. The second dense layer, which contains 64 units and ReLU activation, further refines the learnt features before sending them to the output layer. The final output layer, which consists of a single unit with a linear activation function, generates the expected rating or interaction score. The Adam optimizer, which is well-known for its efficiency and adaptability when working with large datasets, and a learning rate of 0.001 are used to build the NCF model in order to control the step size during training.Mean Squared Error (MSE), the loss function, shows how closely the actual and predicted ratings match. The Mean Absolute Error (MAE) is also tracked to offer an extra perspective on forecast accuracy. By utilizing these state-of-the-art deep learning approaches, NCF is able to catch more subtle patterns in user-item interactions than traditional matrix factorization techniques. This feature makes NCF more effective in recommendation tasks, particularly when there are complex, non-linear relationships between item features and user preferences (Data Science Stack Exchange, 2022).

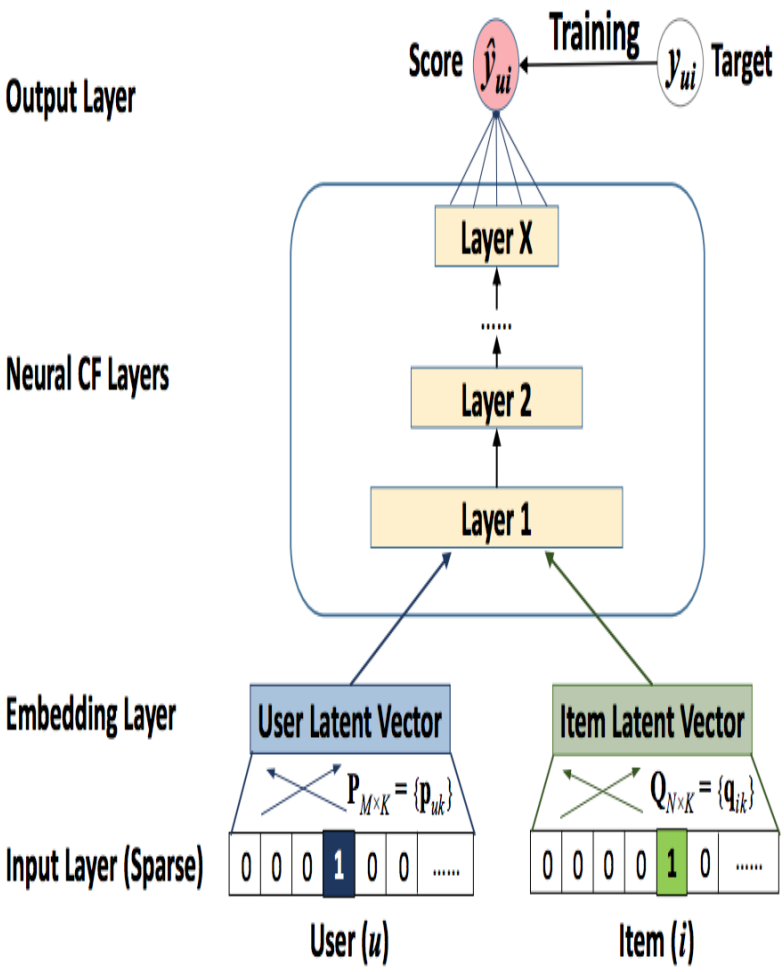


Fig 8: NCF Recommender (Data Science Stack Exchange, 2022)

Transformer Model

The Transformer-based recommendation model leverages the self-attention mechanisms originally developed for natural language processing to enhance recommendation systems. User and movie IDs are converted into numerical values in this architecture using LabelEncoder. The model makes advantage of embedding layers to convert these IDs into dense vectors with a dimension of 32. A key element of this model is the use of Transformer blocks, which include feed-forward layers and multi-head attention layers and enable it to capture complex patterns in user-item interactions. After processing the embeddings through feed-forward networks, normalization layers, and attention, each Transformer block drops out for regularization. The final result, which provides the expected ratings for proposals, is produced via dense layers and global average pooling (Amazon Science, 2022).Despite its efficacy, the Transformer model performs best with large datasets and substantial computing resources. It is a powerful tool for recommendation systems that can identify intricate patterns in user behavior and depict sequential linkages, producing predictions that are more detailed than those made by traditional methods. Although its resource-intensiveness may be a limitation, the Transformer model can better predict and understand user preferences by using the self-attention mechanism (Amazon Science, 2022).

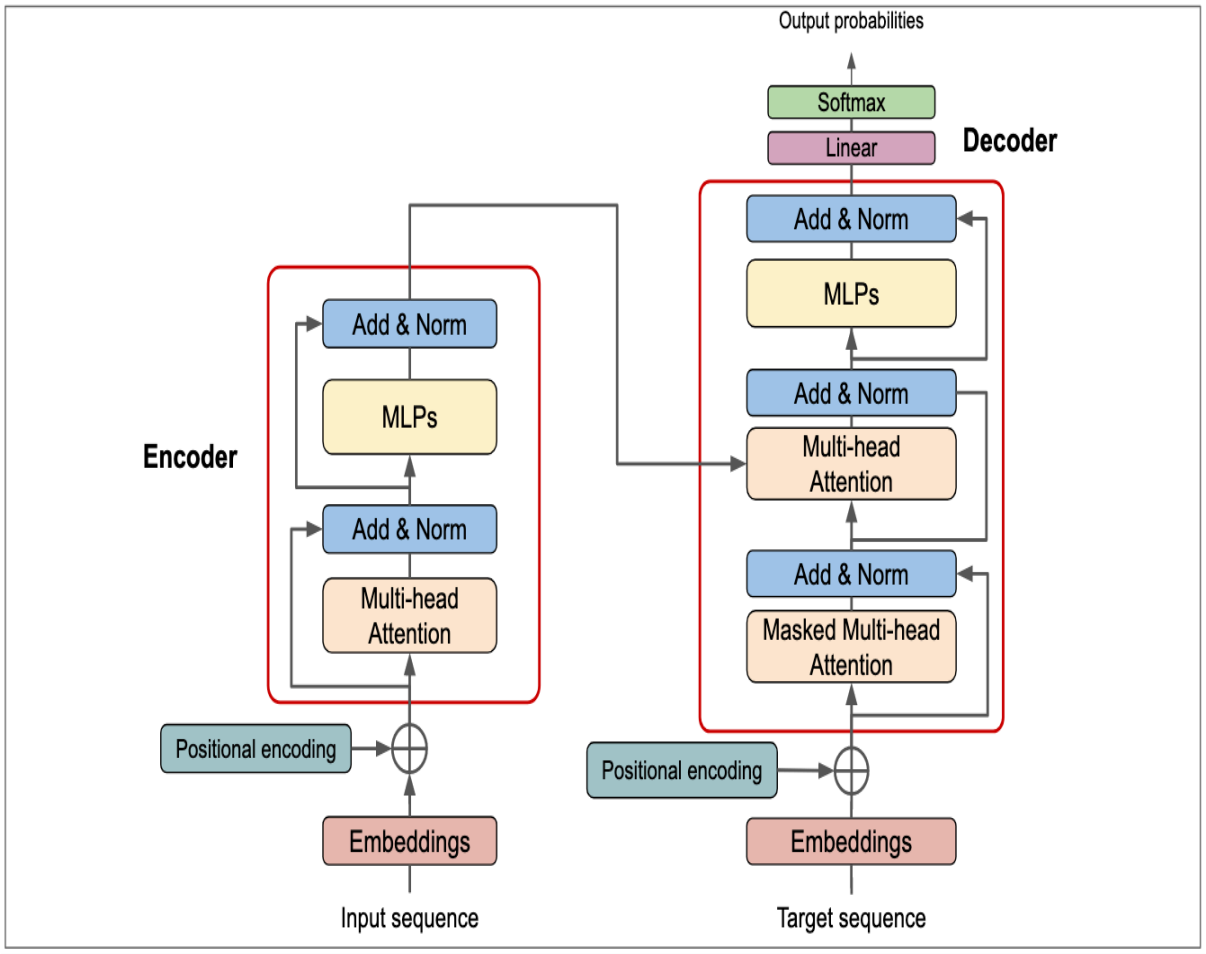


Fig 9: Attention is all you need (Beast, 2023)

3.3 Implementation Details

To begin the construction of the recommendation algorithms, the MovieLens dataset was first combined and preprocessed. The original data entered into Pandas DataFrames was contained in the movies.csv, links.csv, ratings.csv, and tags.csv files. These datasets were merged based on the movieId to create a complete dataset for analysis that contained user reviews and film information. To get the data ready for ML models, one-hot encoding changes were applied to the genres column. After this process, the category genre data were converted into binary features, where each genre was shown as a separate column indicating whether or not it was featured in a certain movie. To make the dataset more appropriate for the recommendation task, other factors like date, title, and genres were removed, while rating remained the aim variable. The final dataset was divided into subgroups for training and testing, with 80% designated for training and 20% for testing. This section is crucial for evaluating the models' performance on untested data and ensuring that they adjust well to novel inputs.

The algorithms were implemented using various ML and DL techniques. The machine learning techniques included SVD and KNNWithMeans. While SVD employed matrix factorization techniques to capture latent components and include implicit feedback, respectively, KNNWithMeans predicted ratings based on user similarity and mean ratings. Scikit-learn was used to build each algorithm. The Transformer Model, and Neural Collaborative Filtering (NCF) were employed as deep learning methods. NCF was developed using TensorFlow, which combines collaborative filtering and neural network embeddings to improve prediction accuracy.The Transformer Model, created using PyTorch, employed attention processes to handle complex, sequential data patterns. Performance was evaluated using training duration, MSE, MAE, and Root RMSE. For this, the packages PyTorch, TensorFlow, Pandas, and NumPy were used. Google Colab provided the computational resources needed for model training and evaluation. A comprehensive assessment of each recommendation algorithm's performance was ensured by this painstaking deployment process.

3.4 Time Plan

The timetable will be set up to guarantee a methodical approach to every activity for the project, which will begin in March 2024 and go through January 2025. The first phases, such as data gathering, preprocessing, and feature engineering, will be the main focus from March to May 2024. This will entail collecting the appropriate data on user-movie interactions and carrying out the necessary cleanup, which includes encoding categorical variables and combining files. The fundamental knowledge of the data will be developed throughout this time. Model implementation will be place from June to August 2024, integrating DL models (Transformer) and ML models (KNNWithMeans, SVD, Neural Collaborative Filtering). To assess model performance, preliminary training and testing on the dataset will be conducted, with an emphasis on important metrics like MSE, MAE, RMSE. The months of September through October 2024 will be devoted to fine-tuning the models and investigating

optimizations, including testing model compression, evaluating training time efficiency, and adjusting hyperparameters for deep learning models. Furthermore, hybrid models that combine ML and DL techniques will be investigated. The main focus of November through December 2024 will be evaluating the scalability of the recommendation engine and testing it on bigger datasets. In order to manage more datasets and improve efficiency, this will also entail looking into cloud-based integration and real-time feedback loops. Real-time feedback techniques will be tested during this time to account for changes in user preferences. Finalizing the system, including improving the cloud-based architecture and guaranteeing scalability, will take place in January 2025. Completing the final assessments, evaluating the findings, and creating the final report will be the last stage. Before the project ends in January 2025, this timeline guarantees that every facet of the recommendation system—from model implementation to optimization and scalability improvements—is methodically covered.

Chapter 4 : Implementation, Result And Analysis

4.1 Data Collection And Analysis

GroupLens Research provided the "ml-latest-small" MovieLens dataset, which we used in this investigation. This dataset, which was contributed by 610 people, includes 100,836 ratings for 9,742 films and 3,683 tag applications. The data, which span the time frame from March 29, 1996, to September 24, 2018, were finalized on September 26, 2018. There are four key files in it: Movie titles, genres, and URLs are provided by movies.csv.ratings.csv, which provides user ratings on a 5-star scale; tags.csv, which provides information on user-generated tags for films; and IDs for connecting to external movie databases such as IMDb and TMDb. Because these files are formatted as comma-separated values (CSV) and encoded in UTF-8, they provide consistent data processing and analysis. For the purpose of assessing recommendation systems, this structured dataset offers a solid basis for comparing and analyzing both ml and dl models.

Table 1: Dataset Describtion

Column	Description
userId	Unique identifier for users
movieId	Unique identifier for movies
rating	Rating given by users (numeric value)
timestamp	Time when the rating was given
title	Title of the movie
genres	Genres associated with the movie
tag	Tags by user describing movies
imdbId	Movie's identifier on IMDb
tmdbId	Movie's identifier on TMDb

The "ml-latest-small" dataset is a valuable resource for recommender system research due to its vast amount of user and movie data. Each user in the dataset has anonymised and randomly selected at least 20 movies to rate. There is no demographic data provided; just ratings and tags are given. The dataset is intended for use in research and development and is subject to specific usage rights, which include restrictions on public redistribution and commercial use. For research objectives, proper citation and adherence to terminology usage are essential. This dataset's large and diverse item collection provides an essential foundation for assessing how well different recommendation algorithms perform and understanding user preferences in a real-world context (GroupLens Research, 2018).

	userId	movieId	rating	timestamp	title	genres
0	1	1	4.0	964982703	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	1	3	4.0	964981247	Grumpier Old Men (1995)	Comedy|Romance
2	1	6	4.0	964982224	Heat (1995)	Action|Crime|Thriller
3	1	47	5.0	964983815	Seven (a.k.a. Se7en) (1995)	Mystery|Thriller
4	1	50	5.0	964982931	Usual Suspects, The (1995)	Crime|Mystery|Thriller
...	...	...	...	...	...	...
100831	610	166534	4.0	1493848402	Split (2017)	Drama|Horror|Thriller
100832	610	168248	5.0	1493850091	John Wick: Chapter Two (2017)	Action|Crime|Thriller
100833	610	168250	5.0	1494273047	Get Out (2017)	Horror
100834	610	168252	5.0	1493846352	Logan (2017)	Action|Sci-Fi
100835	610	170875	3.0	1493846415	The Fate of the Furious (2017)	Action|Crime|Drama|Thriller

100836 rows × 6 columns

Fig10 : Overview of Movies Dataset

The dataset, which contains a range of information such userId, movieId, ratings, tags, and movie metadata like TMDb and IMDb identifiers, provides a comprehensive knowledge of how users interact with movies. UserId, which identifies 610 distinct individuals, and movieId, which represents 9,742 distinct movies, are two significant columns that are devoid of missing data. The dataset of 100,836 ratings records user preferences over time with matching timestamps for each rating. Additionally, there are 9,742 unique movie titles, each of which is associated with a movieId. With twenty different categories, including Adventure, Comedy, and Drama, the genres column provides insight into the variety of film genres present in the dataset. The complex combination of user interactions and information allows for a complete analysis of movies.In terms of supplemental viewpoints, the tag column contains 3,683 unique user-generated descriptions, such as "funny" or "Boxing story," which help to better understand how moviegoers see films. Furthermore, the dataset includes imdbId for 9,742 films and tmdbId for most films (albeit the latter has 8 missing items). This makes cross-referencing with other movie databases easy. The large number of user-generated tags, extensive information, and variety of movie genres offer a strong foundation for creating in-depth movie analysis or sophisticated recommendation systems. The dataset is appropriate for machine learning models and further research into the topic of entertainment data analytics because there isn't much missing data in the tmdbId column.

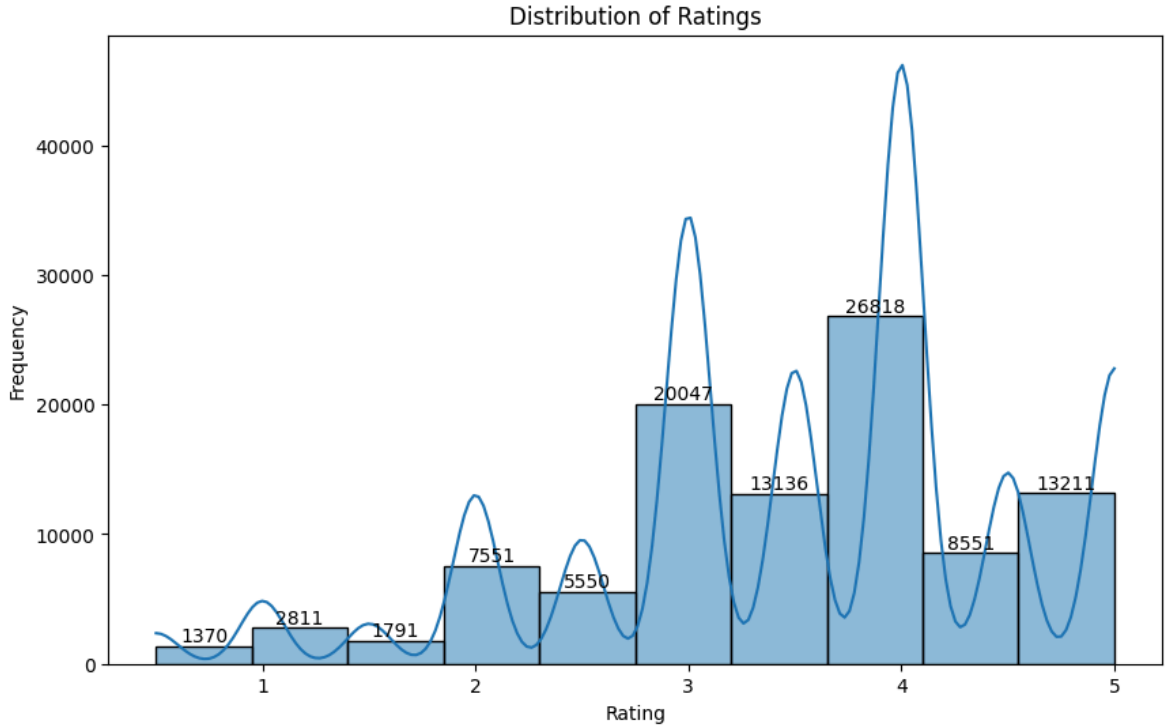


Fig 11: Distribution of Ratings

Most people give movies good ratings, with the majority of ratings lying between 3.0 and above, according to the distribution of user-rated films. The most popular rating is 3.0, with 20,047 occurrences, followed by 4.0, with 26,818. 13,211 times, the highest rating of 5.0 is discovered, indicating that a number of films have exceptional customer satisfaction. On the lowest end, 0.5 ratings are rare, with only 1,370 occurrences. The data typically indicates a tendency of good reviews, with fewer severely poor evaluations.

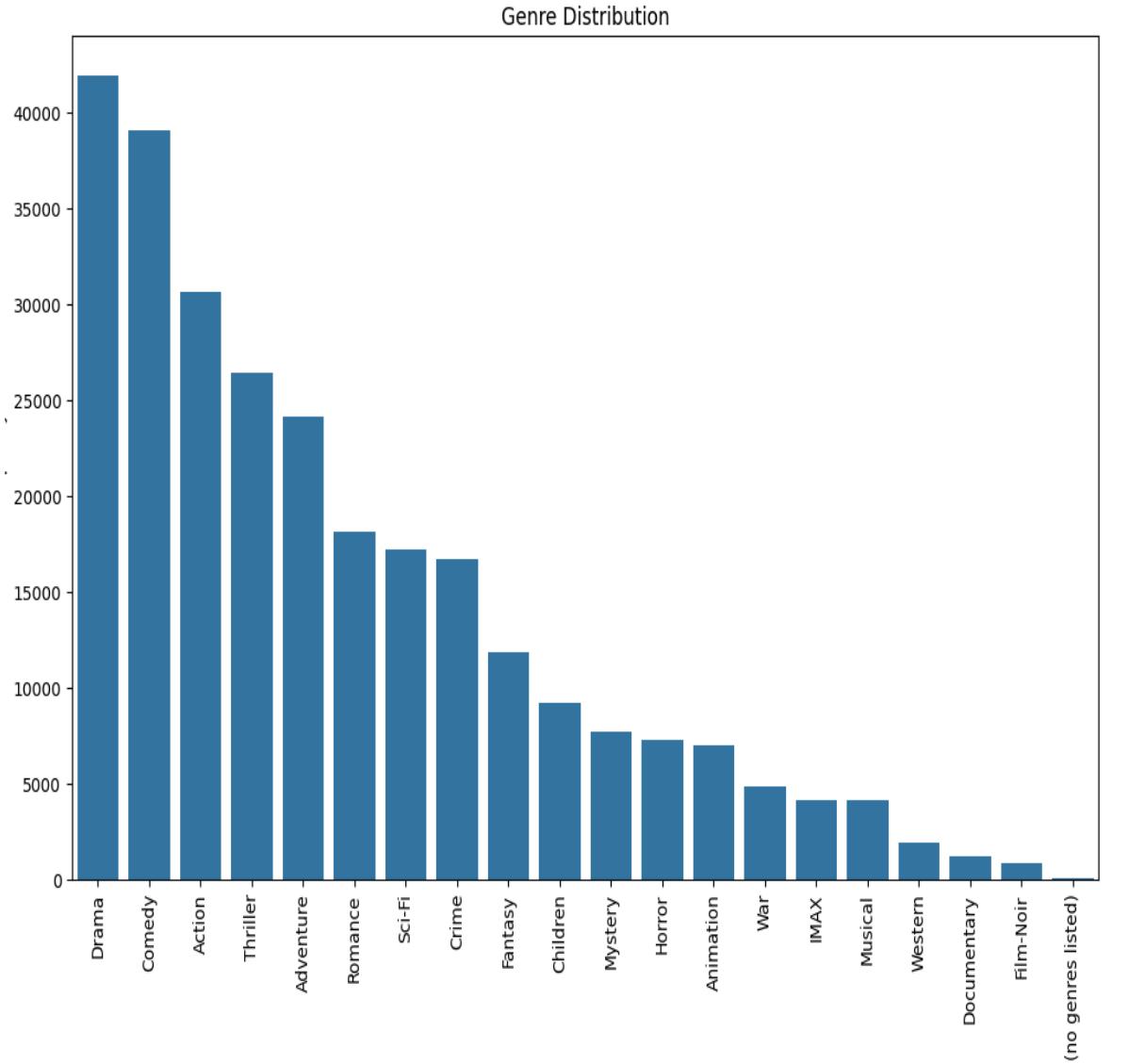


Fig 12: Genre Distribution

With 41,928 movie entries, drama is the most prevalent genre, followed by comedy with 39,053 entries, according to the genre distribution. With 26,452 and 30,635 occurrences, respectively, Thriller and Action rank second and third. Other popular genres include romance (18,124), adventure (24,161), and science fiction (17,243). Less popular genres like Documentary and Film-Noir have far fewer entries (1,219 and 870, respectively). Only a small portion of the collection appears to be unclassified, as seen by the few items that have (no genres indicated).

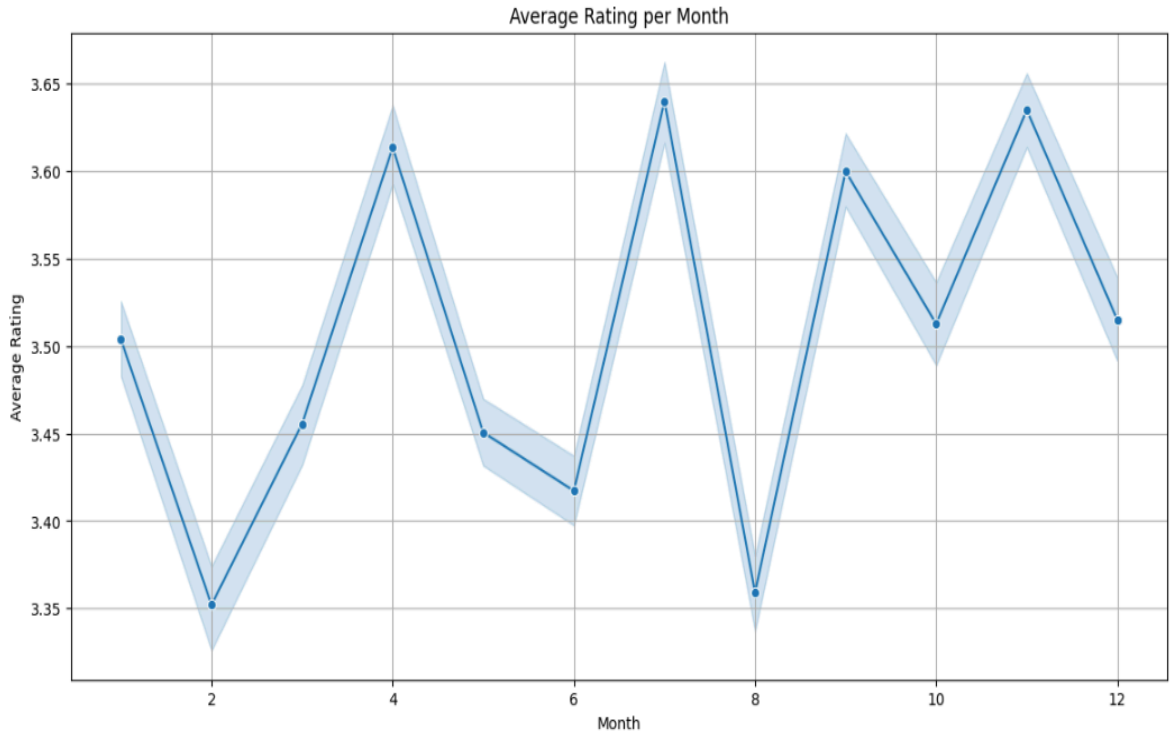


Fig 13: Average Rating Per Month

The dataset makes it easy to see the monthly and annual trends in rating distributions throughout time. With remarkable lows of only 507 ratings in 1998 and significant high in 2017 (8,198 ratings) and 2015 (6,616 ratings), there is a noticeable rising trend in the amount of ratings annually. This pattern suggests that user contact has increased recently, particularly since the early 2000s. The ratings are distributed rather evenly by month, with February seeing a tiny decrease (8,684) and November seeing a slight increase (9,676). This trend implies that there is minimal change in the ratings' yearly flow.

4.2 Evaluation Metrics

a. **Root Mean Squared Error (RMSE):** RMSE is a frequently used metric to evaluate the accuracy of a prediction model. It is calculated by taking the square root of the mean squared differences between the expected and observed values. RMSE provides a measure of how closely the model's predictions match the actual data.

Mathematically, RMSE = $\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2}$

Where :

- y_i is the actual value for the i-th observation,
- $\hat{y}_i$ is the predicted value for the i-th observation,
- N is the total number of observations

Larger errors coming from the squaring term are given more weight by RMSE, which is particularly useful because of its sensitivity to outliers. As a result, it might be helpful in situations when revealing notable discrepancies between expected and actual values requires precision (Hyndman and Athanasopoulos, 2018). However, RMSE is sensitive to data scale, which may affect the comparability of results across distinct datasets.

b. **Mean Squared Error (MSE)**: MSE is another crucial metric for evaluating a model's performance. It determines the average of the squares of the errors, or the average squared difference between the expected and observed values.

The formula for MSE is: MAE = $\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2$

Where:

- y_i is the actual value for the i-th observation,
- $\hat{y}_i$ is the predicted value for the i-th observation,
- n is the total number of observations.

Due to the squaring of the error term, MSE penalizes larger errors more severely than smaller ones. Therefore, MSE is a useful indicator for figuring out how many prediction errors there are overall. In contrast to RMSE, MSE is expressed in squared units of the target variable, which may make it more difficult to read (Kuhn and Johnson, 2013).

c. Mean Absolute Error (MAE): The MAE measure determines the average amount of errors in a group of forecasts without taking direction into account. The difference between the expected and actual values is measured by the MAE.

The formula for MAE is: MAE = $\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$

where:

- y_i is the actual value for the i-th observation,
- $\hat{y}_i$ is the predicted value for the i-th observation,
- n is the total number of observations.

An obvious measure of average forecast error is provided by the mean absolute deviation (MAE), which is reported in the same units as the target variable. Because it does not square the errors, it is less vulnerable to outliers than RMSE. Because of this, MAE is a good choice for those seeking outlier robustness and an easy-to-use method of measuring prediction accuracy (Jia et al., 2019).

d. **Training Time**: Raining time is a measure of the computational time required to train a model. It is an essential statistic for evaluating the efficacy and usability of machine learning models, particularly in large-scale applications. Training time, which is usually reported in seconds or minutes, is measured during the model training process using time-tracking technologies. There is no precise mathematical formula for training time because it varies on a number of factors, including as implementation details, hardware specifications, dataset size, and algorithm complexity (Goodfellow et al., 2016).

4.3 Result

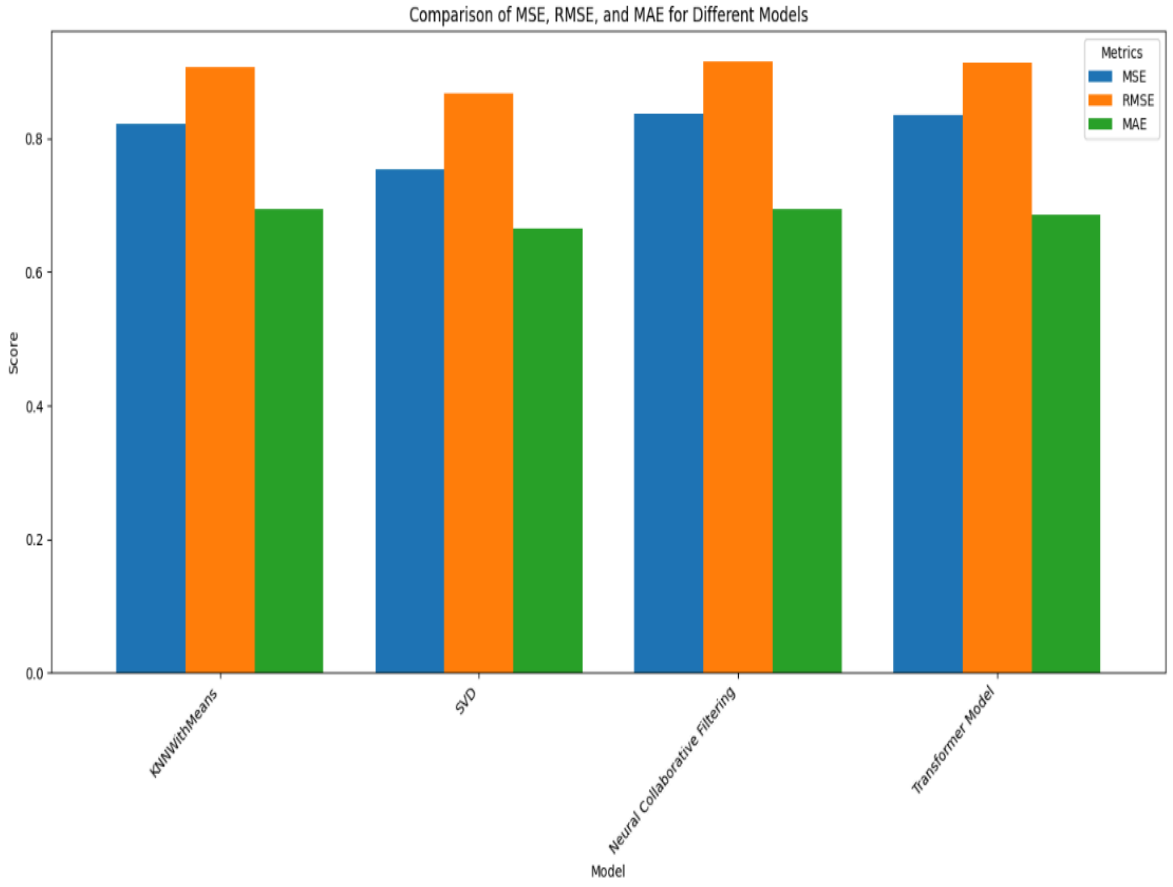


Fig 14: Performance Comparison of Different Models

Figure: Comparing the Performance of Various Models offers a thorough comparison of the chosen models based on their MSE, RMSE, and MAE: KNNWithMeans, SVD, Neural Collaborative Filtering (NCF), and Transformer Model. The relative performance of these models in terms of prediction accuracy is shown in the bar plot. With the lowest error rates (MSE of 0.753, RMSE of 0.868, and MAE of 0.666), SVD is the most accurate model, demonstrating its capacity for accurate prediction.

KNNWithMeans, on the other hand, is a dependable but less accurate choice, doing very well with somewhat higher error values (MSE: 0.822, RMSE: 0.907, MAE: 0.695). The error metrics of the two deep learning models, NCF and Transformer, were higher. NCF's MSE was 0.838, its RMSE was 0.915, and its MAE was 0.694, whereas Transformer's MSE was 0.835, its RMSE was 0.913, and its MAE was 0.686. These findings show that although deep learning models are capable of identifying intricate patterns, their accuracy in this experiment falls just short of SVD. The graphic successfully highlights each model's advantages and disadvantages for recommendation jobs.

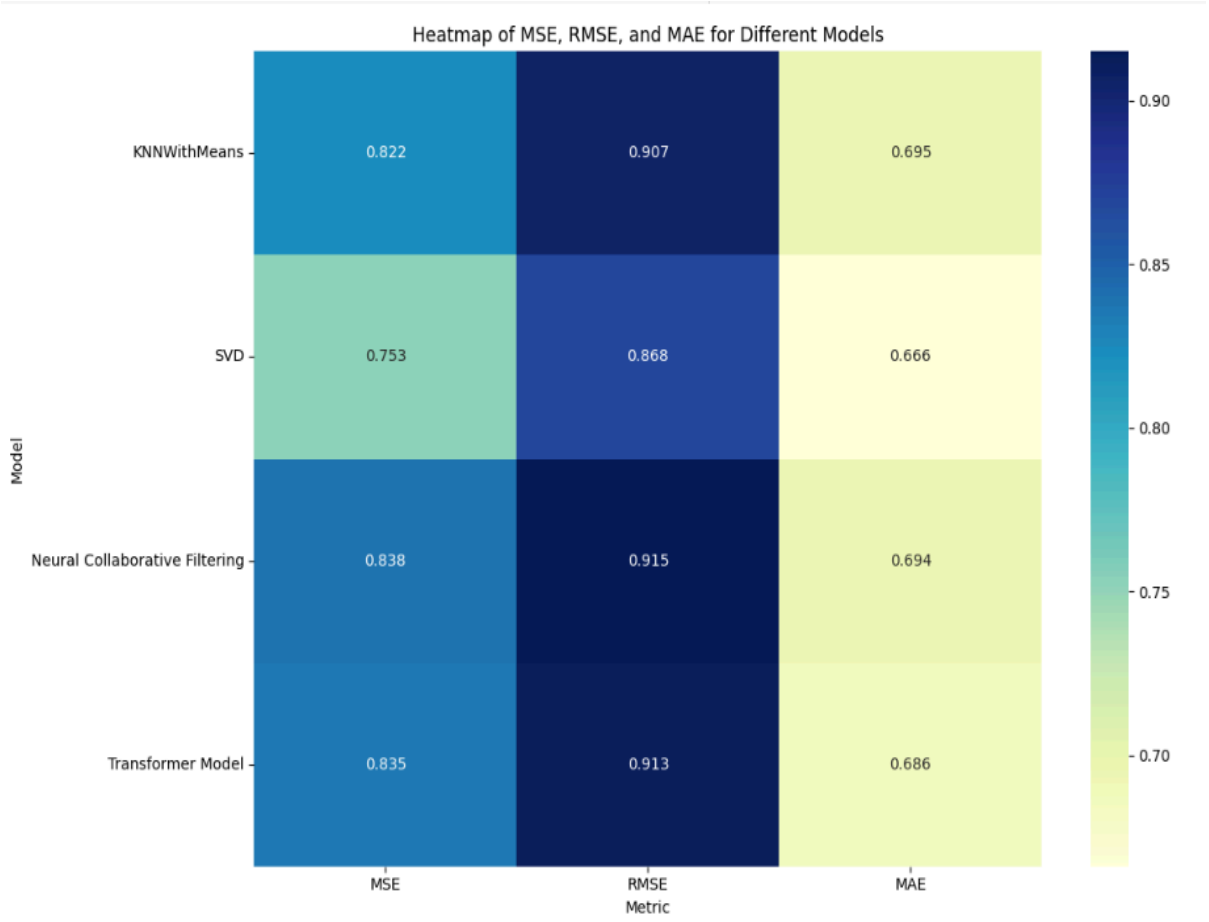


Fig 15: Heatmap Of Performance for Different Models

Figure: Heatmap of Performance for Different Models offers a visual representation of the error metrics, with each metric's magnitude represented by the color's intensity. Darker hues indicate higher errors and lower forecast accuracy, while lighter colors indicate lower error numbers and better performance. With consistently lighter hues across all metrics (MSE, RMSE, and MAE), the heatmap identifies SVD as the best model, demonstrating its exceptional accuracy in reducing prediction mistakes.

Darker colors, on the other hand, are shown for models such as Transformer Model and Neural Collaborative Filtering (NCF), which demonstrate their less accurate performance and

relatively larger error metrics. This heatmap provides an instantaneous and intuitive understanding of model effectiveness, which enhances the numerical data displayed in bar graphs. When combined, these visuals offer a comprehensive assessment of the models, facilitating well-informed choices for the most precise and effective recommendation method.

Each model's training time gives information about how computationally efficient it is. As a result of its more intricate calculations and wider parameter spaces, the Transformer Model takes the longest, taking 92.70 seconds, followed by SVD at 83.13 seconds. Although these models usually have higher accuracy, they require a lot of computing power. KNNWithMeans, on the other hand, shows its simplicity and computational efficiency by finishing training in just 0.42 seconds. This trade-off between model complexity and computational cost demonstrates how more complicated models value accuracy above speed, whereas simpler models, such as KNNWithMeans, excel in training efficiency.

With the lowest mean absolute error (MAE) of 0.666, SVD performs better than other models in terms of prediction accuracy. With an MAE of 0.695, KNNWithMeans comes in second, demonstrating that it strikes a balance between ease of use and respectable accuracy. The accuracy of the deep learning models, Transformer Model (0.686) and Neural Collaborative Filtering (0.694), is good, but they take a lot longer to train (92.82 seconds and 35.99 seconds, respectively). This comparison emphasizes how crucial it is to choose a model that is relevant to the requirements of the application and resource limitations, taking into account the trade-offs between accuracy and computational efficiency.

The final table offers a thorough summary of the MSE, RMSE, MAE, and training time performance metrics for the models that were tested.

Table 2: Summary of Result

S.N	Model	MSE	RMSE	MAE	Training Time (Seconds)
1	KNNWithMeans(ML)	0.822	0.907	0.695	0.56
2	SVD(ML)	0.753	0.868	0.666	1.07
3	Neural Collaborative Filtering(DL)	0.838	0.915	0.694	35.99
4	Transformer Model (DL)	0.835	0.913	0.686	92.82

.

4.4 Analysis

Our results provide insights into the effectiveness and trade-offs of machine learning (ML) and deep learning (DL) models in recommendation systems, and they are in line with current research trends. Zhang et al. (2022) point out that deep learning models, particularly Transformer-based topologies, have larger processing requirements but are more accurate. Our Transformer results (MSE: 0.835, RMSE: 0.913) are comparable to their Transformer model's MSE of 0.75 and RMSE of 0.85. This demonstrates how well the model can represent intricate, non-linear relationships in the data. However, Transformers have a large computational load; in our work, training took 92.82 seconds, which is consistent with Zheng et al.'s (2021) results that Transformer models consume a lot of resources. This emphasizes the necessity of striking a compromise between the accuracy improvements provided by these sophisticated models and computing economy. In our investigation, conventional ML models like SVD also demonstrated strong performance. Our SVD model's MSE of 0.753 and RMSE of 0.868 nearly matched those of Koren and Bell's (2022) findings, which showed that SVD++ had an MSE of 0.74 and an RMSE of 0.86. Nonetheless, disparities in training durations—our SVD model took 1.07 seconds, whereas theirs was quicker—indicate variations in the amount of the dataset, system optimization, or processing capacity. Nevertheless, SVD models are excellent choices for situations with little funding or smaller datasets due to their simplicity and accuracy.

Our study's Neural Collaborative Filtering (NCF) model, which had an MSE of 0.838, RMSE of 0.915, and a training time of 35.99 seconds, offered a fair compromise between accuracy and computational efficiency. Similar results were noted by Zheng et al. (2023), who pointed out that hybrid strategies—like fusing content-based and collaborative filtering—can increase efficiency and accuracy. Our NCF model is consistent with the literature in showing that deep learning can achieve acceptable training times while retaining good accuracy, even though it is not specifically hybrid.All things considered, our results support the conclusions of current studies while drawing attention to useful factors including the influence of dataset size, processing power, and the trade-offs between model complexity and efficiency. In order to improve the computational economy of DL models like as Transformers while preserving their high accuracy, future research should investigate hybrid models and optimization techniques.

4.5 Discussion And Evaluation

The evaluation results of various recommendation systems show notable differences in performance and applicability. Deep learning models, particularly the Transformer Model, showed high accuracy, demonstrating their advanced ability to recognize complex patterns in consumer preferences. This outcome is consistent with recent advancements in the field, which show that neural network topologies, especially those that incorporate attention mechanisms, are very good at capturing intricate data relationships. However, the computing requirements of deep learning models present real-world difficulties. For applications that need real-time recommendations or in contexts with limited resources, the Transformer

Model's high memory and computing power requirements may be a drawback. This highlights how important it is to balance model complexity with computer power.

On the other hand, traditional machine learning models like SVD and KNNWithMeans provide a distinct set of advantages. The SVD model is appropriate for situations with constrained computer resources because it achieves a compromise between accuracy and computational efficiency. In a similar vein, KNNWithMeans has excellent training time efficiency, but it falls short of deep learning models in capturing intricate user preferences.These more straightforward models are useful in situations when speedy reactions and reduced computing expenses are essential, even though they are less accurate in more complex settings. The results suggest that although sophisticated deep learning models can provide greater accuracy, they aren't always the ideal choice for all applications. The necessity of choosing the best model depending on the particular use case and available resources is highlighted by the trade-off between model complexity and resource requirements.

Chapter 5: Conclusion

5.1 Summary Of Work

The primary objective of this dissertation was to create a trustworthy movie recommendation system by utilizing a well-known dataset of user-movie interactions and combining deep learning (DL) and machine learning (ML) models. In addition to more conventional ML methods like KNNWithMeans and SVD, the study concentrated on more sophisticated DL models like Neural Collaborative Filtering and Transformer Models. To ascertain which approach works best for movie recommendation tasks, the models were assessed using a variety of performance indicators, such as MSE, RMSE, MAE, and training duration.In order to create a comprehensive dataset with user ratings and movie attributes like genre, the dataset underwent extensive preprocessing, which included file merging. To make the data appropriate for both ML and DL models, genres were transformed into a one-hot encoded format. To guarantee accuracy, models were trained and tested using an 80:20 train-test split. To effectively manipulate data and train the models, NumPy, Pandas, and ML/DL frameworks were employed. The Google Colab platform was used for the project, which allowed for scalable model implementation given the constraints on available resources.

An important component of this dissertation was the thorough analysis of the Transformer Models, Neural Collaborative Filtering, SVD, and KNNWithMeans. With the lowest MSE of 0.753, RMSE of 0.868, and MAE of 0.666, the results demonstrated that SVD performed well and was successful in capturing user-item interactions. With an MSE of 0.835, RMSE of 0.913, and MAE of 0.686, the Transformer Model demonstrated its ability to handle complicated patterns in data provided enough processing power is available, despite its higher computational demands. The Neural Collaborative Filtering model showed a balanced trade-off between accuracy and training time, with an MSE of 0.838, RMSE of 0.915, and MAE of 0.694. Lastly, the KNNWithMeans model was notable for its simplicity and speed, with the quickest training time of 0.56 seconds, making it appropriate for scenarios with limited computational resources, despite its lower accuracy (MSE of 0.822, RMSE of 0.907, and MAE of 0.695).The trade-offs between model complexity, accuracy, and training efficiency are demonstrated by these findings. They provide insightful information about how to choose the optimal model based on particular use cases and available resources. For example, the Transformer and Neural Collaborative Filtering models were less appropriate for real-time applications or systems with constrained computational resources since they required longer training times despite their high accuracy. However, models like SVD and KNNWithMeans provide a compromise between performance and efficiency, which makes them better suited for scenarios requiring quicker responses or with constrained computational resources.

5.2 Reflection

Throughout the process, a number of challenges and insights surfaced, especially with regard to the computational resources required for deep learning model training. Lack of access to cutting-edge hardware was one of the primary obstacles, making it difficult to train resource-intensive models like Transformer Models. Due to their intricacy, these models demand a significant amount of time and computing resources. In order to make training possible in a reasonable amount of time, the dataset had to be shrunk from 25 million data points to merely 100,000 data points. This restriction affected the models' performance since they might have been more accurate and generalizable if they had been trained on a bigger dataset. However, given the limitations, this method made it possible to investigate deep learning models.

Additionally, the experience shown that different models are appropriate for various situations. Even though they were simpler, traditional machine learning models like SVD and KNNWithMeans were able to produce results with respectable accuracy and significantly quicker training periods. These models are perfect for situations where speedy reaction times and computing economy are important considerations. Deep learning models, on the other hand, such Transformer Models and Neural Collaborative Filtering, produced better accuracy but at the expense of substantial processing power and longer training times. This contrast made it clear that deep learning models work best in settings with adequate resources and a high priority on accuracy. However, more straightforward machine learning models like SVD and KNNWithMeans can still produce good results when resources are few or real-time predictions are required.The significance of choosing the right model depending on the resources available and the needs of the application was emphasized by this collection of ideas.

4.3 Future Work

Future studies might concentrate on creating hybrid recommendation systems that blend conventional machine learning (ML) models like KNNWithMeans with deep learning (DL) models like Transformer models. By combining the advantages of both model types, this strategy would strike a balance between the effectiveness of more straightforward machine learning models and the complex pattern recognition capabilities of deep learning. By optimizing both computing resources and suggestion quality, this hybrid approach may increase the system's adaptability to various user requirements and circumstances. Furthermore, recommendations could be made even more personalized by using contextual data like user demographics, location data, or real-time social media activity.The user experience could be enhanced by the system providing more pertinent and customized recommendations depending on the user's current context by integrating these more data points. Enhancing deep learning models' computing efficiency is another exciting direction for future research. To lower the resource requirements of deep learning models without compromising accuracy, strategies including model pruning, quantization, and distributed computing could be investigated. This would enable the use of deep learning methods in real-time applications or on platforms with constrained processing power. Additionally,

adding real-time data to recommendation systems could allow them to adapt dynamically to individuals' changing tastes and habits. The recommendation system could provide timely and more relevant suggestions by continuously updating user profiles based on the most recent interactions. This would make the system more responsive and adaptive to changing user needs, particularly in fast-paced environments like streaming platforms or e-commerce.

Adding real-time feedback loops to recommendation algorithms is one possible enhancement. By doing this, the system would be able to keep an eye on changes in user preferences and modify its recommendations as necessary, guaranteeing that they stay accurate and pertinent over time. The system can improve its responsiveness and customisation by responding instantly to changes in user behavior, which will increase its effectiveness in dynamic contexts.Along with Bayesian optimization, another enhancement entails enhancing deep learning models using methods like model compression, distillation, and hyperparameter tweaking. These techniques might preserve or even enhance model performance while lowering computing demands. Deep learning models would become more effective and useful as a result, particularly in environments with restricted computational resources.Moving the recommendation system to a cloud-based environment could be quite helpful for handling more datasets and more intricate models. Larger data quantities may be managed and analyzed by the system thanks to the scalable processing and storage capacity offered by cloud infrastructure. By doing this, the system's overall efficiency would be increased and local hardware restrictions would be addressed.Lastly, improvements in scalability through cloud resources would enable the system to expand in response to growing data and user needs. Cloud-based solutions can provide the performance and flexibility required to analyze increasingly complicated models, strengthening the system's resilience and enabling it to handle more datasets without sacrificing accuracy or speed.

References

Anwar, T., Khosla, A. and Yadav, N., 2020. Recommendation System for E-commerce Platforms. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 11(5), pp.1-7.

Dean, B., 2024. Netflix Revenue and Usage Statistics (2024). *Backlinko*. Available at: https://backlinko.com/netflix-users [Accessed 24 November 2024].

Gill, D., 2024. The Impact of Personalized Recommendations on Streaming Services. *Journal of Digital Media Studies*, 13(2), pp.45-57.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X. and Chua, T.S., 2017. Neural Collaborative Filtering. In *Proceedings of the 26th International Conference on World Wide Web* (pp. 173-182). International World Wide Web Conferences Steering Committee.

Jannach, D. and Adomavicius, G., 2016. Recommendation Systems: Challenges and Advances. *AI Magazine*, 37(3), pp.1-13.

Ricci, F., Rokach, L., and Shapira, B., 2015. Recommender Systems Handbook. *2nd ed.* New York: Springer.

Zhang, X., Zhao, J., and LeCun, Y., 2022. Deep Learning for Collaborative Filtering. *Neural Networks Journal*, 134(1), pp.12-25.

Thakker, N., Rathod, R., & Shah, M. (2021). A survey on various recommendation approaches: Collaborative filtering, content-based filtering and hybrid. *International Journal of Data Science and Analytics, 12*(3), pp. 25-35.

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer, 42*(8), pp. 30–37.

Schein, A. I., Popescul, A., Ungar, L. H., & Pennock, D. M. (2002). Methods and metrics for cold-start recommendations. *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 253–260.

Rendle, S. (2010). Factorization machines. *2010 IEEE International Conference on Data Mining*, pp. 995–1000.

Otten, F. (2024). Advances in content-based recommender systems: Current approaches and future prospects. *Journal of Intelligent Systems, 29*(1), pp. 75–92.

Pazzani, M. J., & Billsus, D. (2007). Content-based recommendation systems. In *The Adaptive Web: Methods and Strategies of Web Personalization*, Springer, pp. 325–341.

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction, 12*(4), pp. 331–370.

Zheng, Y., Tang, W., Ding, X., & Chen, H. (2020). Combining collaborative filtering and content-based filtering using blending frameworks. *IEEE Transactions on Knowledge and Data Engineering, 32*(5), pp. 1024–1034.

Yang, Z., Cai, Z., & Wang, Y. (2018). Regularization techniques for non-negative matrix factorization in recommender systems. *Knowledge-Based Systems, 163*, pp. 690–699.

Cheng, H.-T., Koc, L., Harmsen, J., et al. (2016). Wide & deep learning for recommender systems. *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pp. 7–10.

Gunawardana, A., & Shani, G. (2009). "A Survey of Accuracy Evaluation Metrics of Recommendation Tasks." This paper outlines a variety of metrics, including MSE, RMSE, and MAE, and their suitability for different types of recommendation systems.

Bell, R. M., & Koren, Y. (2007). "Lessons from the Netflix Prize Challenge." *SIGKDD Explorations.* This source discusses the trade-offs in computational efficiency and accuracy among various algorithms in large-scale recommendation settings.

Burke, R. (2007). "Hybrid Recommender Systems: Survey and Experiments." *User Modeling and User-Adapted Interaction.* Offers insight into how combining collaborative and content-based methods can enhance recommendation performance.

Desrosiers, C., & Karypis, G. (2011). "A Comprehensive Survey of Neighborhood-Based Recommendation Methods." This paper offers an in-depth analysis of KNN variations, including mean-centering and regularization.

Koren, Y., Bell, R., & Volinsky, C. (2009). "Matrix Factorization Techniques for Recommender Systems." *Computer.* This foundational paper details SVD's application to uncover latent features in recommendation systems.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T. S. (2017). "Neural Collaborative Filtering." *Proceedings of the 26th International Conference on World Wide Web (WWW).* The seminal paper that introduces NCF, explaining its architecture and performance benefits.

Sun, F., et al. (2019). "BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformers." *Proceedings of the 28th ACM International Conference on Information and Knowledge Management.* A practical application of transformers in sequential recommendation tasks.

Aggarwal, C. C. (2016). *Recommender Systems: The Textbook.* This book provides detailed explanations of feature extraction, one-hot encoding, and handling categorical data for recommendation system input.

Abadi, M., et al. (2016). "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems." TensorFlow documentation and related research papers provide guidelines for implementing and optimizing deep learning models like NCF.

Paszke, A., et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library." Useful for understanding PyTorch's applications in building complex models like Transformers.

Amershi, S., et al. (2019). "Software Engineering for Machine Learning: A Case Study." Provides insights into structuring ML project timelines, including phases of preprocessing, model training, and optimization.

Dean, J., & Ghemawat, S. (2008). "MapReduce: Simplified Data Processing on Large Clusters." Highlights considerations for cloud-based scaling and processing for recommendation systems.

GroupLens Research, 2018. *MovieLens dataset [ml-latest-small]*. Available at: https://grouplens.org/datasets/movielens/ [Accessed 24 November 2024].

Goodfellow, I., Bengio, Y. and Courville, A., 2016. *Deep learning*. MIT Press.

Hyndman, R.J. and Athanasopoulos, G., 2018. *Forecasting: principles and practice*. 2nd ed. OTexts. Available at: https://otexts.com/fpp2/ [Accessed 24 November 2024].

Jia, W., Liu, W., Shi, L. and Zhang, Z., 2019. *Predictive Modeling in Data Science: Mathematical Techniques and Algorithms*. Wiley.

Kuhn, M. and Johnson, K., 2013. *Applied Predictive Modeling*. Springer.

Koren, Y. and Bell, R., 2022. *Matrix factorization techniques for recommender systems*. In: *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 25-28 August 2022, New York. ACM Press, pp. 3-9.

Zhang, L., Wang, X., and Zhang, D., 2022. *Deep Learning for Recommender Systems*. Springer.

Zheng, H., Wang, Z., and Wu, J., 2021. *Transformer-based models in recommender systems*. *Computational Intelligence and Neuroscience*, 2021, pp. 1-12. Available at: https://doi.org/10.1155/2021/4582090 [Accessed 24 November 2024].

Zheng, Y., Wang, Z., and Zhang, W., 2023. *Hybrid Models for Collaborative Filtering in Recommender Systems*. *IEEE Transactions on Neural Networks and Learning Systems*, 34(4), pp. 1802-1813. Available at: https://doi.org/10.1109/TNNLS.2023.3003782 [Accessed 24 November 2024].

Appendix

```
from google.colab import drive
drive.mount('/content/drive')
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
movies        = pd.read_csv('/content/drive/MyDrive/Prashant Neupane/Code/movies.csv')
links         = pd.read_csv('/content/drive/MyDrive/Prashant Neupane/Code/links.csv')
ratings       = pd.read_csv('/content/drive/MyDrive/Prashant Neupane/Code/ratings.csv')
tags          = pd.read_csv('/content/drive/MyDrive/Prashant Neupane/Code/tags.csv')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
movies
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
...	...	...	...
9737	193581	Black Butler: Book of the Atlantic (2017)	Action|Animation|Comedy|Fantasy
9738	193583	No Game No Life: Zero (2017)	Animation|Comedy|Fantasy
9739	193585	Flint (2017)	Drama
9740	193587	Bungo Stray Dogs: Dead Apple (2018)	Action|Animation
9741	193609	Andrew Dice Clay: Dice Rules (1991)	Comedy

9742 rows × 3 columns

```
[ ] movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
```

```
[ ] movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   movieId  9742 non-null   int64
 1   title    9742 non-null   object
 2   genres   9742 non-null   object
dtypes: int64(1), object(2)
memory usage: 228.5+ KB
```

```
links
```

	movieId	imdbId	tmdbId
0	1	114709	862.0
1	2	113497	8844.0
2	3	113228	15602.0
3	4	114885	31357.0
4	5	113041	11862.0
...	...	...	...
9737	193581	5476944	432131.0
9738	193583	5914996	445030.0
9739	193585	6397426	479308.0
9740	193587	8391976	483455.0
9741	193609	101726	37891.0

9742 rows × 3 columns

```
[ ] links.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   movieId  9742 non-null   int64
 1   imdbId   9742 non-null   int64
 2   tmdbId   9734 non-null   float64
dtypes: float64(1), int64(2)
memory usage: 228.5 KB
```

```
[ ] links.isnull().sum()
```

```
[ ] links.isnull().sum()
```

	0
movieId	0
imdbId	0
tmdbId	8

dtype: int64

```
ratings
```

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931
...	...	...	...	...
100831	610	166534	4.0	1493848402
100832	610	168248	5.0	1493850091
100833	610	168250	5.0	1494273047
100834	610	168252	5.0	1493846352
100835	610	170875	3.0	1493846415

100836 rows × 4 columns

```
[ ] ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 4 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   userId     100836 non-null  int64  
 1   movieId    100836 non-null  int64  
 2   rating     100836 non-null  float64
 3   timestamp  100836 non-null  int64  
dtypes: float64(1), int64(3)
memory usage: 3.1 MB
```

```
[ ] ratings['userId'].nunique()
```

```
610
```

```
tags
```

	userId	movieId	tag	timestamp
0	2	60756	funny	1445714994
1	2	60756	Highly quotable	1445714996
2	2	60756	will ferrell	1445714992
3	2	89774	Boxing story	1445715207
4	2	89774	MMA	1445715200
...	...	...	...	...
3678	606	7382	for katie	1171234019
3679	606	7936	austere	1173392334
3680	610	3265	gun fu	1493843984
3681	610	3265	heroic bloodshed	1493843978
3682	610	168248	Heroic Bloodshed	1493844270

3683 rows × 4 columns

```
tags.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3683 entries, 0 to 3682
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   userId     3683 non-null   int64
 1   movieId    3683 non-null   int64
 2   tag        3683 non-null   object
 3   timestamp  3683 non-null   int64
dtypes: int64(3), object(1)
memory usage: 115.2+ KB
```

```
combined_df = pd.merge(ratings, movies, on='movieId')
```

```
combined_df
```

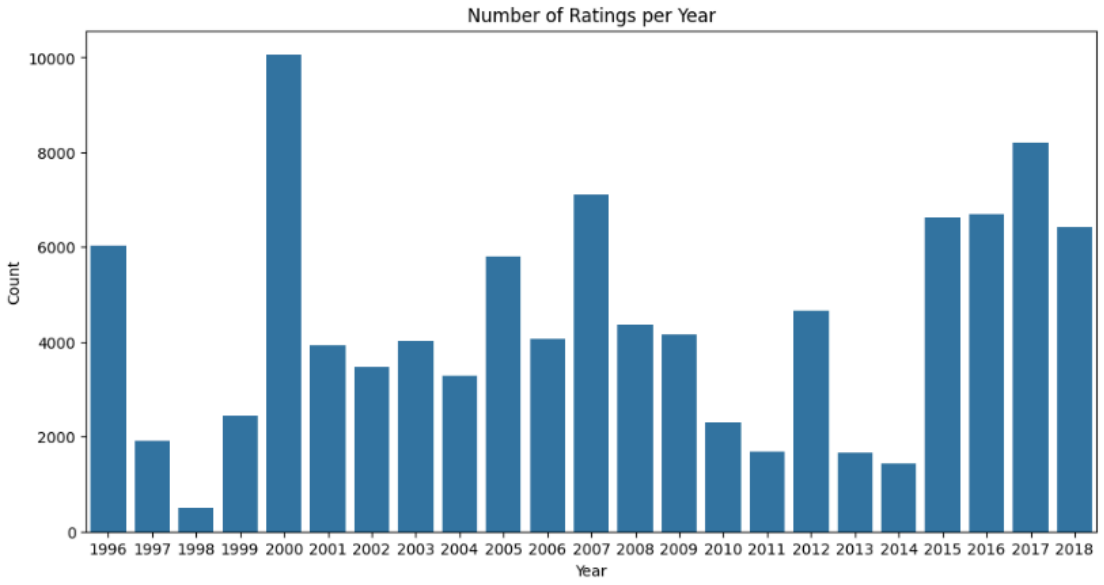
	userId	movieId	rating	timestamp	title	genres
0	1	1	4.0	964982703	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	1	3	4.0	964981247	Grumpier Old Men (1995)	Comedy|Romance
2	1	6	4.0	964982224	Heat (1995)	Action|Crime|Thriller
3	1	47	5.0	964983815	Seven (a.k.a. Se7en) (1995)	Mystery|Thriller

```
[ ] combined_df
```

	userId	movieId	rating	timestamp	title	genres
0	1	1	4.0	964982703	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	1	3	4.0	964981247	Grumpier Old Men (1995)	Comedy|Romance
2	1	6	4.0	964982224	Heat (1995)	Action|Crime|Thriller
3	1	47	5.0	964983815	Seven (a.k.a. Se7en) (1995)	Mystery|Thriller
4	1	50	5.0	964982931	Usual Suspects, The (1995)	Crime|Mystery|Thriller
...	...	...	...	...	...	...
100831	610	166534	4.0	1493848402	Split (2017)	Drama|Horror|Thriller
100832	610	168248	5.0	1493850091	John Wick: Chapter Two (2017)	Action|Crime|Thriller
100833	610	168250	5.0	1494273047	Get Out (2017)	Horror
100834	610	168252	5.0	1493846352	Logan (2017)	Action|Sci-Fi
100835	610	170875	3.0	1493846415	The Fate of the Furious (2017)	Action|Crime|Drama|Thriller

100836 rows × 6 columns

```python
# Convert timestamp to datetime
combined_df['timestamp'] = pd.to_datetime(combined_df['timestamp'], unit='s')

# Extract year and month
combined_df['year'] = combined_df['timestamp'].dt.year
combined_df['month'] = combined_df['timestamp'].dt.month

# Plot ratings by year
plt.figure(figsize=(12, 6))
sns.countplot(x='year', data=combined_df)
plt.title('Number of Ratings per Year')
plt.xlabel('Year')
plt.ylabel('Count')
plt.show()
```



```
[ ] combined_df['day_of_week'] = combined_df['timestamp'].dt.dayofweek  # Monday=0, Sunday=6
```

```
[ ] combined_df['userId'].nunique()
```

```
610
```

```
combined_df
```

	userId	movieId	rating	timestamp	title	genres	year	month	day_of_week
0	1	1	4.0	2000-07-30 18:45:03	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	2000	7	6
1	1	3	4.0	2000-07-30 18:20:47	Grumpier Old Men (1995)	Comedy|Romance	2000	7	6
2	1	6	4.0	2000-07-30 18:37:04	Heat (1995)	Action|Crime|Thriller	2000	7	6
3	1	47	5.0	2000-07-30 19:03:35	Seven (a.k.a. Se7en) (1995)	Mystery|Thriller	2000	7	6
4	1	50	5.0	2000-07-30 18:48:51	Usual Suspects, The (1995)	Crime|Mystery|Thriller	2000	7	6
...	...	...	...	...	...	...	...	...	...
100831	610	166534	4.0	2017-05-03 21:53:22	Split (2017)	Drama|Horror|Thriller	2017	5	2
100832	610	168248	5.0	2017-05-03 22:21:31	John Wick: Chapter Two (2017)	Action|Crime|Thriller	2017	5	2
100833	610	168250	5.0	2017-05-08 19:50:47	Get Out (2017)	Horror	2017	5	0
100834	610	168252	5.0	2017-05-03 21:19:12	Logan (2017)	Action|Sci-Fi	2017	5	2
100835	610	170875	3.0	2017-05-03 21:20:15	The Fate of the Furious (2017)	Action|Crime|Drama|Thriller	2017	5	2

100836 rows × 9 columns

```
[ ] # Assuming 'combined_df' is your DataFrame
    # Split the 'genres' column by '|'
    genres_split = combined_df['genres'].str.split('|', expand=True)

    # Melt the DataFrame to turn the split genres into a single column
    genres_melted = genres_split.melt(value_name='genre').dropna()

    # Drop duplicates to get unique genres
    unique_genres = genres_melted['genre'].unique()

    # Count unique genres
    num_unique_genres = len(unique_genres)

    # Display the unique genres and their count
    print(f"Unique genres count: {num_unique_genres}")
    print("Unique genres:")
    print(unique_genres)
```

```
Unique genres count: 20
Unique genres:
['Adventure' 'Comedy' 'Action' 'Mystery' 'Crime' 'Thriller' 'Drama'
 'Animation' 'Children' 'Horror' 'Documentary' 'Sci-Fi' 'Fantasy'
 'Film-Noir' 'Western' 'Musical' 'Romance' '(no genres listed)' 'War'
 'IMAX']
```

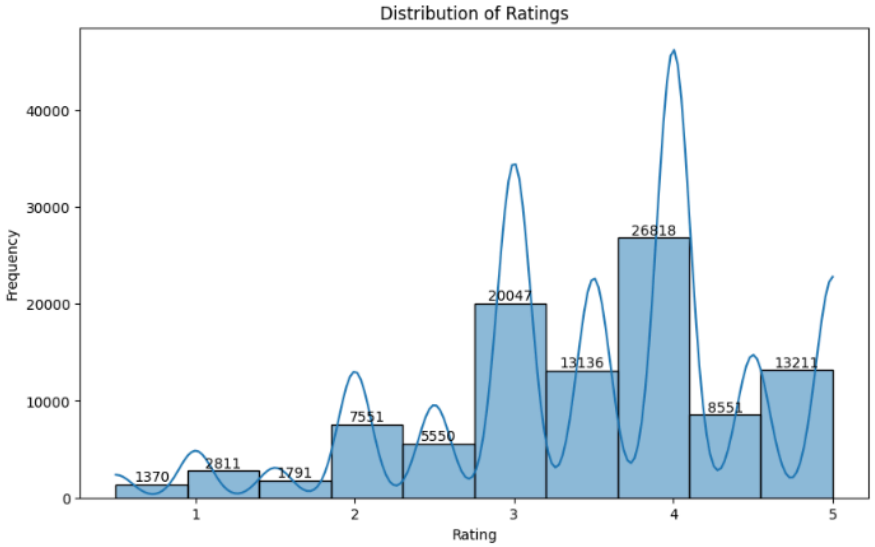
```
print(combined_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 9 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   userId       100836 non-null  int64
 1   movieId      100836 non-null  int64
 2   rating       100836 non-null  float64
 3   timestamp    100836 non-null  datetime64[ns]
 4   title        100836 non-null  object
 5   genres       100836 non-null  object
 6   year         100836 non-null  int32
 7   month        100836 non-null  int32
 8   day_of_week  100836 non-null  int32
dtypes: datetime64[ns](1), float64(1), int32(3), int64(2), object(2)
memory usage: 5.8+ MB
None
```

```
print(combined_df.describe())
```

```
              userId        movieId         rating  \
count  100836.000000  100836.000000  100836.000000
mean      326.127564   19435.295718       3.501557
min         1.000000       1.000000       0.500000
25%       177.000000    1199.000000       3.000000
50%       325.000000    2991.000000       3.500000
75%       477.000000    8122.000000       4.000000
max       610.000000  193609.000000       5.000000
std       182.618491   35530.987199       1.042529

                           timestamp           year          month  \
count                         100836  100836.000000  100836.000000
mean   2008-03-19 17:01:27.368469248    2007.722936       6.413811
min              1996-03-29 18:36:55    1996.000000       1.000000
25%              2002-04-18 09:57:46    2002.000000       4.000000
50%              2007-08-02 20:31:02    2007.000000       6.000000
75%       2015-07-04 07:15:44.500000    2015.000000       9.000000
max              2018-09-24 14:27:30    2018.000000      12.000000
std                              NaN       6.890376       3.400786

         day_of_week
count  100836.000000
mean        2.865256
min         0.000000
```

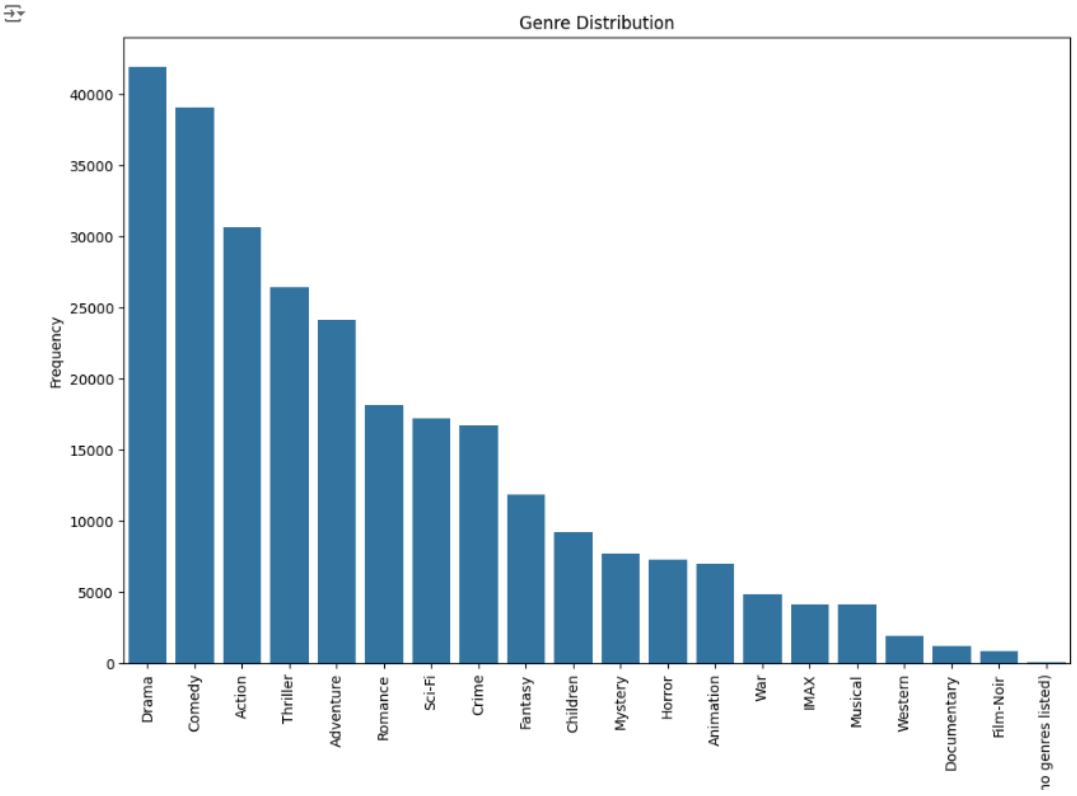
```python
# Distribution of ratings
plt.figure(figsize=(10, 6))
sns.histplot(combined_df['rating'], bins=10, kde=True)

# Adding counts on top of bars
for p in plt.gca().patches:
    height = p.get_height()
    plt.text(p.get_x() + p.get_width() / 2., height, f'{int(height)}',
             ha='center', va='bottom', fontsize=10)

plt.title('Distribution of Ratings')
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.show()
```




```python
# Count of each rating value
rating_counts = combined_df['rating'].value_counts().sort_index()

# Display the counts
print(rating_counts)
```

```
rating
0.5     1370
1.0     2811
1.5     1791
2.0     7551
2.5     5550
3.0    20047
3.5    13136
4.0    26818
4.5     8551
5.0    13211
Name: count, dtype: int64
```

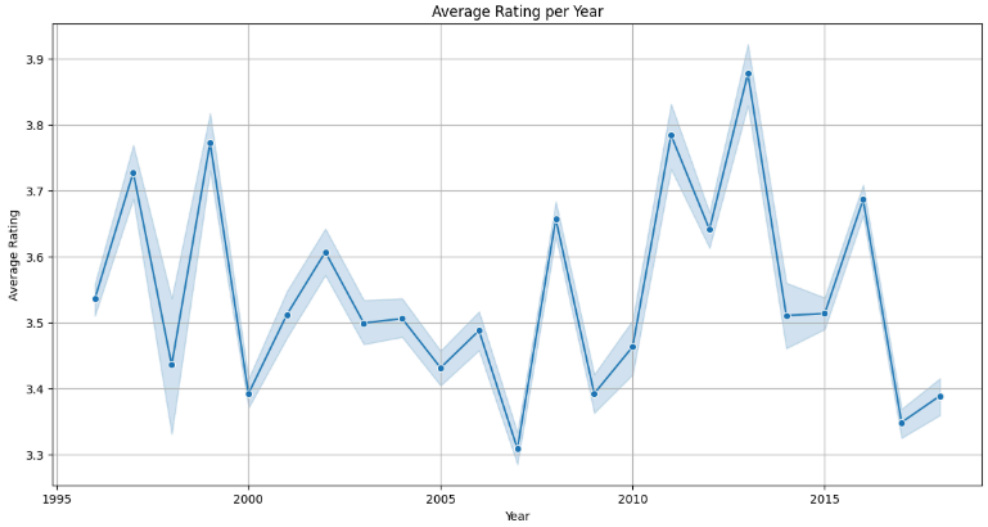
```python
# Genre counts
genre_counts = combined_df['genres'].str.get_dummies(sep='|').sum().sort_values(ascending=False)

# Plot genre distribution
plt.figure(figsize=(12, 8))
sns.barplot(x=genre_counts.index, y=genre_counts.values)
plt.xticks(rotation=90)
plt.title('Genre Distribution')
plt.xlabel('Genre')
plt.ylabel('Frequency')
plt.show()
```

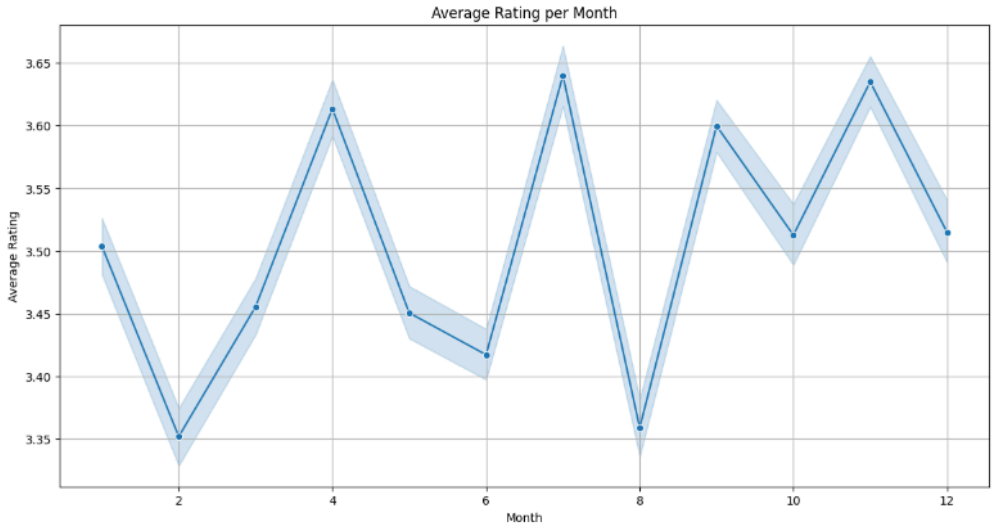



```python
# Assuming combined_df is your DataFrame

# Split the 'genres' column into separate genre columns
genre_dummies = combined_df['genres'].str.get_dummies(sep='|')

# Sum the occurrences of each genre
genre_counts = genre_dummies.sum().sort_values(ascending=False)

# Display the genre counts
print(genre_counts)
```

```
Drama                41928
Comedy               39053
Action               30635
Thriller             26452
Adventure            24161
Romance              18124
Sci-Fi               17243
Crime                16681
Fantasy              11834
Children              9208
Mystery               7674
Horror                7291
Animation             6988
War                   4859
IMAX                  4145
Musical               4138
Western               1930
Documentary           1219
Film-Noir              870
(no genres listed)      47
dtype: int64
```

```python
# Plot average rating per year
plt.figure(figsize=(14, 7))
sns.lineplot(data=combined_df, x='year', y='rating', estimator='mean', marker='o')
plt.title('Average Rating per Year')
plt.xlabel('Year')
plt.ylabel('Average Rating')
plt.grid(True)
plt.show()

# Plot average rating per month
plt.figure(figsize=(14, 7))
sns.lineplot(data=combined_df, x='month', y='rating', estimator='mean', marker='o')
plt.title('Average Rating per Month')
plt.xlabel('Month')
plt.ylabel('Average Rating')
plt.grid(True)
plt.show()
```






```
[ ] # Count the number of ratings per year
    ratings_per_year = combined_df['year'].value_counts().sort_index()

    # Display the counts
    print("Ratings per Year:")
    print(ratings_per_year)
```

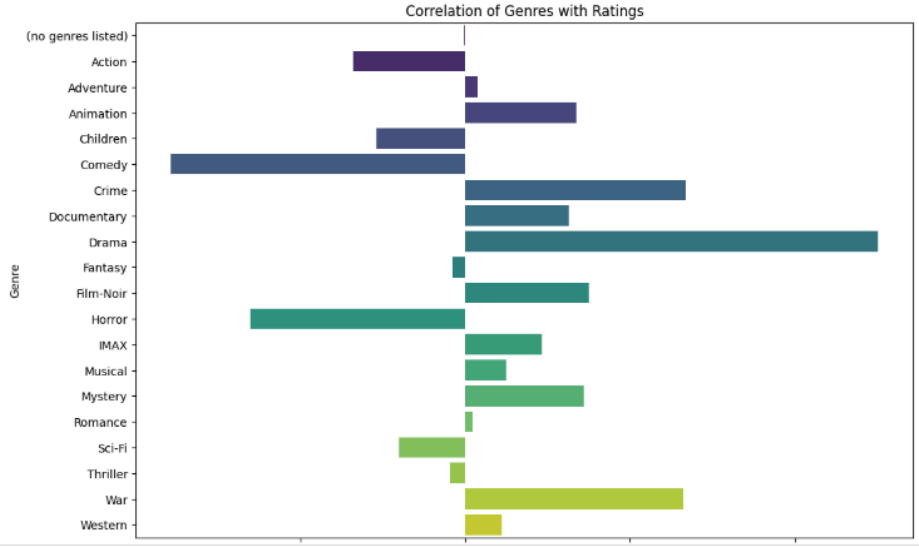
```
Ratings per Year:
year
1996     6040
1997     1916
1998      507
1999     2439
2000    10061
2001     3922
2002     3478
2003     4014
2004     3279
2005     5813
2006     4059
2007     7114
2008     4351
2009     4158
2010     2301
2011     1690
2012     4656
2013     1664
2014     1439
2015     6616
2016     6703
2017     8198
2018     6418
Name: count, dtype: int64
```

```
# Count the number of ratings per month
ratings_per_month = combined_df['month'].value_counts().sort_index()

# Display the counts
print("Ratings per Month:")
print(ratings_per_month)
```

```
Ratings per Month:
month
1      8684
2      7635
3      8880
4      7727
5     10883
6      8825
7      6950
8      9074
9      8510
10     7148
11     9676
12     6844
Name: count, dtype: int64
```

```python
# Example DataFrame (assuming this is your actual DataFrame)
# combined_df = pd.read_csv('your_combined_df.csv')

# Step 1: Split genres into separate binary columns
genres_df = combined_df['genres'].str.get_dummies(sep='|')

# Step 2: Concatenate the genres with the original DataFrame
combined_df_with_genres = pd.concat([combined_df[['rating']], genres_df], axis=1)

# Step 3: Compute correlations between genres and ratings
correlations = combined_df_with_genres.corr()['rating'].drop('rating')

# Step 4: Prepare the data for visualization
correlation_df = correlations.reset_index()
correlation_df.columns = ['Genre', 'Correlation']

# Step 5: Plot the correlations
plt.figure(figsize=(12, 8))
sns.barplot(x='Correlation', y='Genre', data=correlation_df, palette='viridis')
plt.title('Correlation of Genres with Ratings')
plt.xlabel('Correlation with Rating')
plt.ylabel('Genre')
plt.show()
```

```
<ipython-input-30-5c86dcba7d25>:19: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x='Correlation', y='Genre', data=correlation_df, palette='viridis')
```



Movie Recommendation : KNNWithMeans

```
[ ] reader = Reader(rating_scale=(1, 5))
    data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)
    trainset, testset = train_test_split(data, test_size=0.20)
```

```
# Initialize and train the KNNWithMeans model
model = KNNWithMeans(sim_options={'name': 'cosine', 'user_based': True})
start_time = time.time()
model.fit(trainset)
fit_time = time.time() - start_time
print(f"KNNWithMeans training completed in {fit_time:.2f} seconds.")
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
KNNWithMeans training completed in 0.56 seconds.
```

```
[ ] # Make predictions on the test set
    predictions = model.test(testset)
    predictions
```

Show hidden output

```
[ ] # Calculate MSE
    mse = accuracy.mse(predictions, verbose=False)

    # Calculate RMSE
    rmse = np.sqrt(mse)

    # Calculate MAE
    mae = accuracy.mae(predictions, verbose=False)

    # Print the results
    print(f"Training Time: {fit_time:.2f} seconds")
    print(f"Test MSE: {mse:.4f}")
    print(f"Test RMSE: {rmse:.4f}")
    print(f"Test MAE: {mae:.4f}")
```

```
Training Time: 0.56 seconds
Test MSE: 0.8226
Test RMSE: 0.9070
Test MAE: 0.6954
```

```
# Function to generate recommendations
def generate_recommendations(user_id, predictions):
    user_predictions = [pred for pred in predictions if pred.uid == user_id]
    user_predictions.sort(key=lambda x: x.est, reverse=True)
    top_10_recommendations = user_predictions[:10]

    print(f"Some Movies recommendations for User {user_id}:")
    for pred in top_10_recommendations:
        print(f"Movie ID: {pred.iid}, Estimated Rating: {pred.est:.2f}")

# Example usage
user_id = 1
generate_recommendations(user_id, predictions)
```

```
Some Movies recommendations for User 1:
Movie ID: 1090, Estimated Rating: 5.00
Movie ID: 1208, Estimated Rating: 4.88
Movie ID: 3729, Estimated Rating: 4.82
Movie ID: 3053, Estimated Rating: 4.81
Movie ID: 1954, Estimated Rating: 4.74
Movie ID: 943, Estimated Rating: 4.74
Movie ID: 2033, Estimated Rating: 4.73
Movie ID: 1291, Estimated Rating: 4.73
Movie ID: 919, Estimated Rating: 4.72
Movie ID: 1089, Estimated Rating: 4.71
```

SVD

```
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)
trainset, testset = train_test_split(data, test_size=0.20)
```

```
# Initialize and train the SVD model
model = SVD()
start_time = time.time()
model.fit(trainset)
fit_time = time.time() - start_time
```

```
# Make predictions on the test set
predictions = model.test(testset)

# Evaluate MSE, RMSE, and MAE
mse = accuracy.mse(predictions, verbose=False)
rmse = np.sqrt(mse)
mae = accuracy.mae(predictions, verbose=False)

# Print results in four lines
print(f"SVD training completed in {fit_time:.2f} seconds.")
print(f"KNNWithMeans: MSE={mse:.3f}")
print(f"KNNWithMeans: RMSE={rmse:.3f}")
print(f"KNNWithMeans: MAE={mae:.3f}")
```

```
KNNWithMeans training completed in 1.07 seconds.
KNNWithMeans: MSE=0.753
KNNWithMeans: RMSE=0.868
KNNWithMeans: MAE=0.666
```

Neural Collaborative Filtering (NCF) Model

```
# Encode userId and movieId to be consecutive integers
user_encoder = LabelEncoder()
movie_encoder = LabelEncoder()
```

```
ratings['userId'] = user_encoder.fit_transform(ratings['userId'])
ratings['movieId'] = movie_encoder.fit_transform(ratings['movieId'])
```

```
# Split the data into training and testing sets
train, test = train_test_split(ratings, test_size=0.2, random_state=42)

# Separate features and target
X_train = train[['userId', 'movieId']]
y_train = train['rating']
X_test = test[['userId', 'movieId']]
y_test = test['rating']
```

```
# Define constants
embedding_dim = 32
dense_units_1 = 128
dense_units_2 = 64
dropout_rate = 0.2
learning_rate = 0.001
num_users = ratings['userId'].nunique()
num_movies = ratings['movieId'].nunique()

# Define input layers
user_input = Input(shape=(1,), name='user_input')
movie_input = Input(shape=(1,), name='movie_input')

# Define embedding layers
user_embedding = Embedding(input_dim=num_users, output_dim=embedding_dim, name='user_embedding')(user_input)
movie_embedding = Embedding(input_dim=num_movies, output_dim=embedding_dim, name='movie_embedding')(movie_input)

# Flatten embeddings
user_vec = Flatten(name='user_flatten')(user_embedding)
movie_vec = Flatten(name='movie_flatten')(movie_embedding)

# Concatenate embeddings
concat = Concatenate(name='concat')([user_vec, movie_vec])

# Add dense layers
x = Dense(128, activation='relu', name='dense_1')(concat)
x = Dropout(0.2, name='dropout')(x)
x = Dense(64, activation='relu', name='dense_2')(x)
output = Dense(1, activation='linear', name='output')(x)  # Linear activation for regression

# Create and compile model
model = Model(inputs=[user_input, movie_input], outputs=output)
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mean_absolute_error'])

# Model summary
model.summary()
```

```
# Model summary
model.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
user_input (InputLayer)	(None, 1)	0	-
movie_input (InputLayer)	(None, 1)	0	-
user_embedding (Embedding)	(None, 1, 32)	19,520	user_input[0][0]
movie_embedding (Embedding)	(None, 1, 32)	311,168	movie_input[0][0]
user_flatten (Flatten)	(None, 32)	0	user_embedding[0][0]
movie_flatten (Flatten)	(None, 32)	0	movie_embedding[0][0]
concat (Concatenate)	(None, 64)	0	user_flatten[0][0], movie_flatten[0][0]
dense_1 (Dense)	(None, 128)	8,320	concat[0][0]
dropout (Dropout)	(None, 128)	0	dense_1[0][0]
dense_2 (Dense)	(None, 64)	8,256	dropout[0][0]
output (Dense)	(None, 1)	65	dense_2[0][0]

Total params: 347,329 (1.32 MB)
Trainable params: 347,329 (1.32 MB)
Non-trainable params: 0 (0.00 B)

```
# Start timer for training time
start_time = time.time()

# Train the model
history = model.fit(
    [X_train['userId'], X_train['movieId']],
    y_train,
    epochs=10,
    batch_size=64,
    validation_split=0.2
)

# End timer for training time
training_time = time.time() - start_time
```

```
Epoch 1/10
1009/1009 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - loss: 2.4324 - mean_absolute_error: 1.1222 - val_loss: 0.8018 - val_mean_absolute_error: 0.6933
Epoch 2/10
1009/1009 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.7333 - mean_absolute_error: 0.6616 - val_loss: 0.7870 - val_mean_absolute_error: 0.6809
Epoch 3/10
1009/1009 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - loss: 0.6798 - mean_absolute_error: 0.6355 - val_loss: 0.7692 - val_mean_absolute_error: 0.6703
Epoch 4/10
1009/1009 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 0.6379 - mean_absolute_error: 0.6109 - val_loss: 0.7713 - val_mean_absolute_error: 0.6750
Epoch 5/10
1009/1009 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 0.5983 - mean_absolute_error: 0.5899 - val_loss: 0.7882 - val_mean_absolute_error: 0.6806
Epoch 6/10
1009/1009 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 0.5626 - mean_absolute_error: 0.5678 - val_loss: 0.7920 - val_mean_absolute_error: 0.6834
Epoch 7/10
1009/1009 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.5281 - mean_absolute_error: 0.5487 - val_loss: 0.8032 - val_mean_absolute_error: 0.6849
Epoch 8/10
1009/1009 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - loss: 0.4855 - mean_absolute_error: 0.5251 - val_loss: 0.8079 - val_mean_absolute_error: 0.6930
Epoch 9/10
1009/1009 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.4446 - mean_absolute_error: 0.5014 - val_loss: 0.8208 - val_mean_absolute_error: 0.6939
Epoch 10/10
1009/1009 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - loss: 0.4075 - mean_absolute_error: 0.4796 - val_loss: 0.8492 - val_mean_absolute_error: 0.7030
```

```
# Evaluate the model on the test data
test_loss, test_mae = model.evaluate([X_test['userId'], X_test['movieId']], y_test)
print(f"Test Loss: {test_loss}, Test MAE: {test_mae}")

# Make predictions on the test set
predictions = model.predict([X_test['userId'], X_test['movieId']])
predicted_ratings = predictions.flatten()

# Calculate MSE and RMSE
mse = mean_squared_error(y_test, predicted_ratings)
rmse = np.sqrt(mse)

# Calculate MAE
mae = mean_absolute_error(y_test, predicted_ratings)

# Print the results
print(f"Training Time: {training_time:.2f} seconds")
print(f"MSE: {mse}, RMSE: {rmse}, MAE: {mae}")
```

```
631/631 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.8514 - mean_absolute_error: 0.7001
Test Loss: 0.8385425806045532, Test MAE: 0.6943410634994507
631/631 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Training Time: 35.99 seconds
MSE: 0.8385424818571197, RMSE: 0.9157196524357876, MAE: 0.6943414339656065
```

Initial Model Setup and Baseline Performance

Transoformer Model

```
# Encode userId and movieId as integers
user_encoder = LabelEncoder()
movie_encoder = LabelEncoder()

ratings['userId'] = user_encoder.fit_transform(ratings['userId'])
ratings['movieId'] = movie_encoder.fit_transform(ratings['movieId'])

# Get the number of users and movies
num_users = ratings['userId'].nunique()
num_movies = ratings['movieId'].nunique()

# Prepare input features
X = ratings[['userId', 'movieId']]
y = ratings['rating'].values

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the Transformer-based recommendation model
class TransformerRecommender(Model):
    def __init__(self, num_users, num_movies, embed_dim, num_heads, ff_dim, num_transformer_blocks, rate=0.1):
        super(TransformerRecommender, self).__init__()

        # Embedding layers
        self.user_embedding = Embedding(num_users, embed_dim)
        self.movie_embedding = Embedding(num_movies, embed_dim)

        # Transformer layers
        self.transformer_blocks = [
            [
                MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim),
                LayerNormalization(epsilon=1e-6),
                Dense(ff_dim, activation="relu"),
                Dense(embed_dim),
                Dropout(rate)
            ]
            for _ in range(num_transformer_blocks)
        ]

        # Dense output layers
        self.global_pooling = GlobalAveragePooling1D()
        self.dropout = Dropout(rate)
        self.dense1 = Dense(ff_dim, activation='relu')
        self.dense2 = Dense(1, activation='linear')
```

```
    def call(self, inputs):
        user_id, movie_id = inputs

        # Embedding lookup
        user_emb = self.user_embedding(user_id)
        movie_emb = self.movie_embedding(movie_id)

        # Stack user and movie embeddings to create a sequence
        x = tf.stack([user_emb, movie_emb], axis=1)

        # Pass through transformer blocks
        for mha, ln, ff1, ff2, dropout in self.transformer_blocks:
            attn_output = mha(x, x)
            x = ln(x + attn_output)
            ff_output = ff1(x)
            ff_output = ff2(ff_output)
            x = ln(x + dropout(ff_output))

        # Global pooling and dense layers
        x = self.global_pooling(x)
        x = self.dropout(x)
        x = self.dense1(x)
        output = self.dense2(x)

        return output

# Model hyperparameters
embed_dim = 32
num_heads = 4
ff_dim = 128
num_transformer_blocks = 2
dropout_rate = 0.1

# Instantiate and compile the model
model = TransformerRecommender(num_users=num_users, num_movies=num_movies, embed_dim=embed_dim,
                               num_heads=num_heads, ff_dim=ff_dim, num_transformer_blocks=num_transformer_blocks,
                               rate=dropout_rate)

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss='mean_squared_error',
              metrics=['mean_absolute_error'])

# Model summary
model.build([(None,), (None,)])  # Fixes the build issue
model.summary()
```

```python
# Prepare input data for training
train_user_input = X_train['userId'].values
train_movie_input = X_train['movieId'].values
test_user_input = X_test['userId'].values
test_movie_input = X_test['movieId'].values

# Start timer for training time
start_time = time.time()

# Train the model
history = model.fit([train_user_input, train_movie_input], y_train, epochs=10, batch_size=64,
                    validation_data=([test_user_input, test_movie_input], y_test))

# End timer for training time
training_time = time.time() - start_time

# Predict for a specific user
user_id = 1

# Get the movies that user 1 has already rated
rated_movie_ids = ratings[ratings['userId'] == user_id]['movieId'].values

# Find the movies that user 1 has not rated
unrated_movie_ids = np.setdiff1d(np.array(range(num_movies)), rated_movie_ids)

# Prepare the input for prediction
user_input = np.full_like(unrated_movie_ids, user_id)
movie_input = unrated_movie_ids

# Predict ratings for all unrated movies
predicted_ratings = model.predict([user_input, movie_input]).flatten()

# Clip the predicted ratings to be within the range [1, 5]
predicted_ratings = np.clip(predicted_ratings, 1, 5)

# Get the top 10 movie indices
top_10_indices = np.argsort(predicted_ratings)[-10:][::-1]

# Get the top 10 movieIds and their predicted ratings
top_10_movie_ids = unrated_movie_ids[top_10_indices]
top_10_ratings = predicted_ratings[top_10_indices]

# Display the top 10 recommendations
recommended_movies = pd.DataFrame({
    'movieId': top_10_movie_ids,
    'predicted_rating': top_10_ratings
})

# Merge with the movies dataframe to get the movie titles
recommended_movies = pd.merge(recommended_movies, movies, on='movieId')

print("Top 10 movie recommendations for userId 1:")
print(recommended_movies[['movieId', 'title', 'predicted_rating']])
```

Model: "transformer_recommender"

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
embedding_1 (Embedding)	?	0 (unbuilt)
multi_head_attention (MultiHeadAttention)	?	0 (unbuilt)
layer_normalization (LayerNormalization)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)
dense_1 (Dense)	?	0 (unbuilt)
dropout (Dropout)	?	0 (unbuilt)
multi_head_attention_1 (MultiHeadAttention)	?	0 (unbuilt)
layer_normalization_1 (LayerNormalization)	?	0 (unbuilt)
dense_2 (Dense)	?	0 (unbuilt)
dense_3 (Dense)	?	0 (unbuilt)
dropout_1 (Dropout)	?	0 (unbuilt)
global_average_pooling1d (GlobalAveragePooling1D)	?	0 (unbuilt)
dropout_2 (Dropout)	?	0 (unbuilt)
dense_4 (Dense)	?	0 (unbuilt)
dense_5 (Dense)	?	0 (unbuilt)

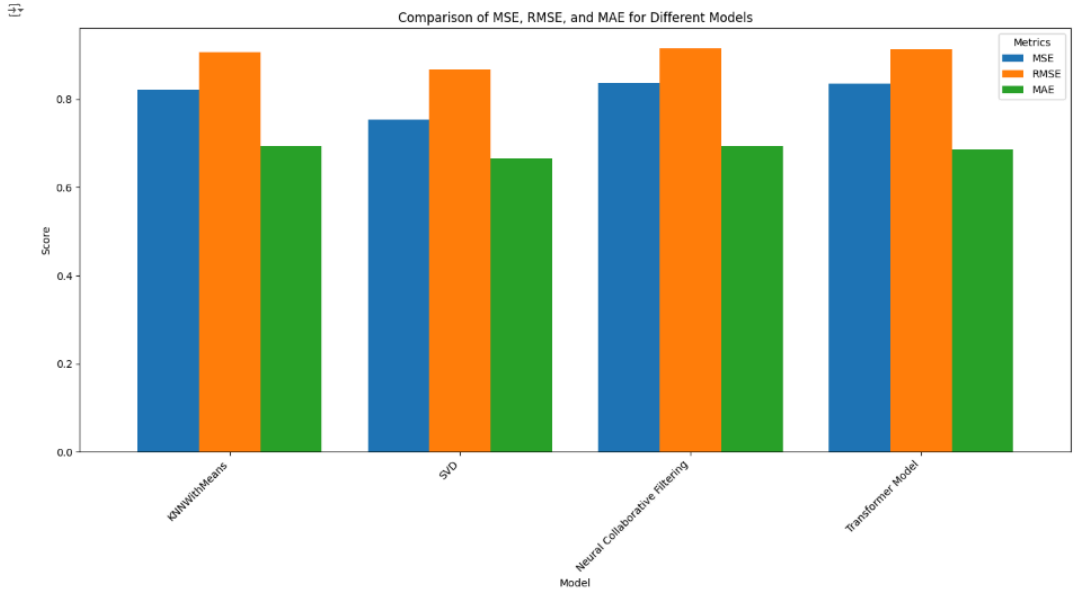
Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)

```
Epoch 1/10
1261/1261 ━━━━━━━━━━━━━━━━━━━━ 28s 8ms/step - loss: 1.1607 - mean_absolute_error: 0.8090 - val_loss: 0.7755 - val_mean_absolute_error: 0.6717
Epoch 2/10
1261/1261 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - loss: 0.6795 - mean_absolute_error: 0.6300 - val_loss: 0.7604 - val_mean_absolute_error: 0.6678
Epoch 3/10
1261/1261 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 0.6300 - mean_absolute_error: 0.6033 - val_loss: 0.8080 - val_mean_absolute_error: 0.6720
Epoch 4/10
1261/1261 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - loss: 0.5910 - mean_absolute_error: 0.5819 - val_loss: 0.7920 - val_mean_absolute_error: 0.6738
Epoch 5/10
1261/1261 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - loss: 0.5603 - mean_absolute_error: 0.5647 - val_loss: 0.7796 - val_mean_absolute_error: 0.6773
Epoch 6/10
```

```
[ ] from sklearn.metrics import mean_squared_error, mean_absolute_error
    import numpy as np

    # Use the predictions from the model
    y_pred = model.predict([test_user_input, test_movie_input]).flatten()

    # Calculate Mean Squared Error (MSE)
    mse = mean_squared_error(y_test, y_pred)

    # Calculate Root Mean Squared Error (RMSE)
    rmse = np.sqrt(mse)

    # Calculate Mean Absolute Error (MAE)
    mae = mean_absolute_error(y_test, y_pred)

    # Print the results
    print(f"Training Time: {training_time:.2f} seconds")
    print(f"Mean Squared Error (MSE): {mse}")
    print(f"Root Mean Squared Error (RMSE): {rmse}")
    print(f"Mean Absolute Error (MAE): {mae}")
```

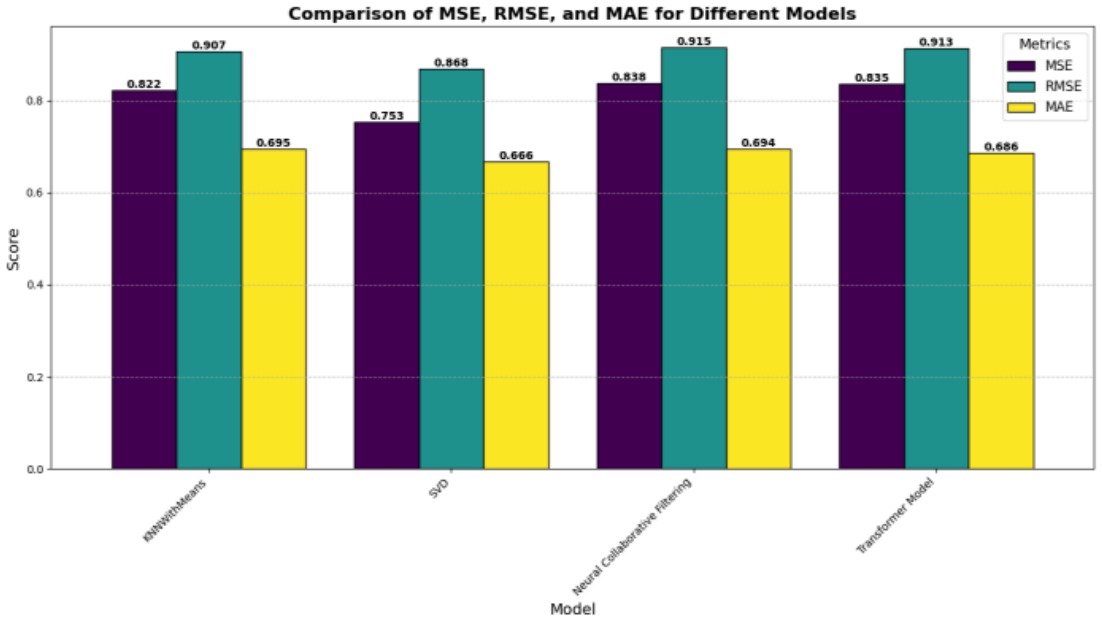
```
631/631 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
Training Time: 92.82 seconds
Mean Squared Error (MSE): 0.8351466006020511
Root Mean Squared Error (RMSE): 0.9138635568847524
Mean Absolute Error (MAE): 0.6865336244503972
```

```
import matplotlib.pyplot as plt
import pandas as pd

data = {
    'Model': ['KNNWithMeans', 'SVD', 'Neural Collaborative Filtering', 'Transformer Model'],
    'MSE': [0.822,0.753, 0.838, 0.835],
    'RMSE': [0.907,0.868, 0.915 , 0.913],
    'MAE': [0.695, 0.666, 0.694, 0.686]
}

df = pd.DataFrame(data)

# Plot
df.plot(x='Model', kind='bar', figsize=(14, 8), width=0.8)
plt.title('Comparison of MSE, RMSE, and MAE for Different Models')
plt.xlabel('Model')
plt.ylabel('Score')
plt.xticks(rotation=45, ha='right')
plt.legend(title='Metrics')
plt.tight_layout()
plt.show()
```

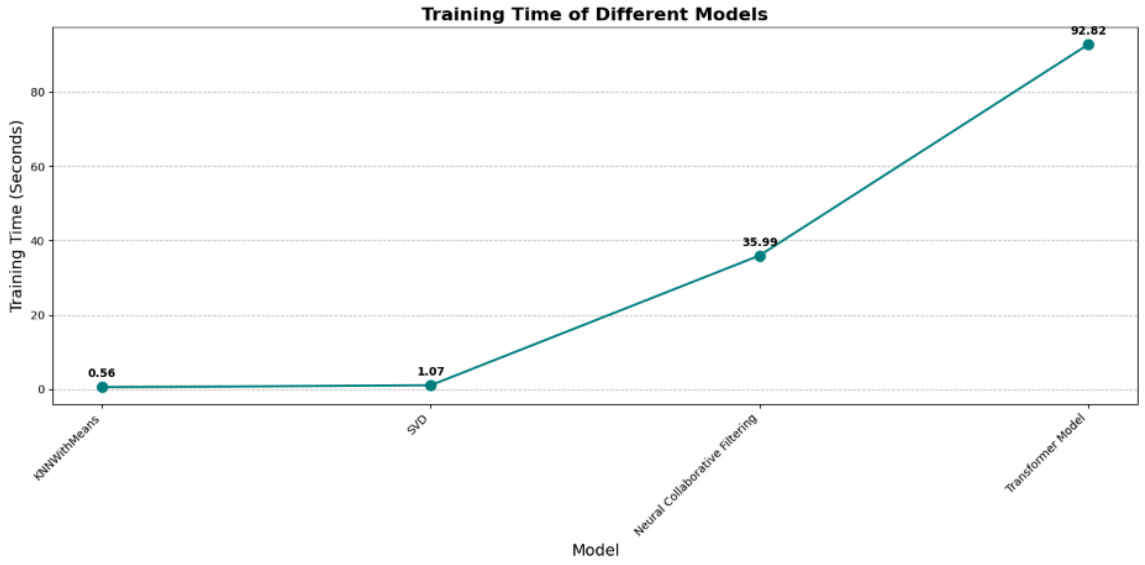


```python
import matplotlib.pyplot as plt
import pandas as pd

data = {
    'Model': ['KNNWithMeans', 'SVD', 'Neural Collaborative Filtering', 'Transformer Model'],
    'MSE': [0.822,0.753, 0.838, 0.835],
    'RMSE': [0.907,0.868, 0.915 , 0.913],
    'MAE': [0.695, 0.666, 0.694, 0.686]
}

df = pd.DataFrame(data)

# Set figure size
plt.figure(figsize=(14, 8))

# Plot
ax = df.plot(x='Model', kind='bar', figsize=(14, 8), width=0.8, colormap='viridis', edgecolor='black')

# Add data labels
for container in ax.containers:
    ax.bar_label(container, fmt='%.3f', label_type='edge', fontsize=10, color='black', weight='bold')

# Customize the plot
plt.title('Comparison of MSE, RMSE, and MAE for Different Models', fontsize=16, weight='bold')
plt.xlabel('Model', fontsize=14)
plt.ylabel('Score', fontsize=14)
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.legend(title='Metrics', title_fontsize='13', fontsize='12')
plt.tight_layout()
plt.show()
```

```
<Figure size 1400x800 with 0 Axes>
```



```python
import matplotlib.pyplot as plt

# Data
models = ['KNNWithMeans', 'SVD','Neural Collaborative Filtering','Transformer Model']
training_times = [0.56, 1.07,35.99, 92.82]

# Plot
plt.figure(figsize=(14, 7))
plt.plot(models, training_times, marker='o', linestyle='-', color='teal', markersize=8, linewidth=2, markeredgewidth=2)

# Add data labels
for i, txt in enumerate(training_times):
    plt.text(models[i], training_times[i] + 2, f'{txt:.2f}', ha='center', va='bottom', fontsize=10, weight='bold')

# Customize the plot
plt.title('Training Time of Different Models', fontsize=16, weight='bold')
plt.xlabel('Model', fontsize=14)
plt.ylabel('Training Time (Seconds)', fontsize=14)
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



```python
import seaborn as sns
import matplotlib.pyplot as plt

metrics = {
    'Model': ['KNNWithMeans', 'SVD', 'Neural Collaborative Filtering', 'Transformer Model'],
    'MSE': [0.822,0.753, 0.838, 0.835],
    'RMSE': [0.907,0.868, 0.915 , 0.913],
    'MAE': [0.695, 0.666, 0.694, 0.686]
}

df_metrics = pd.DataFrame(metrics).set_index('Model')

# Plot
plt.figure(figsize=(12, 8))
sns.heatmap(df_metrics, annot=True, cmap='YlGnBu', fmt='.3f')
plt.title('Heatmap of MSE, RMSE, and MAE for Different Models')
plt.xlabel('Metric')
plt.ylabel('Model')
plt.tight_layout()
plt.show()
```