Continual Learning pipeline for Neural Collaborative Filtering

Rafael Fernandes Gonçalves
DETI - Universidade de Aveiro
Aveiro, Portugal
rfg@ua.pt
Student ID: 102534

Daniel Jorge Bernardo Ferreira
DETI - Universidade de Aveiro
Viseu, Portugal
djbf@ua.pt
Student ID: 102885

Abstract—**Recommendation systems are widely used in the industry to provide personalised recommendations to users, but research often emphasises the algorithm performance on a static dataset, overlooking the dynamic addition of new users and items. In this work, we first explore three Neural Collaborative Filtering (NCF) methods for predicting item ratings on the 'MovieLens 1M' dataset. Then, using the best model and its optimal configuration, we build an end-to-end engineering solution for continual learning from scratch with Apache Kafka, Apache Spark and Flask. It starts with an initial offline training of the selected model using 1 million samples from the 'MovieLens 25M' dataset. Afterwards, the model is fine-tuned in Spark with the incoming batches of user-item interactions, which are generated from the remaining 24 million samples. Finally, we deploy the model in a Flask server, for inference. In order to ensure horizontal scalability and handle a large number of user-item interactions, the entire pipeline is Docker-ready.**

Index Terms—**Recommendation Systems, Neural Collaborative Filtering, Data Engineering, Continual Learning, Fine-tuning, Deep Learning, AutoEncoder**

I. INTRODUCTION

In virtue of the increasing amount of digital content available, businesses are leveraging recommendation systems to provide personalised recommendations to users, and a variety of strategies have been put forth in an effort to guarantee customer loyalty and satisfaction [1]. Among the emerging methods, content-based and collaborative filtering stood out as the most popular ones. The first type of methods relies on item features (metadata, tags, description, etc.) to suggest new items similar to those the user has liked or interacted with. Although less affected by data sparsity, the content-based approach does not introduce diversity and novelty to the user, as it tends to recommend the type of content for which the user already has positive feedback [2]. From a business perspective, this is not ideal by the fact that it does not help to create viral content. Besides, the item features can be complex (text, images, videos), requiring computationally demanding methods. On the other hand, Collaborative Filtering (CF) methods take advantage of the community feedback by recommending items to a user based on the past behaviour of similar users. This method is more effective in introducing new content that would otherwise be unknown to particular users and in creating a chain reaction of positive feedback. However, CF methods are more prone to the cold start problem of data sparsity, and also face scalability issues when dealing with a large number of users and items [3].

One of the major applications of CF methods can be found in video streaming platforms (e.g. Netflix, HBO, Amazon Prime Video), where new items and users are constantly being added. In this context, a recommendation relying on a Machine Learning (ML) or Deep Learning (DL) model should not be only trained offline on a static dataset. Instead, it should be able to learn in an online fashion from new incoming batches of user-item interactions, preferably without forgetting the knowledge learned in the past. This approach is known as Continual Learning (CL) [4] and is often complementary with MLOps, a set of operational practices that aims to streamline the ML lifecycle, from development to deployment [5].

Aware of the importance of dynamic recommendation systems, in the scope of the second assignment of 'Complements of Machine Learning' (Complementos de Aprendizagem Automática), we harness the widely used 'MovieLens 1M' dataset [1] (about 1 million samples) to propose three Neural Collaborative Filtering (NCF) methods for predicting item ratings, and also the 'MovieLens 25M' dataset [2] (about 25 million samples) to build an end-to-end engineering solution for continual learning, made from scratch with Apache Spark, Apache Kafka and Flask. The idea is to first select the best model and the optimal hyperparameters on the 'MovieLens 1M' dataset, allowing a fair comparison between our models and the state-of-the-art. Then, we train the chosen model offline using 1 million random samples whose user and movie IDs are present in the 25 million dataset. The remaining 24 million samples are sent periodically, one at a time, to the Kafka broker. In Spark, we consume these samples in form of batches and fine-tune the model. Finally, we deploy the model to a Flask server.

II. STATE-OF-THE-ART

The interest of companies in keeping their clients satisfied have driven research in recommendation systems. As a result, there is a high level of scientific production in CF methods. In [6], the authors present a literature review on CF methods

[1]https://grouplens.org/datasets/movielens/1m/
[2]https://grouplens.org/datasets/movielens/25m/

and categorise them into three types: memory-based, model-based and hybrid (a combination of the previous two), as Fig. 1 illustrates.

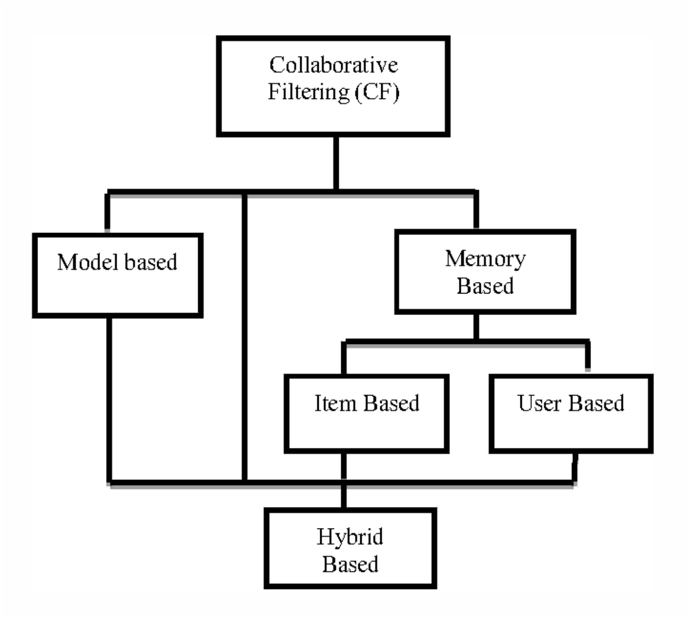

Fig. 1: Types of Collaborative Filtering methods [6].

Memory-based CF methods store all user-item interactions, and use nearest neighbour techniques to predict ratings for a target user based on the ratings of similar users. These techniques may struggle with ”cold-start” problems, where new users or items have no ratings, and also don't scale well. Model-based CF methods, on the other hand, leverage ML or statistical algorithms to learn a model from the user-item interactions. There are two classical approaches in this category: item-based, where the model learns the similarity between items, and user-based, where the model learns the similarity between users. However, they still suffer from data sparsity and scalability issues. In [7], other two modern variants capable of predicting ratings for unseen items/users are discussed: Matrix Factorisation (MF) and Neural Collaborative Filtering (NCF).

MF methods factorise the user-item interaction matrix into two low-rank matrices (user matrix and item matrix). Each matrix represents the latent features of users and items, respectively. The NCF approach presented in the paper is similar to a MF, but the inner product of latent features is replaced by a neural network. The authors defend that, despite the widespread use of neural networks, they should be used with caution, as they may not always outperform MF. We therefore decided to look not only for NCF methods, but also for MF methods to serve as a baseline for our models in the initial offline training. We first found a Probabilistic Matrix Factorisation (PMF) method in [8] that was originally evaluated on the Netflix dataset. In another paper [9], this method was tested on the 'MovieLens 1M' dataset achieving a Root Mean Squared Error (RMSE) of 0.9060. Regarding NCF methods, our main references were TmTx (Time, Text) and TmTI (Time, Text, Item correlation) [10]. Both apply convolutional neural networks and incorporate temporal dynamics, mitigating the cold-start problem, but the latter introduces item correlation into the model. The authors claim a RMSE of 1.0574 for TmTx and 0.9739 for TmTI on the MovieLens-1M dataset.

Research in CF methods often focuses solely on the offline training of models, but the user-item interactions are constantly changing in a real-world scenario. In this sense, the aforementioned paper [4] adopts a continual learning perspective for CF and gave us insights into the main challenge of this paradigm: catastrophic forgetting. This phenomenon occurs when a model trained on a static dataset is fine-tuned with new data, causing it to forget the knowledge learned from the past. Their solution is based on gradient alignment, i.e, combining different tasks in a single optimisation problem, but its hard implementation may not compensate the benefits. According to [11], dropout regularisation may also help alleviate catastrophic forgetting. In the most cases, dropout increases the optimal network size, and the authors believe that larger networks are less prone to forgetting. Based on this insight, we use dropout in our NCF models.

III. Dataset Analysis

A. Data Description

MovieLens is originally a movie recommendation service that also uses CF to suggest movies to its users and was launched in 1997 by the GroupLens Research. This research lab at the University of Minnesota have been collecting datasets from the website and sharing them with the community, so there are several versions of the dataset, which are widely used in research.

For assessing the performance of the developed models and comparing them to the state-of-the-art, we rely on the 'MovieLens 1M' dataset. This version contains 3 files: 'movies.dat', 'ratings.dat' and 'users.dat'. We are only interested in the 'ratings.dat' file, which contains 1,000,209 anonymous ratings of approximately 3,706 movies made by 6,040 MovieLens users. Each dataset row is in the format 'userId::movieId::rating::timestamp'. The 'userId' and 'movieId' columns are integers representing the user and movie identifiers, respectively. The 'rating' column is an integer from 1 to 5 representing the rating given by the user to a movie. The 'timestamp' column is an integer representing the time at which the rating was given.

For the initial offline training of the best discovered model and the continual learning pipeline, we use the 'MovieLens 25M' dataset. This version contains 6 CSV files: 'genome-scores.csv', 'genome-tags.csv', 'links.csv', 'movies.csv', 'ratings.csv' and 'tags.csv'. Once again, we only need the ratings file, which contains 25,000,095 anonymous ratings of 59047 movies made by 162541 users. Each dataset row is in the format 'userId,movieId,rating,timestamp'. The 'userId' and 'movieId' are still integers from 1 to 5, but a key difference is that the 'rating' column is now a float and can also take the following intermediate values: 1.5, 2.5, 3.5 and 4.5.

B. Exploratory Data Analysis (EDA)

EDA is crucial for building effective CF methods, as it helps identify data sparsity and the uneven distribution of user and item interactions. In this stage, after checking the number of distinct users and movies and analysing the structure of the datasets (see subsection III-A), we plotted the value distribution of the ratings in the 1M dataset (Fig. 2) and the 25M dataset (Fig. 3). If we round up the ratings to the nearest whole number (e.g. 3.5 to 4), we see that in both datasets the most common rating is 4, followed by 3, 5, 2 and 1.

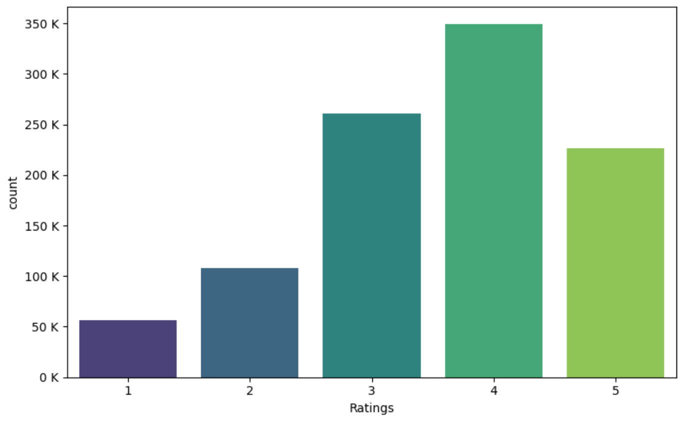

Fig. 2: Ratings distribution in the MovieLens 1M dataset.

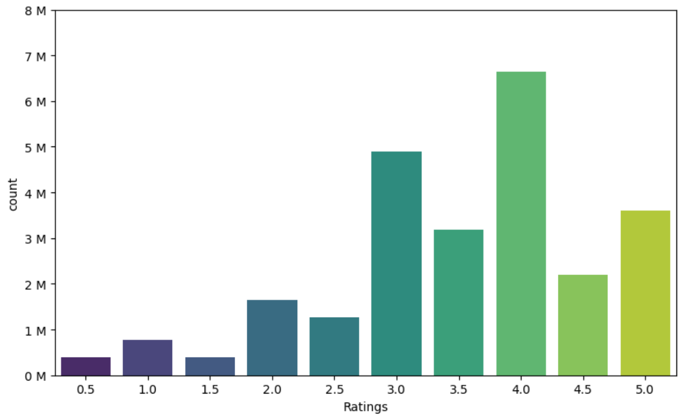

Fig. 3: Ratings distribution in the MovieLens 25M dataset.

The mean rating is also very similar: 3.5816 and 3.5339 for the 1M and 25M datasets, respectively, so we can infer that the users tend to rate movies they like.

If we define a dense CF dataset as one in which all users have rated all items, we can calculate the density of the datasets as the ratio of the actual number of ratings to the total number of possible ratings, where the latter is the product of the number of distinct users and the number of distinct movies). The small dataset's density is thus 4.4684% and the larger dataset's density is far less: 0.2605%. From the observed data sparsity, we can expect many users and items to have few ratings - which is a challenge for our models.

To understand the likelihood of a user rating a movie and vice versa, we performed a statistical analysis on the counts of each entity, as tables I and II show.

TABLE I: Descriptive statistics of the MovieLens 1M dataset

Statistic	**User count**	**Movie count**
Mean	165.60	269.89
Minimum	20	1
25% quantile	44	33
50% quantile	96	123
75% quantile	208	350
Maximum	2314	3428

TABLE II: Descriptive statistics of the MovieLens 25M dataset

Statistic	**User count**	**Movie count**
Mean	153.81	423.39
Minimum	20	1
25% quantile	36	2
50% quantile	71	6
75% quantile	162	36
Maximum	32202	81491

We found that each user rated at least 20 movies in both datasets, and there are movies with only 1 rating. From the mean and the quantiles of the counts, we concluded that the distribution of users and items is very skewed. For instance, note that half of the users in the 25M dataset rated at most 71 movies, but the maximum number of ratings per user is 32202. Therefore, some user ratings may be much harder to predict than others.

C. Data Preprocessing

Since our models do not incorporate temporal information, we discard the 'timestamp' column from the datasets. We also normalise the ratings to the range [0, 1] before feeding them into the models.

Regarding the model comparison phase, we convert the 'ratings.dat' file to a CSV file and split it into training and testing sets with a 90/10 ratio. Recall that the small dataset has approximately 1 million samples, so we have in this stage about 900k samples for training and 100k samples for testing. After conducting hyperparameter tuning, as described later in section IV, we perform a k-fold cross-validation on the top 5 configurations.

As for the 'MovieLens 25M' dataset, we randomly select 1 million samples for a initial offline training of the best model found. From the remaining 24 million samples, we take only 4% of them (about 960k samples) to create a static test set. The other samples are used for fine-tuning the model in the continual learning pipeline. Furthermore, as detailed in section V, the batches received in Spark are stored in a buffer dataset, which is splitted into training and validation sets with a 80/20 ratio. The validation set is used to monitor the model's performance on the distribution of the new incoming samples and the test set is used to evaluate the model's performance on the original data distribution.

The table III summarises the datasets split.

TABLE III: Data split of the MovieLens 25M dataset.

Source	Purpose	Samples
MovieLens 1M	Model selection	1M
MovieLens 25M	Initial offline training	1M
MovieLens 25M	Continual Learning (test)	960k
MovieLens 25M	Continual Learning (train/val)	23.04M

Finally, we cannot overlook the fact that we don't know all user and movie IDs in advance. Moreover, they were not assigned continuously. To address this issue, we create two dictionaries, 'user2idx' and 'item2idx', that map the original IDs to auto-incremented indices (1, 2, 3, ...). In the initial offline training, we initialise these dictionaries and store them as Pickle files. In the continual learning pipeline, we load the dictionaries from the Pickle files and update them with any new IDs from incoming batches.

IV. Model implementation

In this section, we cover the initial offline training of three Neural Networks (NN) using Torch, leveraging the Movielens 1M dataset.

The hyperparameter tuning for each model is performed using Ray Tune, a scalable hyperparameter optimisation framework [12]. This framework facilitates efficient and distributed tuning, ensuring that we identify the optimal configuration for each network. We utilise the ASHAScheduler (Adaptive Stochastic Hyperparameter Async Scheduling) with a predefined configuration to manage the individual trial runs. This configuration aims to minimise the evaluation loss during training. Each trial is limited to a maximum of 10 iterations. To prevent premature termination of promising trials, a grace period of 1 iteration is implemented before ASHAScheduler considers stopping a trial. Additionally, a reduction factor of 2 is used to prioritise high-performing trials by aggressively pruning trials with lower evaluation loss. It's important to note that the number of training samples used within each trial varied across different models.

The primary metric used to evaluate the performance of our models is the Root Mean Square Error (RMSE). This metric quantifies the average magnitude of the difference between predicted and observed values. It is particularly useful in regression tasks. In this case, the objective is to minimise the error between the predicted ratings and the actual ratings. The RMSE is defined as:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{1}$$

where y_i represents the actual rating, $\hat{y}_i$ represents the predicted rating, and N is the number of ratings.

For each model, we select the top 5 configurations resulting from the hyperparameter tuning process and follow a cross-validation strategy with 10 folds. We first split the dataset into 10 equally sized folds. The model is trained on 9 folds (training set) and evaluated on the remaining fold (validation set) in a round-robin fashion. This process is repeated 10 times varying the validation fold. During this process, we monitor the loss for each split to ensure that the models are converging properly and generalise well to unseen data.

The configuration with the best cross-validation score, as measured by the average RMSE across the 10 folds, is selected as the best configuration for each model. This approach ensures that we choose the model that consistently performs well across different subsets of the data.

A. *Non-Linear*

In traditional collaborative filtering, matrix factorisation is a technique used to predict user ratings for items by decomposing the user-item rating matrix R into two lower-dimensional matrices, U (user latent factors) and V (item latent factors). These matrices are optimised iteratively to minimise the loss between the actual and predicted ratings through techniques such as gradient descent. The linear version of this approach combines these matrices via dot products to form $\hat{R}$, a reconstructed rating matrix. This is conceptually straightforward, treating the dot product as a similarity metric between user and item latent factors, which results in $\hat{R}$, a matrix of predicted ratings based on these similarities.

However, the linear model has limitations, especially when dealing with complex and large datasets, where interactions between users and items might be more intricate than what a simple dot product can capture. Here, a non-linear neural network (NN) model provides a more powerful alternative. Instead of relying on matrix multiplication, the NN model uses non-linear transformations to capture more complex interactions between user and item embeddings.

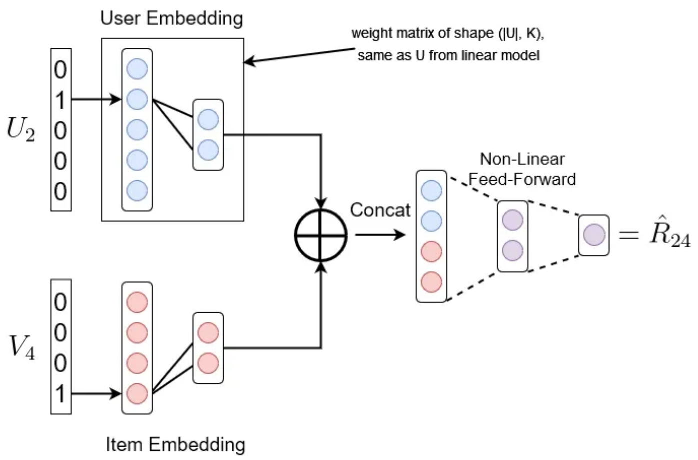

Fig. 4: Non-linear model architecture: single sample prediction of User 2 and Item 4 (where $|U| = 5$, $|V| = 4$, $K = 2$) [13].

In our non-linear approach, we replace the dot product with a neural network that can learn a more flexible, parameterised similarity metric. The user and item embeddings are first learned through embedding layers, which act as look-up tables for these latent factors. These embeddings are then concatenated and passed through several non-linear transformations using fully connected (dense) layers with activation functions like ReLU (Rectified Linear Unit), followed by dropout regularisation to prevent overfitting. The conceptual architecture, depicted in Fig. 4, enables the model to learn complex, non-linear patterns that a simple dot product cannot capture, and is particularly beneficial in dealing with sparse and diverse datasets like movie ratings, where user preferences and item characteristics might interact in non-trivial ways.

To implement this, we used a training procedure with a MSE loss function, which is robust and penalises large prediction errors more significantly. For optimisation, we use the Adam optimiser, known for its efficiency in handling sparse gradients, which is common in recommendation systems.

The hyperparameter tuning was performed using Ray Tune with an ASHAScheduler to find the best configuration for the following parameters:

- **Embedding dimension**: This controls the size of the latent factors learned by the model. Increasing the embedding dimension can allow the model to capture more complex relationships, but it can also lead to overfitting if not chosen carefully.
- **Dropout rate**: This technique randomly drops neurons during training to prevent overfitting. A higher dropout rate means more neurons are dropped, which can help reduce model complexity.
- **Learning rate**: This controls the step size during gradient descent optimisation. A larger learning rate can lead to faster convergence but may cause the model to miss the optimal solution. Conversely, a smaller learning rate can lead to slower convergence but might find a better minimum.

The search space for hyperparameter tuning is shown in Table IV. Note that the learning rate uses a log-uniform distribution, signifying an exponential search across different magnitudes.

TABLE IV: Configuration Space for the Non-Linear Model.

Hyperparameter	**Values**
Embedding Dimension	{32, 64, 128, 256, 512, 1024}
Dropout Rate	{0.0, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5}
Learning Rate	$\{1 \times 10^{-4}$ to 1×10^{-2} (log-uniform)$\}$
Max Epochs	100
Batch size	1024

Testing 100 random configurations with Ray Tune, we were able to balance exploration (trying new configurations) and exploitation (focusing on promising configurations) during the search process. Table V summarises the configuration with the best average cross-validation score (lowest RMSE).

TABLE V: Best Configuration for the Non-Linear Model.

Hyperparameter	**Selected Value**
Embedding Dimension	512
Dropout Rate	0.2
Learning Rate	0.001089
Epochs	18
Batch Size	1024

Fig. 5 illustrates the distribution of validation losses across 10 folds for the best configuration of the non-linear model. The box plots provide insights into the variability and central tendency of the validation losses, highlighting the model's consistency across different validation sets. Each box plot shows the inter-quartile range, median (orange line), mean (green triangle), and potential outliers (circles), offering a comprehensive view of the model's performance stability.

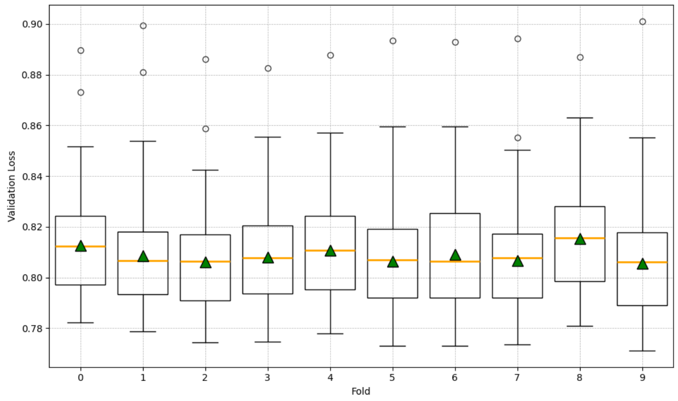

Fig. 5: MSE loss distribution across 10 folds for the best configuration of the Non-Linear model.

Fig. 6 presents the training and validation loss convergence for one fold (true rating scale). It demonstrates the reduction in loss over 100 epochs, with the training loss decreasing steadily while the validation loss stabilises early. The black dashed line indicates the epoch with the lowest validation loss, suggesting the optimal point for early stopping (epoch 18).

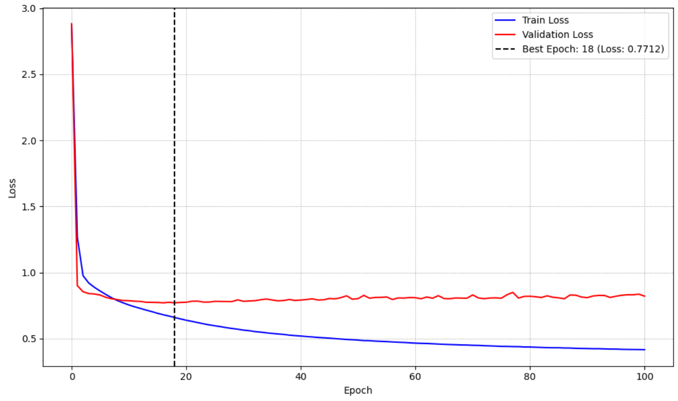

Fig. 6: Example of the training and validation loss convergence of the Non-Linear model for one fold.

The results, as depicted in these figures, confirm the effectiveness of the non-linear approach in capturing complex interactions, achieving a reliable RMSE score of 0.8817. This RMSE refers to the minimum average cross-validation score for each fold, using the best epoch identified for this metric.

B. AutoEncoder (AE)

Similar to the non-linear model, the AutoEncoder (AE) model aims to address the limitations of linear collaborative filtering by capturing complex user-item interactions. However, it takes a different approach to achieve this.

AutoEncoders represent a class of neural network architectures traditionally used for unsupervised learning tasks, particularly dimensionality reduction and data reconstruction. In the context of collaborative filtering, AEs can be utilised to capture latent factors of users and items, providing a robust method for predicting user preferences.

The AE architecture can be visualised as having two interconnected components: an encoder and a decoder. The encoder compresses the concatenated user and item embeddings into a latent representation, while the decoder reconstructs the interaction (rating) from this latent space.

- **Encoder**: The core functionality lies within the encoder network. It takes the concatenated user and item embeddings as input, representing the user-item interaction. This combined vector is then fed through a series of fully-connected layers with non-linear activation functions (like ReLU) and dropout for regularisation. These layers progressively compress the high-dimensional combined vector into a lower-dimensional latent representation. This latent representation acts as a bottleneck layer, forcing the model to identify the most critical aspects of the user-item interaction.

- **Decoder**: The decoder network processes the latent representation generated by the encoder. It receives this latent representation and maps it to a single scalar value, representing the predicted interaction score between the user and item. The decoder's architecture consists of fully-connected layers with non-linear activation functions and dropout, similar to the encoder, but it does not expand the latent representation back to the original combined vector's dimensionality. Instead, the goal of the decoder is to predict a single output value from the latent representation. By minimising the prediction error, the AE compels the encoder to learn latent representations that effectively capture the user-item interaction.

During the training process, the AE is presented with numerous user-item pairs. It encodes the combined vector, decodes it, and strives to minimise the difference between the original and reconstructed versions. This reconstruction loss (MSE, in our case) function guides the AE to discover latent representations that faithfully encode the user-item interactions within the data. The model also uses the Adam optimiser to efficiently update the weights, during training.

The hyperparameter tuning process for the AE model was similar to the non-linear model, using Ray Tune to explore the hyperparameter space and identify the optimal configuration. The hyperparameters considered for tuning, besides the learning rate and dropout rate, were the following:

- **Hidden Dimension**: The number of neurons in the hidden layers of the encoder and decoder. This parameter controls the model's capacity to learn complex user-item interactions. A larger hidden dimension can capture more intricate patterns but may lead to overfitting if not regularised properly.

- **Latent Dimension**: The size of the latent representation learned by the encoder. This parameter determines the dimensionality of the bottleneck layer and influences the model's ability to capture the most critical aspects of the user-item interaction.

The search space for hyperparameter tuning is shown in Table VI. Note that we increased the probability of the dropout rate being 0 (no dropout) slightly to explore the possibility of the model not requiring dropout for regularisation effectively.

TABLE VI: Configuration Space for the AutoEncoder.

Hyperparameter	**Values**
Hidden Dimension	{32, 64, 128, 256, 512, 1024}
Latent Dimension	{8, 16, 32, 64, 128, 256}
Dropout Rate	{0.0, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5}
Learning Rate	{1×10^{-4} to 1×10^{-2} (log-uniform)}
Max Epochs	100
Batch Size	1024

The number of random configurations used for hyperparameter tuning was 100, the same as the non-linear model. This approach allows the algorithm to explore the hyperparameter space efficiently and identify promising configurations. Following the hyperparameter tuning process, we continued with the cross-validation step described in the previous subsection.

The best configuration found for the AE model is summarised in Table VII.

TABLE VII: Best Configuration for the AutoEncoder.

Hyperparameter	**Selected Value**
Hidden Dimension	512
Latent Dimension	8
Dropout Rate	0.0
Learning Rate	0.006923
Epochs	19
Batch Size	1024

The box plot in Figure 7 shows the MSE loss distribution across 10 cross-validation folds for the optimal AutoEncoder (AE) model configuration. This plot illustrates the consistency and variability of the model's performance across different data splits (just a few outliers).

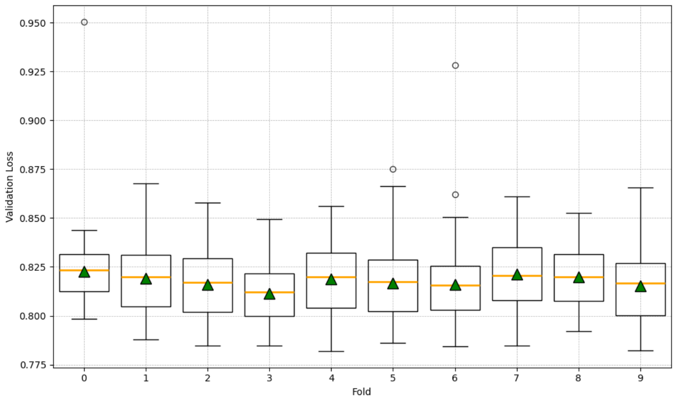

Fig. 7: MSE loss distribution across 10 folds for the best configuration of the AutoEncoder (AE) model.

In Figure 8, the training and validation loss convergence over 100 epochs (true rating scale) of the AE model is clear. The training loss steadily decreases, indicating progressive learning and improvement in fitting the training data. The validation loss initially decreases similarly but stabilises and slightly increases after approximately 21 epochs. This suggests a point of diminishing returns, where further training does not enhance generalisation and may lead to overfitting. The vertical dashed line marks the epoch with the best validation loss (epoch 21), with an optimal MSE loss value of 0.7820.

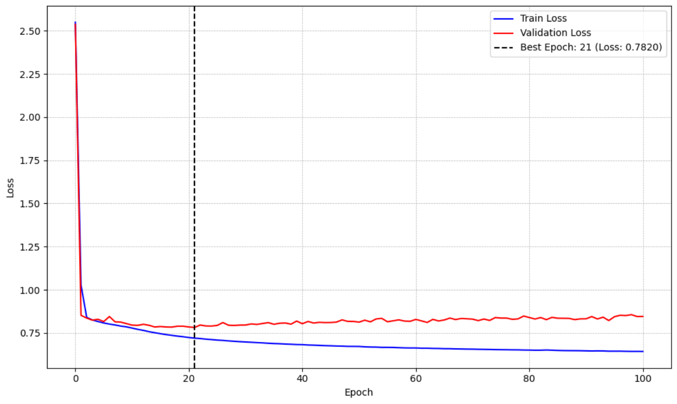

Fig. 8: Example of the training and validation loss convergence of the AutoEncoder (AE) model for one fold.

Overall, the AE model achieved a best RMSE of 0.8886, making it slightly less effective than the non-linear model. However, the AE model's performance is still competitive, demonstrating the effectiveness of the AutoEncoder architecture in capturing user-item interactions.

C. Variational AutoEncoder (VAE)

The Variational AutoEncoder (VAE) extends the capabilities of traditional AutoEncoders by introducing probabilistic components to the model, which enable it to capture the uncertainty in the latent representations. This characteristic makes VAEs particularly suitable for generating new data points and modelling complex distributions, adding robustness to the collaborative filtering process by accounting for variability in user-item interactions.

The VAE architecture also consists of an encoder and a decoder, but includes additional steps for sampling from the latent space. This enables the model to learn a distribution over the latent variables rather than a single fixed point.

- **Encoder**: The encoder network compresses the concatenated user and item embeddings into a latent representation. It follows a structure similar to the AE encoder, employing fully-connected layers with non-linear activation functions (like ReLU) and dropout for regularisation. However, instead of directly producing a latent vector, the encoder outputs two separate vectors: the mean and the log-variance of a Gaussian distribution. This probabilistic representation allows the model to capture the inherent uncertainty in the user-item interactions.

- **Sampling**: The latent representation is sampled from the Gaussian distribution parameterised by the mean and log-variance vectors. This sampling step is crucial for the VAE's ability to generate diverse user-item interaction patterns. The re-parameterisation trick is employed here to allow gradient descent optimisation, ensuring the sampling process is differentiable.

- **Decoder**: The decoder network reconstructs the interaction score from the sampled latent representation. Similar to the AE, the decoder consists of fully-connected layers with non-linear activation functions and dropout. It maps the latent vector to a single scalar value representing the predicted interaction score.

During the training process, the VAE minimises a loss function that combines the reconstruction loss (e.g., MSE) and a regularisation term (KL divergence) that encourages the learned distribution to be close to a prior Gaussian distribution. The KL divergence term acts as a regulariser, promoting the generation of realistic latent representations. The VAE model also utilises the Adam optimiser for efficient weight updates during training.

To address the vanishing or exploding gradient problem that can occur during the training, we used Kaiming (He) Initialisation [14]. In deep networks, especially those using rectified linear unit (ReLU) activation functions, traditional weight initialisation methods such as random normal or Xavier initialisation may lead to gradients that vanish or explode as they are propagated through the layers during backpropagation. Kaiming Initialisation is specifically designed for layers with ReLU activations and helps maintain the variance of the gradients throughout the network, thereby improving training stability and convergence. It sets the initial weights to values drawn from a distribution with a variance that accounts for the number of input units to the layer, thereby mitigating the risk of vanishing or exploding gradients.

The hyperparameters considered for tuning included the hidden dimension, latent dimension, dropout rate, learning rate, and additional parameters specific to VAEs:

- **Annealer**: The strategy used to schedule the beta coefficient during training. Common annealing strategies include linear, exponential, cosine, constant, and zero, each influencing how the KL divergence term's impact changes over epochs.

- **Start Beta**: The initial value of the beta coefficient used in the KL divergence term of the VAE loss function. It controls the impact of the regularisation term during early training stages.

- **Offset Beta**: The range within which the initial beta coefficient can vary during the training process. It helps in adjusting the annealing process of the KL divergence to optimise the model's performance.

These parameters collectively contribute to tuning the VAE model's architecture and training dynamics, ensuring it effectively learns and generalises from the user-item interaction data, while preventing issues like the vanishing KL divergence problem during training.

The search space for hyperparameter tuning is shown in Table VIII.

TABLE VIII: Configuration Space for the Variational AutoEncoder (VAE).

Hyperparameter	**Values**
Hidden Dimension	{32, 64, 128, 256, 512, 1024}
Latent Dimension	{8, 16, 32, 64, 128, 256}
Dropout Rate	{0.0, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5}
Learning Rate	{1×10^{-4} to 1×10^{-2} (log-uniform)}
Start Beta	{0.0001 to 0.01 (uniform)}
Offset Beta	{0.0 to 1.0 (uniform)}
Annealer	{linear, exponential, cosine, constant, zero}
Max Epochs	20
Batch Size	1024

The hyperparameter tuning process for the VAE model involved 500 random configurations. This increased number of configurations was necessary, due to the VAE's additional hyperparameters and the complexity of the model. We were not only interested in finding the best configuration, but also understanding how the annealing strategy and beta coefficient influenced the model's performance.

The best configuration found for the VAE model is summarised in Table IX. Note that the annealer strategy was set to 'zero', indicating that the beta coefficient remained constant throughout training.

TABLE IX: Best Configuration for the Variational AutoEncoder (VAE).

Hyperparameter	**Selected Value**
Hidden Dimension	256
Latent Dimension	128
Dropout Rate	0.15
Learning Rate	0.002910
Annealer	Mode 'zero'
Epochs	8
Batch Size	1024

Fig. 9 shows that the epochs loss for each fold has more outliers than in the previous models, indicating more variability in the results. A possible reason could be the reduced number of epochs, given the increased complexity of VAE.

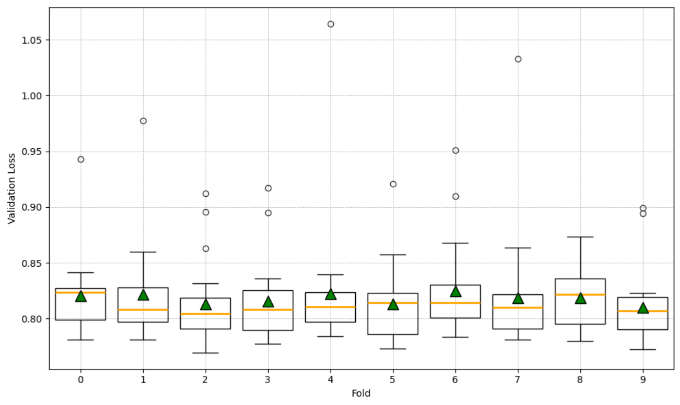

Fig. 9: MSE loss distribution across 10 folds for the best configuration of the Variational AutoEncoder (VAE).

The training and validation loss convergence of the VAE over 20 epochs (true rating scale) for a single fold is shown in Figure 11. The plot suggests that the model is learning from the training data and generalising well to unseen data. The validation loss is decreasing along with the training loss, which indicates that the model is not overfitting. It reaches the optimal point for early stopping at epoch 8, where the validation loss is minimised.

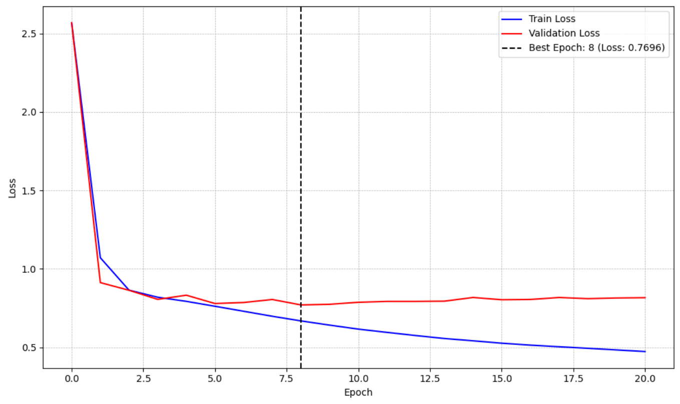

Fig. 11: Example of the training and validation loss convergence of the Variational AutoEncoder (VAE) for one fold.

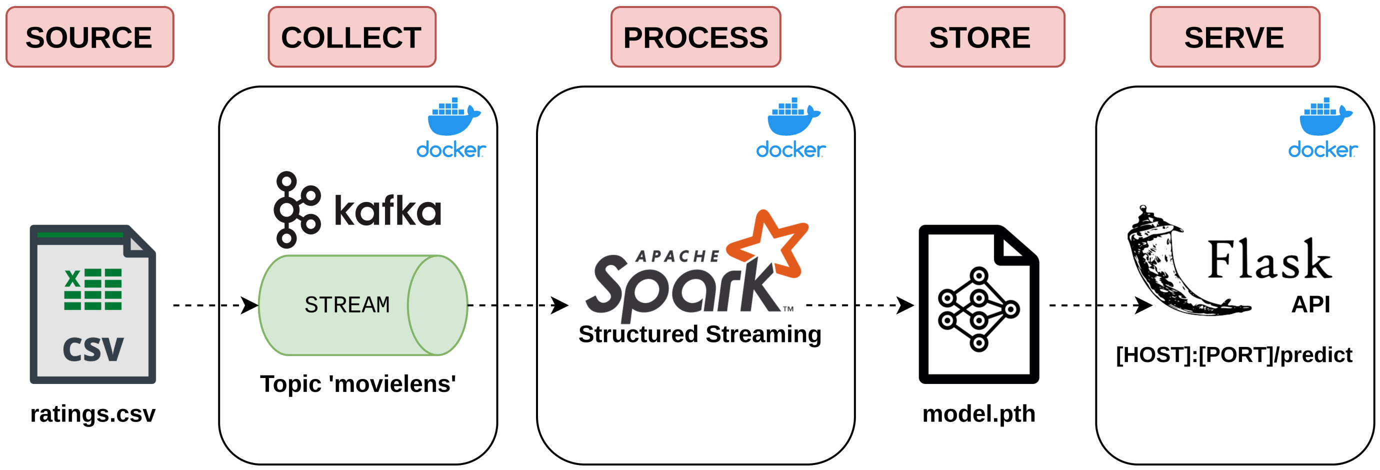

Fig. 10: Overview of the end-to-end ML pipeline for continual learning.

The VAE model achieved an RMSE of 0.8846, which is an improvement over the AE model but still falls short compared to the Non-Linear model. This result highlights the effectiveness of the VAE model in capturing the underlying data distribution, although there remains room for improvement in terms of model performance.

V. End-to-end ML Pipeline

A. Architecture Overview

Having concluded that the best model on a subset of 1 million samples is the non-linear one, we built an end-to-end ML pipeline for continual learning made from scratch. Fig. 10 gives an overview of the architecture, involving 5 main stages: fetching, collecting and streaming, batch processing, model storing and model serving. We relied on the Python's APIs for Apache Spark, Apache Kafka and Flask to implement the pipeline. In the source code, all the aforementioned components are accompanied by a Dockerfile, so that the pipeline can be easily deployed in containerised environments. The advantage of using Docker is that it allows the pipeline to easily scale horizontally, as we can add more Kafka brokers, Spark workers and Flask servers, thus being able to handle a large number of user-item interactions.

B. Data Streaming

Apache Kafka is a distributed event streaming platform that allows the creation of real-time data pipelines and streaming applications. In our pipeline, Kafka acts as a message broker, receiving the user-item interactions from the 'ratings.csv' file and sending them to the Spark application. For better portability and isolation, Kafka can be dockerised and its image can be pulled from the Docker Hub repository [3]. As it comes with Apache Zookeeper, a service that manages Kafka brokers, we had to launch both services with a Docker Compose file, since it is the simplest way to manage multi-container Docker applications.

[3]https://hub.docker.com/r/apache/kafka

Kafka works with a publish-subscribe model, where a producers send messages to topics and consumers read messages from specific topics. This is important for ensuring independence between applications, as producers and consumers do not need to know each other. In our case, we have a Python script ('producer.py') that first creates a topic called 'movielens', then reads the 24 million training samples from the file, and finally periodically sends them one by one to the Kafka broker.

C. Batch Processing

Apache Spark is a very popular distributed computing engine that provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. In our pipeline, Spark is responsible for consuming the user-item interactions from the Kafka broker, processing them in batches and fine-tuning the chosen model.

We take advantage of the PySpark's Structured Streaming API to create a streaming query that is triggered every 10 seconds. This query reads the messages from the 'movielens' topic in form of batches and accumulates them in a DataFrame, but first we load the model, the user2idx and item2idx dictionaries and the scaler from the initial offline training. During the batch processing, we check the buffer size and, when it exceeds 100,000 samples, we start the fine-tuning. We split the data into training and validation sets (80/20 ratio) and train the model from the loaded state. As further shown in section VI, to track the model performance on the validation and test sets, we log the results to the Weights and Biases platform [4]. The model is then saved to disk to be ready for deployment. Finally, we clear the buffer and wait for the next batch.

D. Model Serving

Flask is a lightweight WSGI web application framework that is used to build web applications. In our pipeline, Flask is responsible for serving the model, after loading it and its dependencies (user2idx / item2idx dictionaries and the scaler)

[4]https://wandb.ai

from the disk. It acts as a straightforward RESTful API with a single endpoint that takes a JSON payload containing the user and movie IDs and returns the predicted rating.

VI. Results

TABLE X: Model comparison on the initial offline training.

Novelty And Contributions	
Model	**RMSE**
PMF [8] (baseline)	0.9060
TmTx [10] (baseline)	1.0574
TmTI [10] (baseline)	0.9739
Non-linear	0.8817
AutoEncoder	0.8886
Variational AutoEncoder	0.8846

As Tab. X illustrates, the non-linear model (barely) outperformed the AutoEncoder and Variational AutoEncoder models, achieving a RMSE of 0.8817. This result is particularly noteworthy, as it surpasses the state-of-the-art references, PMF, TmTx and TmTI, which achieved RMSE values of 0.9060, 1.0574 and 0.9739, respectively. The non-linear model's superior performance can be attributed to its ability to capture complex user-item interactions through non-linear transformations, enabling it to generalise well to unseen data. The AutoEncoder and Variational AutoEncoder models also performed well, achieving RMSE values of 0.8886 and 0.8846, respectively. These results demonstrate the effectiveness of NCF methods in predicting movie ratings and highlight the potential of non-linear models in capturing intricate user-item interactions.

Regarding the continual learning, we ran the pipeline for several hours and logged the MSE loss of the fine-tuned model on the static test set to the Weights and Biases platform. The figure 12 shows the learning curve for the first 100 steps (fine-tuning stages).

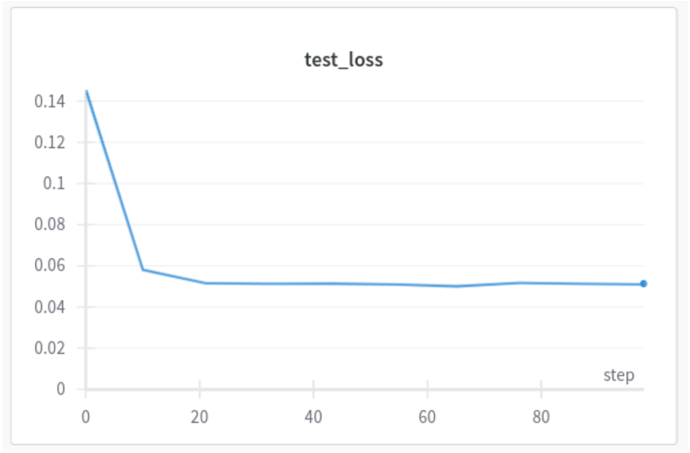

Fig. 12: MSE loss (normalised scale) on the static test set during continual learning.

We emphasise that the platform only stores the most relevant steps, which justifies the lower number of points in the graph. Besides, the loss is normalised to the range [0, 1]. We can see that it decreases over time, which is a good sign that the model is learning from the new incoming samples. Nevertheless, we found that the final RMSE (true scale) on the test set is 1.1191. Despite being a decent value in a dynamic scenario, it is a higher value than the one obtained in the MovieLens 1M dataset (model comparison phase). This result was expected due to the high data sparsity and the arrival of unseen users and items.

VII. Final Conclusions

In this work, we investigated the challenging application of Neural Collaborative Filtering (NCF) methods for predicting movie ratings on a continual learning scenario. Based on a k-fold cross-validation strategy, we concluded that the non-linear model was the best performer and managed to outperform our state-of-the-art references with a RMSE of 0.8817. Its simpler architecture may explain a better generalisation for unseen movies and users. As for the model fine-tuning, each batch presents a different data sparsity level and the data distribution may drift over time. Despite an expected higher RMSE on the static test set, when compared to the model selection phase, the network was able to learn successively from the new incoming samples. Future work should also consider an even more distinct test set to study the catastrophic forgetting phenomenon, as well as the implementation of more architectures and losses. Overall, we believe that the presented solution contributes positively to the development of dynamic recommendation systems.

VIII. Work load per student

Rafael Fernandes Gonçalves: 50%
Daniel Jorge Bernardo Ferreira: 50%

References

[1] N. Polatidis and C. Georgiadis, "Recommender Systems: The Importance of Personalization in E-Business Environments," *International Journal of E-Entrepreneurship and Innovation*, vol. 4, pp. 32–46, 10 2013.

[2] M. Phalle and P. Bhushan, "Content Based Filtering And Collaborative Filtering: A Comparative Study," *Journal of Advanced Zoology*, vol. 45, pp. 96–100, 03 2024.

[3] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, *Collaborative Filtering Recommender Systems*, pp. 291–324. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[4] J. H. Do and H. W. Lauw, "Continual Collaborative Filtering Through Gradient Alignment," in *Proceedings of the 17th ACM Conference on Recommender Systems*, RecSys '23, (New York, NY, USA), p. 1133–1138, Association for Computing Machinery, 2023.

[5] P. Ruf, M. Madan, C. Reich, and D. Ould-Abdeslam, "Demystifying mlops and presenting a recipe for the selection of open-source tools," *Applied Sciences*, vol. 11, no. 19, 2021.

[6] N. Mustafa, A. O. Ibrahim, A. Ahmed, and A. Abdullah, "Collaborative filtering: Techniques and applications," in *2017 International Conference on Communication, Control, Computing and Electronics Engineering (ICCCCEE)*, pp. 1–6, 2017.

[7] S. Rendle, W. Krichene, L. Zhang, and J. Anderson, "Neural Collaborative Filtering vs. Matrix Factorization Revisited," 2020.

[8] A. Mnih and R. R. Salakhutdinov, "Probabilistic Matrix Factorization," in *Advances in Neural Information Processing Systems* (J. Platt, D. Koller, Y. Singer, and S. Roweis, eds.), vol. 20, Curran Associates, Inc., 2007.

[9] R. Joshi, H. Dawson, and K. Roy, "A Personalized Video Recommendation Model Based on Multi-Graph Neural Network and Attention Mechanism," 03 2024.

[10] M. T. XIONG, Y. FENG, T. WU, J. X. SHANG, B. H. QIANG, and Y. N. WANG, "Tdctfic: A novel recommendation framework fusing temporal dynamics, cnn-based text features and item correlation," *IEICE Transactions on Information and Systems*, vol. E102.D, no. 8, pp. 1517–1525, 2019.

[11] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," 2015.

[12] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," *arXiv preprint arXiv:1807.05118*, 2018.

[13] M. Brenner, "Matrix factorization for collaborative filtering: Linear to non-linear models in python," 2022. Accessed: 2024-06-18.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," 2015.