Recommender systems via Covariance Neural Networks

Using precision matrices as Graph Collaborative Filter

Vic Vansteelant[1]
Supervisor(s): Dr. Elvin Isufi[1]**, Andrea Cavallo**[1]**, Chengen Liu**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Vic Vansteelant
Final project course: CSE3000 Research Project
Thesis committee: Dr. Elvin Isufi, Andrea Cavallo, Chengen Liu, Dr. Klaus Hildebrandt

Abstract

This research investigates the application of Graph Neural Networks (GNNs) for rating prediction in recommender systems, utilizing precision matrices as graph filters. The focus is on movie recommendation, where graph-based structures are especially relevant due to the importance of user and item relationships. A leave-one-out masking strategy is employed during training to ensure the model learns from all available training data. The proposed model achieves a test root mean squared error (RMSE) of approximately 0.95 on the MovieLens-100k dataset, performing reasonably well compared to existing graph-based and matrix factorization methods. While the model accurately predicts average ratings, it tends to overestimate lower ratings. These results demonstrate the potential of precision-based graph filters in GNNs but also reveal significant room for improvement before reaching state-of-the-art performance. Future work may include output calibration and sparsification of the precision matrix to enhance both efficiency and predictive accuracy.

1 Introduction

Recommender systems are used to personalize content across almost all online platforms, such as Netflix, Spotify, and Amazon, by predicting user preferences for unseen items. There are numerous ways to approach this problem. In recommender systems, there are two main approaches: the first is context-aware recommendation, which examines the features of both users and items to predict which other items the user might like. The second approach is the category into which this paper falls: collaborative-filtering recommender systems. These types of systems are based on previous interactions from the user themselves, as well as from other users. This is often done by using mathematical concepts to reduce the data and predict values for unknown user-item pairings. This paper will specifically work with movie recommender systems. These systems attempt to predict the rating, typically ranging from 1 to 5, that a user would give to a specific movie. Traditional approaches usually leverage similarities between users or items. This is frequently done using techniques such as Singular Value Decomposition (SVD), Non-negative Matrix Factorization (NMF), or nearest-neighbor algorithms, which are based on various similarity metrics (Jadon and Patil 2024). However, accurately modeling these similarities, while maintaining fast computation, especially in data-sparse environments, remains a key challenge.

To address this issue, recent papers have introduced coVariance Neural Networks (VNNs), initially explored by Sihag et al. (2022). VNNs leverage covariance

matrices as graph representations to encode statistical relationships inherent in the data. These matrices are then used to create Graph Neural Networks (Wu et al. 2021). This approach uses covariance to represent the conditional dependence of features. A promising adaptation is to use *precision matrices* instead, which are the mathematical inverse of the covariance matrices. Precision matrices capture conditional independence among variables, offering the potential for enhanced modeling of intricate relationships.

The paper is structured as follows. Section 2 provides background information and discusses related work. Section 3 presents the methodology, including the model architecture and key design decisions. Section 4 details the experimental setup, datasets, evaluation metrics, and implementation aspects. The results and their discussion are presented in Section 5. Section 6 concludes the paper with a summary of the main findings and suggestions for future research. Finally, Section 7 discusses responsible research practices and the reproducibility of results.

2 Background and Related Work

Numerous studies have explored the effectiveness of various recommender system methodologies. Matrix factorization techniques, notably SVD and NMF, have historically dominated due to their simplicity and efficiency (Koren, Bell, and Volinsky 2009). However, recent advancements increasingly leverage graph-based methods, particularly relevant for movie recommender systems, given their inherent reliance on user-item relationships that can be effectively modeled using graphs.

Recently, significant progress has been made in movie recommendation using hybrid graph-based approaches. Notably, the Graph-based Hybrid Recommendation System (GHRS), proposed by Zamanzadeh Darban and Valipour (2022), achieved state-of-the-art performance on the MovieLens datasets by integrating user-rating similarities with demographic and location data. GHRS employs an autoencoder to derive low-dimensional features, demonstrating superior accuracy and effectively addressing cold-start scenarios. While their hybrid approach shows promising results, their main downside is their reliance on demographic data.

A modern approach to these graph-based approaches is Graph Neural Networks (GNN), which has demonstrated promising results in capturing complex user-item interactions (Wu et al. 2021). A GNN is a type of neural network specifically designed to handle data structured as a graph; this means that data can be reduced to nodes and edges that represent the relationships between these nodes. These GNNs operate by aggregating features from neighboring nodes and updating each

node's representation accordingly. These updated embeddings are subsequently used for making predictions, thus effectively utilizing relational information.

Building upon this framework, coVariance Neural Networks (VNN) were introduced by Sihag et al. (2022). VNNs are specialized GNNs that utilize covariance matrices to initialize graphs, capturing statistical dependencies between variables. This causes the edges of the graph to have weights related to the statistical similarity among variables.

To further refine this approach, Cavallo, Gao, and Isufi (2024) extensively explored sparsification techniques for correlation structures. Their comparative analysis underlined the importance of managing noise in large datasets through methods such as hard thresholding, soft thresholding, and stochastic sparsification. These techniques have shown promising results but fall outside of the scope of this paper.

A promising yet underexplored variant of the VNN approach is the use of precision matrices instead of covariance matrices. A precision matrix, defined as the inverse of the covariance matrix, explicitly encodes conditional independence rather than direct statistical dependence. This characteristic potentially reduces indirect correlations, thereby refining the representation of user-item relationships and reducing noise.

In addition to predictive accuracy, modern recommender systems increasingly emphasize the importance of diversity and novelty in recommendations (Vargas and Castells 2011; Castells, Hurley, and Vargas 2021). Incorporating these aspects helps enhance user satisfaction and explore previously unseen yet relevant content, thus enriching the overall user experience. Balancing diversity, novelty, and accuracy remains a challenging task. Given the novelty of precision matrix-based VNNs, this paper prioritizes an analysis of predictive accuracy, laying the foundation for future exploration of diversity and novelty.

Despite notable advancements, research explicitly focusing on precision matrices within the VNN framework remains scarce. Given the suitability of these techniques for movie recommendation, it leaves the perfect gap for the following research question:

> *How does a coVariance Neural Network (VNN) perform on the rating prediction task when using the precision matrix as a graph collaborative filter?*

3 Methodology

3.1 Overall Architecture

This research builds upon the open-source Graph Neural Network (GNN) framework introduced by Sihag et al. (2022), which provides a general-purpose implementation for structured learning on graph-filtered data. During the research, as few modifications as possible were made to this base architecture.

The model architecture consists of multiple graph convolutional layers, followed by a multilayer perceptron (MLP). Each GNN layer applies a graph filter that aggregates information from local neighborhoods in the graph. Specifically, given input node features $\mathbf{H}^{(l)}$ at layer l, the output of the next layer is computed as:

$$\mathbf{H}^{(l+1)} = \sigma\left(\sum_{k=0}^{K-1} \tilde{\mathbf{A}}^k \mathbf{H}^{(l)} \mathbf{W}_k^{(l)}\right)$$

Here, $\tilde{\mathbf{A}}$ denotes the matrix used to propagate information across the graph, which in our case is derived from the user–user precision matrix (unlike typical GNNs that use binary or weighted adjacency matrices). K is the number of filter taps (i.e., the maximum hop distance considered), $\mathbf{W}_k^{(l)}$ is the trainable weight matrix for the k-th filter tap at layer l, and $\sigma(\cdot)$ is a non-linear activation function, which for this research was ReLu.

This formulation allows each GNN layer to capture multi-hop dependencies in the graph, combining information from different levels of structural context. The final output of the graph layers is passed through an MLP to compute a single scalar prediction for user–item pairs.

3.2 Graph Construction

To encode user–user or item–item relationships, we construct graph structures from the normalized rating matrix. Specifically, we compute the user-user precision matrix by inverting a regularized covariance matrix, as described in the next Section. This matrix is interpreted as the adjacency matrix for the GNN, capturing conditional independence between entities.

By using the precision matrix instead of a raw similarity graph, the model should learn from more informative structural patterns, which could improve generalization over sparse or noisy data.

3.3 Model Input

The model receives as input the full user–item rating matrix, normalized per user as described in Section 4. For each prediction, a leave-one-out strategy is employed where the input vector for the target movie masks the rating of a specific user. The model then outputs a full vector of predictions for that movie, and the masked rating is compared to its true value using the chosen loss function. This loss can then be used to train the model.

To train the model efficiently, the leave-one-out strategy is combined with mini-batching. Rather than processing a single masked movie vector at a time, each batch contains multiple such vectors, typically from different movies. Do note that the same movie may appear with different masked user ratings as well. The model will simultaneously output predictions for all vectors in the batch. The losses for the masked entries are then aggregated, and the total loss is used to update the model weights through backpropagation.

This input strategy was chosen because it allows the model to learn from all entries in the user-item matrix. By masking only a single target rating per input vector, the model learns from the complete data context while focusing on one prediction at a time. However, the approach is computationally demanding because each forward pass predicts an entire movie vector, yet only one entry contributes to the loss. In contrast, alternative strategies that mask multiple entries at once (e.g., 10% of the vector) can speed up training by leveraging more known predictions per pass, but may introduce noisier gradients and reduce the precision of learning.

During evaluation on the test data, the entire user–item rating matrix is fed into the model, and a prediction is made for all user-movie pairs. These predictions are compared with the held-out test ratings using the chosen evaluation metric. This approach ensures that no ground-truth information from the test pair leaks into the model input during inference.

3.4 Implementation Details and Scalability

All models and pipelines are implemented in PyTorch, using GPU acceleration via the CUDA backend for scalable training and inference. Currently, all experiments were conducted on the MovieLens 100k dataset; however, the implementation has been designed with scalability in mind and generalizes to larger datasets, such as MovieLens 1M or 10M. Larger datasets will increase training time and resource requirements, but the use of mini-batching, parallel data loading, and hardware acceleration will help keep training times acceptable.

4 Experimental Setup

4.1 Evaluation Metrics

As described in the introduction, the primary focus of this research is on prediction accuracy. For this extent, there are multiple options, but the most common ones are RMSE and MAE:

- **Root Mean Squared Error (RMSE):** RMSE penalizes larger errors more heavily and is computed as:

$$\text{RMSE} = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,m)\in\mathcal{T}} \left(\hat{R}_{u,m} - R_{u,m}\right)^2}$$

  where $\mathcal{T}$ is the test set, $R_{u,m}$ is the true rating, and $\hat{R}_{u,m}$ is the predicted rating.

- **Mean Absolute Error (MAE):** MAE provides a more interpretable, linear measure of average prediction error:

$$\text{MAE} = \frac{1}{|\mathcal{T}|} \sum_{(u,m)\in\mathcal{T}} \left|\hat{R}_{u,m} - R_{u,m}\right|$$

While Mean Absolute Error (MAE) was initially used during early experimentation due to its ease of interpretation, Root Mean Squared Error (RMSE) is the standard benchmark for movie recommender systems. Therefore, all results in this paper are reported using RMSE.

Predicted ratings for the test set are obtained by feeding the full user–movie matrix, constructed from the training and validation data, into the model. The model then produces predicted ratings for all user–movie pairs. Each test data point is evaluated by comparing its actual rating with the corresponding predicted value using the chosen evaluation metric.

4.2 Data setup

4.2.1 Dataset

This paper focuses explicitly on predicting movie ratings using the MovieLens-100k dataset. The MovieLens data sets were collected by the GroupLens Research Project at the University of Minnesota (Harper and Konstan 2015). This specific dataset consists of 100,000 unique user-movie pairings, each of which has a rating

ranging from 1 to 5. When working with the full dataset, each user has rated at least 20 movies, ensuring that there are enough data points to make predictions for every user. MovieLens also provides simple demographic information for users (age, gender, occupation, and zip code) and genre information for movies, but these are not used by the model in this paper.

The MovieLens-100k dataset is commonly used for benchmarking recommender systems, allowing straightforward comparison with existing methods. Despite its widespread use, the dataset is relatively small and sparse, which limits the generalizability of conclusions drawn from it to larger-scale settings.

4.2.2 Data-preprocessing

Prior to training the VNN, data preprocessing involves the following steps:

1. **Train–Validation–Test Split:** Ratings are randomly partitioned into training (80%), validation (10%), and test (10%) sets, using a fixed random seed for reproducibility. Alternatively, predefined splits provided by MovieLens (80% training, 20% testing) can be used to compare the model against benchmarks, which will be done in Section 5.

2. **Normalization:** User-specific z-score normalization is applied to the training ratings, centering ratings around each user's mean and scaling by their standard deviation. This normalization, which uses the means and standard deviations of the training set, is consistently applied to both the validation and test sets. Following is the mathematical description of these steps:

$$\mu_u = \frac{1}{|\mathcal{M}_u|} \sum_{m \in \mathcal{M}_u} R_{u,m}, \quad \sigma_u = \sqrt{\frac{1}{|\mathcal{M}_u|} \sum_{m \in \mathcal{M}_u} (R_{u,m} - \mu_u)^2 + \epsilon},$$

   where $\mathcal{M}_u$ is the set of movies rated by user u, and $\epsilon = 10^{-4}$ ensures numerical stability.

   The normalized user-item matrix is then constructed as follows:

$$Z_{u,m} = \frac{R_{u,m} - \mu_u}{\sigma_u}$$

3. **Precision Matrix Construction:** A user–user precision matrix is computed from the normalized rating matrix, capturing conditional dependencies

between users. An analogous item–item precision matrix can similarly be constructed. The procedure can be described as follows:

Given the normalized rating matrix $Z \in R^{n_u \times n_m}$ and mask $M \in \{0,1\}^{n_u \times n_m}$, the user-user precision matrix is computed as follows.

First, we calculate the covariance matrix between users:

$$\text{Cov} = \frac{ZZ^\top}{C-1},$$

where $C = \max(MM^\top, 2)$ represents the co-rating counts between user pairs, ensuring numerical stability.

Then, the precision matrix is obtained by adding a small regularization term to the covariance matrix and inverting it:

$$P = (\text{Cov} + \epsilon I)^{-1}, \quad \epsilon = 10^{-4}.$$

4. **Creating model input:** We construct the model input by selecting, for each data point, the corresponding row from the movie-user rating matrix and temporarily masking out the rating of the user-item pair under evaluation. The model then tries to predict this masked rating, and the loss will be computed based solely on the difference between the predicted and specific masked rating. This approach is commonly referred to as a *leave-one-out* strategy.

5. **Data loading:** PyTorch DataLoaders are used to load and batch the training and validation data efficiently.

This structured approach ensures consistent preprocessing across datasets and prepares the data efficiently for training and evaluating the GNN model.

4.3 Hyperparameters

4.3.1 Description

The training of the model includes six key hyperparameters: batch size, learning rate, loss function, `dimNodeSignals`, `nFilterTaps`, and `dimLayersMLP`. The first three are standard across most deep learning models and are defined as follows:

- **Batch size**: Specifies the number of samples used to compute each gradient update. Larger batch sizes improve training speed but may reduce accuracy.

- **Learning rate**: Controls the step size in the gradient descent optimization process. A smaller value ensures more stable convergence, while a larger value may speed up training but risk overshooting minima.
- **Loss function**: Defines the objective the model minimizes. The most common loss functions are the Mean Squared Error (MSE), the Root Mean Squared Error (RMSE), and the Mean Absolute Error (MAE).

The remaining three hyperparameters are arrays of numbers, which specify the architecture of the Graph Neural Network and the MLP layers at the end:

- `dimNodeSignals`: Specifies the dimensionality of the node feature vectors at each GNN layer. For a given layer l, if `dimNodeSignals`[l] $= d^{(l)}$, then each node is represented by a $d^{(l)}$-dimensional embedding at that layer. The first layer is fixed at 1 to match the input features, but subsequent layers can be freely configured. Larger values of $d^{(l)}$ enable the model to learn more expressive node representations, at the cost of increased parameter count and computational load. Typical values range from 16 to 128, though the choice can be adapted based on model capacity and dataset size.
- `nFilterTaps`: Indicates the number of "filter taps" in each GNN layer. The value of `nFilterTaps`, specified by K, determines the receptive field of the layer, allowing it to aggregate information from up to $(K-1)$-hop neighbors. For instance, $K = 2$ incorporates only immediate neighbors, while $K = 4$ allows the layer to aggregate from nodes up to three hops away. Typical values range from 2 to 5, providing a balance between local structural information and computational efficiency. Although a separate value can be defined for each layer, it is common to use the same number of taps across all layers for consistency.
- `dimLayersMLP`: Defines the architecture of the multilayer perceptron (MLP) applied after the final GNN layer. Each value in `dimLayersMLP` specifies the number of hidden neurons in a fully connected layer, with the last value always set to 1 to produce a scalar rating prediction. The first layer of the MLP must match the output dimensionality of the final GNN layer, i.e., it must be equal to the last value in `dimNodeSignals`. Deeper or wider MLPs can model more complex, nonlinear interactions between node embeddings, but also increase the number of parameters and the risk of overfitting. Standard configurations include one or two hidden layers with sizes ranging from 32 to 128 units. The structure can be adjusted based on model complexity and the availability of data.

4.3.2 Hyperparameter tuning

To determine the optimal values for the six primary hyperparameters, a grid search was performed using `ParameterGrid` from the `scikit-learn` library. The search was conducted over manually selected value ranges based on prior experiments and architectural intuition. Each configuration was evaluated using validation RMSE under the leave-one-out evaluation strategy described earlier.

The following configuration yielded the best balance between performance and runtime; because of this, these values are used to calculate the final results:

- Learning rate: 0.002
- Batch size: 256
- Loss function: Mean Squared Error (MSE)
- `dimNodeSignals`: [1, 32, 64]
- `nFilterTaps`: [4, 4]
- `dimLayersMLP`: [64, 32, 16, 1]

5 Results and Discussion

5.1 Rating Prediction Performance

To evaluate the effectiveness of our proposed method, we report the RMSE on the MovieLens-100k dataset using the standard 80/20 train–test split. Table 1 compares our model to a selection of benchmark methods, including classical matrix completion and state-of-the-art graph-based recommender systems.

Model	**RMSE**	**Ref**
MC (2009)	0.973	(Candes and Recht 2008)
GMC (2014)	0.996	(Kalofolias et al. 2014)
GC-MC (2017)	0.905	(Berg, Kipf, and Welling 2017)
GHRS (2022)	0.887	(Zamanzadeh Darban and Valipour 2022)
MoRGH (2022)	0.881	(Ziaee, Rahmani, and Nazari 2024)
VNN (this work)	**0.9496**	—

Table 1: Benchmark results on MovieLens-100k (lower RMSE is better)

While our model does not outperform the most recent graph-based methods in terms of RMSE, it demonstrates strong performance relative to the basic benchmark methods, such as MC and GMC. Notably, our approach uses only

inductive prediction, which means it does not rely on side information or extensive pre-training, unlike the two state-of-the-art models (GHRS and MoRGH). This highlights the potential of leveraging precision matrix structures in graph construction for collaborative filtering tasks.

The prediction performance of 0.95 RMSE on the test data is not state-of-the-art; however, it demonstrates that the model is capable of learning meaningful patterns and producing reasonable rating estimates. For comparison, a baseline that always predicts the global average rating yields an RMSE of approximately 1.2, while using the user-specific mean improves this slightly to around 1.1.

5.2 Prediction Deviation Analysis

To better understand the model's behavior, we analyze how predicted ratings deviate from ground-truth values across the 20,000 test examples. These differences were calculated by first rounding the prediction to the nearest integer and then comparing it to the expected value. The majority of predictions (46.5%) are exactly correct, and over 88% only deviate at most 1 rating point:

- Exact match ($\Delta = 0$): 46.5%
- Off by 1: 41.8%
- Off by 2: 9.7%
- Off by 3 or more: 2.0%

A more detailed breakdown shown in a heatmap 1, shows systematic biases, particularly for low ratings. For example, when the actual rating is 1, the model often predicts 2 or 3, suggesting difficulty in recognizing strong dislike. Similarly, ratings that should be 5 are frequently underpredicted as 4. These patterns are common in recommender systems and result from models overfitting towards the mean of the dataset.

Overall, the results indicate that while the model is capable of predicting user ratings, it tends to overpredict, particularly struggling with lower ratings such as 1 and 2, which are rarely predicted accurately. The heatmap 1 reveals that the model does detect lower-than-average ratings to some extent, but often defaults to predicting a 3 rather than the dataset's average of 4. This suggests that the model is detecting that the rating is likely lower, but can't produce the extremity. One possible mitigation could involve post-processing the output, rather than simply reversing the normalization step; one could apply a scaling transformation

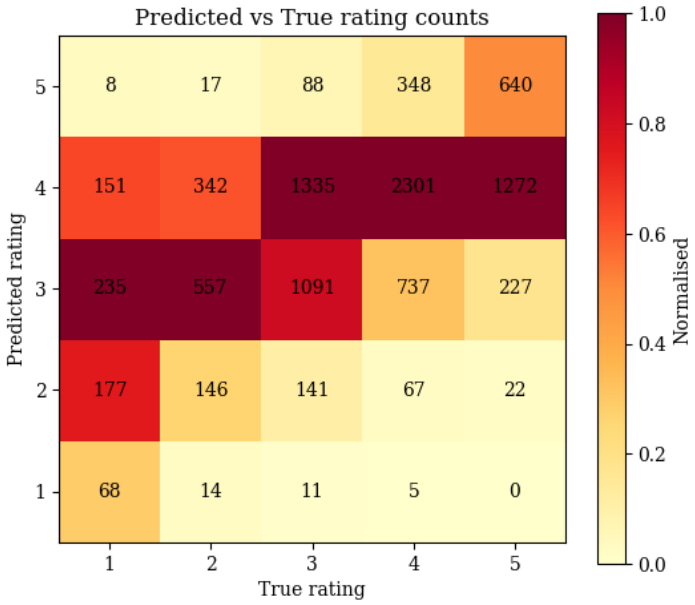

Figure 1: Heatmap of true and predicted ratings

that increases the likelihood of predicting more extreme values. Although such adjustments were outside the scope of this project, they represent a promising direction for improving prediction accuracy.

6 Conclusion and Future Work

6.1 Conclusion

This research explored the use of a Graph Neural Network (GNN) model leveraging precision matrices as graph filters for the rating prediction task on the MovieLens-100k dataset. Building upon the framework introduced by Sihag et al. (2022), the model was adapted to work with the user-user precision matrix derived from the user–item rating data. The model's training employed a leave-one-out strategy to maximize the learning.

Through extensive manual experimentation and hyperparameter tuning, the proposed approach achieved a test RMSE of 0.9496, confirming that the model is indeed capable of predicting movie ratings; however, it falls short of the state-of-the-art. While the model shows strong performance, it tends to overpredict lower ratings, rarely assigning scores of 1 or 2. The results suggest that the model captures meaningful signals from the precision-based graph structure, but may benefit from further techniques to better model rating extremes.

6.2 Future Work

The model shows potential, and there are a few opportunities for further research:

Improved output calibration: Future work could explore post-processing techniques to calibrate the output distribution further and enhance the prediction of extreme values. This may include nonlinear scaling or loss reweighting strategies.

Using larger datasets: Applying the model to larger datasets, such as MovieLens 1M, would help assess scalability and generalization. This would enable more robust comparisons with recent deep learning–based recommender systems.

Sparsification Techniques: Future work could look into sparsifying the precision matrix to reduce computational overhead while preserving its structural information. Techniques such as soft/hard thresholding or stochastic sparsification may enable more scalable training without compromising performance.

Broader Evaluation Metrics: Beyond RMSE, future studies could incorporate other heuristics, such as diversity, novelty, and top-N recommendation metrics, to assess the quality of predictions more comprehensively.

7 Responsible Research

7.1 Data privacy

This research utilizes the publicly available MovieLens 100k dataset, which contains anonymous user–movie ratings. The dataset is released and maintained by GroupLens for academic and non-commercial use, with all personally identifiable information removed. As such, there are no concerns related to data privacy or protection in this work.

7.2 Reproducibility

To ensure the reproducibility of results, all sources of randomness in the data pipeline and model training process—such as weight initialization, data shuffling, and batching—are explicitly seeded. By setting the random seed to `42`, all experiments can be replicated exactly, yielding consistent results across runs. The final results were all archived using the hyperparameters stated in the experimental setup.

The specific implementation pipeline used in this work is available at: https://github.com/VicVsl/VNN-recsys.

References

Berg, Rianne van den, Thomas N. Kipf, and Max Welling (2017). "Graph Convolutional Matrix Completion". In: eprint: 1706.02263. URL: https://arxiv.org/abs/1706.02263.

Candes, Emmanuel J. and Benjamin Recht (2008). "Exact Matrix Completion via Convex Optimization". In: URL: https://arxiv.org/abs/0805.4471.

Castells, Pablo, Neil Hurley, and Saúl Vargas (2021). "Novelty and Diversity in Recommender Systems". In: pp. 603–646.

Cavallo, Andrea, Zhan Gao, and Elvin Isufi (2024). "Sparse Covariance Neural Networks". In: *arXiv preprint arXiv:2410.01669*. URL: https://arxiv.org/abs/2410.01669.

Harper, F. Maxwell and Joseph A. Konstan (Dec. 2015). "The MovieLens Datasets: History and Context". In: *ACM Transactions on Interactive Intelligent Systems (TiiS)* 5.4, 19:1–19:19. DOI: 10.1145/2827872. URL: http://dx.doi.org/10.1145/2827872.

Jadon, Aryan and Avinash Patil (2024). "A Comprehensive Survey of Evaluation Techniques for Recommendation Systems". In: *arXiv preprint arXiv:2312.16015*. URL: https://arxiv.org/abs/2312.16015.

Kalofolias, Vassilis et al. (2014). "Matrix Completion on Graphs". In.

Koren, Yehuda, Robert Bell, and Chris Volinsky (2009). "Matrix Factorization Techniques for Recommender Systems". In: *Computer* 42.8, pp. 30–37. DOI: 10.1109/MC.2009.263.

Sihag, Saurabh et al. (2022). "Covariance Neural Networks". In: *Advances in Neural Information Processing Systems* 35, pp. 17003–17016. URL: https://github.com/sihags/VNN.

Vargas, Saúl and Pablo Castells (2011). "Rank and Relevance in Novelty and Diversity Metrics for Recommender Systems". In: pp. 109–116. DOI: 10.1145/2043932.2043955. URL: https://doi.org/10.1145/2043932.2043955.

Wu, Zonghan et al. (2021). "A Comprehensive Survey on Graph Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1, pp. 4–24. DOI: 10.1109/TNNLS.2020.2978386.

Zamanzadeh Darban, Zahra and Mohammad Hadi Valipour (2022). "GHRS: Graph-based hybrid recommendation system with application to movie recommendation". In: *Expert Systems with Applications*. URL: http://dx.doi.org/10.1016/j.eswa.2022.116850.

Ziaee, Seyed, Hossein Rahmani, and Mohammad Nazari (2024). "MoRGH: Movie Recommender System using GNNs on Heterogeneous Graphs". In: DOI: 10.21203/rs.3.rs-3860094/v1.

Appendix: Use of AI Tools

During the development of this research project, various AI-assisted tools were used to support the author. Their use was limited to supportive tasks and did not replace any core scientific contributions. Specifically:

- **GitHub Copilot** was used for technical coding assistance, particularly when working with PyTorch and SkLearn.
- **ChatGPT** was consulted to explore architectural and design decisions for the pipeline and to debug the codebase. Additionally, ChatGPT was prompted to develop ideas for structuring the paper and improve the clarity of written content by rephrasing poorly worded paragraphs.
- **Grammarly** was used to correct spelling, grammar, and to improve sentence flow in a few sections.

All results, analyses, and conclusions presented in the paper were determined and verified independently by the author.