Structure of the MSc Project Report

Chapter 1. Introduction

Section 1.1. Project Aim

The aim of the project is to show modern standard industrialization-level short video recommendations systems processes, and pick up one core process to research some classical or relatively novel algorithms 's performance in the given datasets. The main goal of the project is to measure and evaluate these algorithms via some classic measurements.

Section 1.2. Objectives

Conduct a literature review on existing short-video recommendations systems, including basic methodologies, basic processes, common measures, classic algorithms, and main issues.

Implement some basic recommendations model which are based on different theory and used in different process of complete recommendations systems.

Find appropriate datasets containing many user interactions data, and preprocesses them so that these data can be used by my following implementation.

Evaluate the performance of the algorithms implemented with HR@N, NDCG.

Optimize and fine-tune these models.

Section 1.3. Deliverables

1. A research report detailing the background, methodology, and findings of them.
2. Source code and documentation for the implemented system.
3. Performance evaluation report comparing different recommendation techniques.
4. A report analysing studying outcomes, potential issues.

Section 1.4 Ethical, legal, and social issues

Chapter 2. Background Research

Section 2.1. Literature Survey (or another appropriate title)

2.1.1 Introduction

Video recommendation is a highly valuable research area. Nowadays, people spend a significant portion of their leisure time on various video platforms. For platforms like YouTube and TikTok, the core issue lies in how to retain users, enhance their experience and stickiness, and make them willing to spend more time on the platform. This is precisely why video recommendation systems are so crucial.

Video recommendation falls under the broader category of recommender systems. The commonalities among different recommender systems far outweigh their differences. Whether it's e-commerce, articles, news, or videos, all recommendation systems heavily rely on a set of shared fundamental algorithms. Therefore, although this project focuses on video recommendation, most of the involved content applies generally to all types of recommender systems, including basic algorithms, research methods, evaluation metrics, common challenges, and more.

Next, we will thoroughly review the history of recommender systems, classical algorithms, commonly used evaluation metrics, key challenges, and how industry handles recommendation workflows and optimization strategies.

2.1.2 Development History

Modern recommender systems began to take shape in the late 20th century. In 1992, Goldberg et al. proposed the idea of Collaborative Filtering to address the problem of email recommendations. Before this, recommendations were purely based on content information (Gifford et al.). Collaborative Filtering introduced the use of interaction history between users and items as a basis for making further recommendations, greatly improving recommendation accuracy.

In 1994, Resnick et al. implemented the collaborative filtering system in GroupLens, becoming one of the first teams to deploy this approach in practice. To this day, despite the emergence of numerous new algorithms, many of them are still grounded in user-item similarity — the core idea of Collaborative Filtering. This is why the advent of Collaborative Filtering is often regarded as the beginning of recommender systems as an independent research field.

In 1998, Amazon introduced item-based collaborative filtering, distinguishing it from the previously dominant user-based collaborative filtering. Around 2009, Koren et al. introduced matrix factorization to recommender systems. Latent vectors began to represent users and items, where highly relevant or strongly preferred user-item pairs were trained to have similar embeddings, resulting in higher dot products or cosine similarity scores.

In 2013, Xiaodong He et al. introduced the DSSM (Deep Structured Semantic Model), bringing deep learning into the recommendation domain. Deep learning enabled the system to learn more complex, non-linear, and abstract user-item relationships. It also made it possible to incorporate multimodal information, such as user profiles and item attributes — particularly relevant in video recommendation, where clicks, likes, video titles, thumbnails, and content can all serve as inputs.

In 2015, Hidasi, Balázs et al. applied Recurrent Neural Networks (RNNs) to model session-based interactions, enabling the system to consider not only historical interactions but also their sequential order, thereby placing more emphasis on recent user behavior.

In 2018, Kang et al. introduced Transformers to recommender systems, resulting in the SASRec model. In recent years, the field has advanced toward multimodal and large-model architectures. In the multimodal domain, Yinwei Wei et al. released MMGCN in 2019, which structurally integrated multimodal information into the recommendation framework. Earlier models typically handled multimodal inputs by simple concatenation, with limited effectiveness. MMGCN, however, built specific graph structures and training mechanisms tailored to multimodal data.

In terms of large models, the focus has shifted to using generative large language models to enhance various stages of traditional recommendation systems rather than as standalone recommenders. GPT4Rec is a notable example of this approach.

To date, we have reviewed the retrieval phase of recommender systems — the process of filtering out a set of potentially interesting items from a large pool of candidates.

However, in real-world industrial applications, a recommender system must not only identify which items a user might be interested in but, more importantly, rank these items to determine which ones to actually display to the user The ranking phase involves entirely different considerations from the retrieval phase and reflects a much higher degree of personalization. This means the development path varies significantly across recommendation domains. In the video recommendation field, for example, Tianxin Wei et al. introduced HoME (Hierarchical Multi-task Learning) in 2021 to address the need for optimizing multiple objectives simultaneously — such as completion rate, like rate, and share rate — in order to meet the diverse needs of users.

2.1.3 The Workflow of Industrial Recommender Systems

Qi Liu et al. (2024) provide a detailed explanation of how the industry currently structures the workflow of recommendation systems, which can be divided into four main stages: **retrieval**, **pre-ranking**, **ranking**, and **re-ranking**. In the retrieval stage—also known as the recall stage—the system aims to fetch hundreds or thousands of candidate items that might match the user's preferences from the entire database. The pre-ranking stage then filters out low-quality content before passing the remaining items to the ranking stage. The ranking stage further selects the highest-quality content that the user is most likely to engage with. Finally, the re-ranking stage determines the exact set of content shown to the user, considering aspects such as diversity and business objectives.

In simple terms, the entire workflow can be broadly divided into two phases: **retrieval** and **ranking**. The retrieval phase focuses on collecting potentially interesting videos for the user from large-scale data, while the ranking phase emphasizes content quality, diversity, and commercialization.

Notably, Shusen Wang (2023) points out that in industrial applications, metrics such as **daily active users (DAU)**, **monthly active users (MAU)**, and **retention** are often considered more important than traditional metrics like CTR (Click-Through Rate) and NDCG (Normalized Discounted Cumulative Gain). These user engagement metrics are more directly linked to platform profitability. It is conceivable that even with highly accurate recommendations and high hit rates, if the content is overly homogeneous and stylistically repetitive, users may experience fatigue and eventually churn. While current academic research primarily focuses on improving recommendation accuracy, there is still a lack of algorithms and strategies that directly target improvements in DAU/MAU.

Furthermore, in real-world production environments, the retrieval phase rarely relies on a single algorithm. Some platforms even employ more than **20 different algorithms** and over **50 retrieval channels** simultaneously. This suggests that, despite differences in hit rates among algorithms, the selection in production is not based solely on the highest-performing algorithm in terms of recommendation accuracy, but rather on deeper considerations like DAU and MAU. Interestingly, in some cases, engineers intentionally introduce **randomness** into otherwise accurate recommendation models such as two-tower architectures to enhance **diversity**, which can paradoxically lead to **better commercial metrics**.

Section 2.2. Methods and Techniques (or another appropriate title)

Provide an introduction to the evaluation metrics, describe both the implemented algorithms and key algorithms in detail, explain the data processing methods, and outline the evaluation process. Discuss the necessity of negative sampling and the different sampling strategies. Elaborate on and explain the core steps of a video recommendation system.

2.2.1 classic retrieval algorithms

2.2.1.1 Collaborative Filtering

To study recommendation systems, one must first understand the basic concept of **collaborative filtering**, as it has significantly influenced the majority of subsequent recommendation algorithms. Broadly speaking, collaborative filtering can be divided into **user-based** and **item-based** collaborative filtering. Here, we briefly explain **item-based collaborative filtering** as an example, since the underlying principles are largely similar in the user-based approach.

Item-based collaborative filtering is based on an intuitive idea: users tend to be interested in content similar to what they have interacted with before. If a large number of users have interacted with both item A and item B, we can infer a higher similarity between these two items.

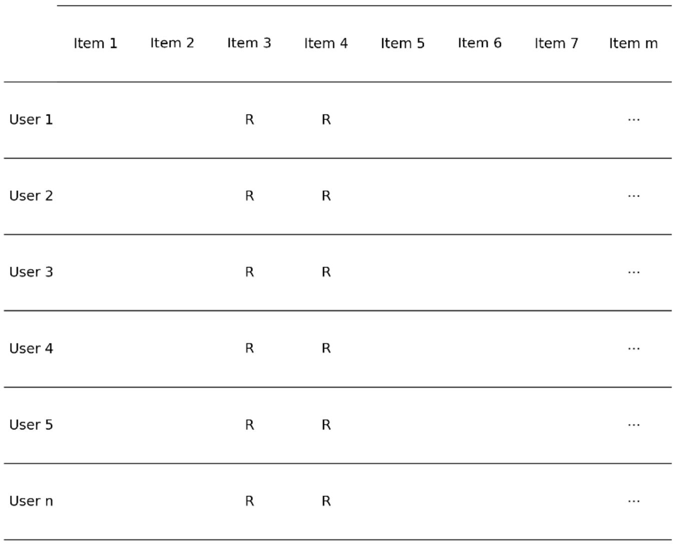
For instance, suppose we have a dataset containing users$\{u_1, u_2, u_3 \dots u_n\}$, and items$\{i_1, i_2 \dots i_m\}$, To estimate how much a user i might like a new item j, we can make the following assumption: the user has previously interacted with N items, and their preference for any item n can be represented by a score $R_{i,n}$. In the context of video recommendation, such scores can be derived from behaviors like watching, liking, or sharing. To evaluate user i's potential interest in item j(which the user has not interacted with before), we can calculate the similarity between item j and each of the N previously interacted items. A **weighted sum** of these similarities and score $R_{i,n}$, can then be used to estimate how much the user may like item j. This estimate provides a strong basis for making recommendations. Regarding how to compute similarity, several commonly used methods have been proposed, such as those outlined by Badrul Sarwar et al. (2001).

2.2.1.1.1 Cosine Similarity

We represent items as vectors, — for example, item *i* is denoted as $\vec{i}$ and item *j* as $\vec{j}$. Then, the similarity between the two items can be expressed as:

$$\mathrm{sim}(i,j) = \cos(\vec{i},\vec{j}) = \frac{\vec{i}\cdot\vec{j}}{|\vec{i}|_2 \times |\vec{j}|_2}$$

After the large-scale adoption of machine learning, item and user vector representations can be learned directly through training. Prior to this, a common method for vectorization was as follows, as shown in Figure 1:



	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6	Item 7	Item m
User 1			R	R				···
User 2			R	R				···
User 3			R	R				···
User 4			R	R				···
User 5			R	R				···
User n			R	R				···

figure 1: User-item interaction matrix: there are n users and m items, and each R represents the rating that a given user has given to a specific item.

It can be seen that for any two items, such as item 3 and item 4, the ratings given by all users in the database naturally form the vector representations of these items. Similarly, for any user, their ratings across all items naturally constitute that user's vector representation.

However, this form of representation has a major drawback: in a database with tens of millions of users and items, the corresponding vectors must also be in the tens of millions of dimensions. This poses a significant burden on both storage and computational resources. Moreover, most users have no interactions with most items, meaning that the interaction matrix is inherently **sparse**, leading to substantial inefficiency and resource waste.

2.2.1.1.2 Pearson correlation-based Similarity

We can also consider using the **Pearson correlation coefficient** to express the relationship between two items. As shown in *Figure 1*, each user-item pair has an associated value RRR, representing the user's rating for that item. Here, $\overline{R}_i$ and $\overline{R}_j$ represent the average ratings of items *i* and *j* across all users, respectively.

From a vectorization perspective, the Pearson correlation formula is quite similar to **cosine similarity**, but with a key modification: instead of using the raw item vectors, each item vector is adjusted by subtracting its mean rating. In other words, the vector representing item *i* is transformed by subtracting $\overline{R}_i$ from each of its components, and the same adjustment is applied to item *j*.

$$\mathrm{sim}(i,j) = \mathrm{corr}_{\mathrm{i,j}} = \frac{\sum_{u \in U}(R_{u,i} - \overline{R}_i)(R_{u,j} - \overline{R}_j)}{\sqrt{\sum_{u \in U}(R_{u,i} - \overline{R}_i)^2} \cdot \sqrt{\sum_{u \in U}(R_{u,j} - \overline{R}_j)^2}}$$

2.2.1.1.3 Adjusted Cosine Similarity

In the previous similarity calculations, one easily overlooked issue is that different users may have varying rating habits — some tend to rate more generously, while others are more conservative. For example, a certain user might often give scores of 100, and a score of 99 already indicates dissatisfaction. In contrast, another user might usually give scores around 50, and a score of 60 would reflect satisfaction.

To address this discrepancy, an improved method for calculating similarity has been proposed:

$$\mathrm{sim}(i,j) = \frac{\sum_{u \in U}(R_{u,i} - \overline{R}_u)(R_{u,j} - \overline{R}_u)}{\sqrt{\sum_{u \in U}(R_{u,i} - \overline{R}_u)^2} \cdot \sqrt{\sum_{u \in U}(R_{u,j} - \overline{R}_u)^2}}$$

Here, $R_{u,i}$ represents the rating given by user *u* to item *i*, and $\bar{R}_u$ denotes the average rating that user *u* has given across all items.

2.2.1.1.4 Dot Product Similarity

Similar to cosine similarity, but without normalization. This approach has both clear advantages and disadvantages.

The **advantage** is that the similarity measure takes into account not only the **direction** (structure) of the vectors but also their **magnitude** (intensity), meaning both the angle and the length of the vectors are considered.

The **disadvantage**, however, is that it becomes difficult to determine whether the similarity between two items is due to strong signals in just a few dimensions, or because the overall structure of the vectors is well aligned.

$$\mathrm{sim}(i,j) = \cos(\vec{i},\vec{j}) = \vec{\mathrm{i}} \cdot \vec{\mathrm{j}}$$

2.2.1.1.5 prediction

$$P_{u,i} = \frac{\sum_{\text{all similar items } N}\left(s_{i,N} \cdot R_{u,N}\right)}{\sum_{\text{all similar items } N}\left(\left|s_{i,N}\right|\right)}$$

After successfully obtaining the similarity scores, the second step is to calculate the final **preference score** of user *u* for item *j*. This is typically done using a simple **weighted average**, which is one of the most classic prediction methods in collaborative filtering.

By applying this operation to all items in the database, and then sorting them based on the predicted preference scores, the top-ranked items are considered to be those the user is most likely to be interested in.

2.2.1.2 Matrix Factorization (MF)

Koren, Y. et al. (2009) provided a comprehensive summary of various matrix factorization variants applied in the field of recommendation. Here, we offer a brief introduction to the core ideas.

In traditional collaborative filtering, user vectors are typically represented by that user's ratings across all items, while item vectors are represented by all users' ratings for that item. This approach leads to the well-known **sparse matrix problem**: in a real-world database, any given user interacts with only a small subset of items, and each item is typically interacted with by only a limited number of users. As a result, we are forced to maintain an extremely large but mostly empty matrix — highly inefficient in practice.

To address this issue, researchers began exploring the use of **latent factor models**, mapping both users and items into a shared low-dimensional vector space. This allows users and items to be embedded in the same space, where their relationships can be modeled mathematically.

Accordingly, a user *u* is represented by a vector $\boldsymbol{p_u}$, $\boldsymbol{p_u} \in R^f$, and an item *i* is represented by a vector$\boldsymbol{q}_i$, $\boldsymbol{q}_i \in R^f$ where f is the dimension of the latent space. The **dot product** of these two vectors can then be used to estimate the predicted preference score $\hat{\mathrm{r}}_{\mathrm{u,i}}$ — that is, how much user u is expected to like item i.

$$\hat{\mathrm{r}}_{\mathrm{u,i}} = q_i^{\top} p_u$$

Our goal is to continuously optimize $\boldsymbol{p_u}$ and $\boldsymbol{q}_i$ such that the dot product between the latent vectors of highly related users and items becomes larger. A classic **mean squared error (MSE)** loss function is defined as follows:

$$\min_{q,p} \sum_{(u,i)\in\mathcal{K}} \left(r_{ui} - q_i^\top p_u\right)^2 + \lambda(|q_i|^2 + |p_u|^2)$$

Here, r_{ui} is the actual rating that user *u* gave to item *i*, and the term $\lambda(|\boldsymbol{q_i}|^2 + |\boldsymbol{p_u}|^2)$ is a **regularization term** that penalizes vectors with large magnitudes. This helps constrain the vector norms within a stable range and prevents overfitting due to extreme values.

A natural question arises: **why not use cosine similarity instead of the dot product**, which would seemingly eliminate the need for regularization?

There are two main reasons for this:

1. **Cosine similarity restricts the expressiveness of vector magnitudes**, forcing the optimization to focus only on the angle between vectors. This limits the flexibility of the model and makes convergence more difficult, as it reduces the search space for optimization.

2. **Cosine similarity is a non-linear function**, which in some cases can lead to problems such as gradient explosion and unstable optimization paths, introducing unnecessary challenges to model training.

Additionally, because some users tend to consistently give higher ratings, and certain items are generally well-received and rated higher by most users, there are often **individual biases** associated with both users and items. As a result, modeling the predicted rating $\hat{r}_{u,i}$ using only the dot product $\boldsymbol{q_i^\top p_u}$ can be insufficient.

In practice, this issue is addressed using **bias-aware formulations**, such as the following:

$$\hat{\mathrm{r}}_{\mathrm{u,i}} = \mu + b_i + b_u + q_i^T \boldsymbol{p_u}$$

Here, μ represents the **global average rating**, b_i is the **item bias** (adjustment for item *i*), and b_u is the **user bias** (adjustment for user *u*).

Although in theory the original vectors could account for such biases by adjusting their magnitudes, this approach tends to perform poorly in practice. Without bias terms, the model has to capture not only the user's preferences and the item's features, but also individual tendencies — such as a user's general tendency to rate higher or an item's overall popularity. This added burden weakens learning effectiveness.

By introducing **bias terms**, each component can focus on its specific role:

- b_i captures the general rating tendency of the item,
- $b_u$reflects the rating habits of the user,
- p_u focuses purely on the user's preferences,
- q_i captures the item's latent features.

This separation of concerns makes the model both **easier to train** and **more interpretable**. It's like giving a person hands when they originally only had arms — instead of forcing the arms to do both coordination and fine manipulation, hands take over the detailed work while arms handle direction and support. This division of responsibility simplifies learning and improves performance.

2.2.1.3 DSSM(Deep Structured Semantic Model) and Two-Tower Model

2.2.1.4 LightGCN (Light Graph Convolutional Network)

2.2.1.5 GRU4Rec (Gated Recurrent Unit for Recommendation)

2.2.1.6 SASRec (Self-Attentive Sequential Recommendation)

2.2.1.7 NextItemNet: A Neural Network Architecture for Next-Item Recommendation

2.2.1.8 MMGCN (Multi-Modal Graph Convolutional Network)

2.2.3 Ranking algorithms using HoME as an example

2.2.4 Loss Function

2.2.4.1 Bayesian Personalized Ranking (BPR) loss

2.2.4.2 In-batch Softmax Loss

2.2.5 Nearest Neighbor Search (NNS) Methods

2.2.6 Negative Sampling Technique

2.2.7 Evaluation Metrics

Section 2.3. Choice of methods (or another appropriate title)

I chose to implement five representative retrieval algorithms and experimented with using In-batch Softmax Loss as the loss function during training. Additionally, I applied FAISS as the tool for nearest neighbor search. Below, I detail the reasoning behind these choices:

2.3.1 Algorithm Selection

I selected five widely used and representative retrieval algorithms, which are commonly adopted by various datasets as baselines. In fact, the dataset I used already includes these five algorithms as official baselines, which makes my selection comparably valid and allows me to evaluate the quality of my training in a meaningful way.

2.3.2 Why In-batch Softmax Loss

Initially, I attempted to use BPR (Bayesian Personalized Ranking) loss as the training objective. This loss requires, for every input, at least one negative sample, encouraging the model to predict scores closer to the positive item and farther from the negative one. While this method is theoretically sound and has been widely adopted in research papers, I encountered several practical issues:

Negative Sampling Strategy. Choosing appropriate negative samples proved challenging. Should they be selected randomly? Should a correction factor be applied to balance out popular items? (Without correction, random sampling tends to favor popular items simply due to their higher overall occurrence.)

I experimented with various strategies — sampling one negative item, sampling ten, applying popularity correction, and using purely random sampling — but found two major problems:

Firstly, The impact of sampling strategy on performance was non-linear. Using ten negatives did not significantly outperform using one. Surprisingly, applying corrected sampling led to results very close to a baseline that simply recommends the most popular items, suggesting the model began to favor globally popular content regardless of user profile.

Secondly, Negative sampling was a CPU-bound, compute-intensive task. It could not be efficiently accelerated on GPU. Each iteration required scanning the dataset and randomly selecting negative items that do not appear in the user's interaction history. To avoid overfitting, it's crucial to sample different negatives per epoch, demanding high CPU usage.

For large-scale datasets, this becomes a bottleneck: training with 1 million positive samples requires generating 1 million negatives per epoch, totaling 10 million sampling operations over 10 epochs. Because each negative must be validated against user history, it's difficult to batch or pre-generate these samples, making this process a major slowdown in training.

Instead, Switching to In-batch Softmax Loss led to significant improvements in both training speed and accuracy. It eliminated the need for explicit negative sampling, making the training pipeline much more efficient.

2.3.3 Use of Nearest Neighbor Search (FAISS)

Although my current dataset is not large enough to require nearest neighbor search, I still implemented it using FAISS to ensure the model can scale to industrial-level applications. In real-world scenarios, the system must match users with the most relevant videos in real time, which makes fast vector search essential. FAISS enables efficient indexing and retrieval of embedding vectors, preparing the system for deployment at scale.

2.3.4 Choice of Similarity Function

I chose to use dot product as the similarity measure instead of cosine similarity, which some papers prefer. Although dot product has a theoretical risk of overfitting due to large vector norms, I observed that when combined with In-batch Softmax Loss, this risk was minimal (further discussed in the results section).
Moreover, dot product led to faster convergence during training compared to cosine similarity, which further justified its selection in my implementation.

Chapter 3 Datasets and Experimental Design

Section 3.1 Datasets/Data Sources

The dataset I selected is Microlens, released by Westlake University. It includes user–video interaction records, video cover images, and video title information. The dataset contains 352,649 interaction records, 50,000 users, and 19,220 videos. Its moderate size makes it suitable for quickly validating model effectiveness in the early stages. In addition, it includes some item-side information, such as images and titles, which facilitates the use of multimodal information.

(Planned, not yet used): The Tenrec dataset, released by Tencent, contains 38,300,254 interaction records and 1,000,000 users. Compared to Microlens, it offers nearly 100 times more data, enabling evaluation of model performance on large-scale datasets. Furthermore, Tenrec provides rich user-side information, which opens up possibilities for addressing the cold-start problem. Its scale also introduces new demands for data processing, making it a suitable benchmark for evaluating performance optimization strategies on large datasets.
(Planned, but uncertain, depending on available time and resources.)

Section 3.2. Experimental Design

Experimental Environment: Google Colab

Code Management Tool: Git

Code Repository: GitHub

Experiment Preparation:

1. Set random seeds to ensure reproducibility.
2. Preprocess data by sorting interaction records based on user ID and timestamp.
3. Filter out users and videos with too few interactions.
4. Split the dataset: use each user's last interaction as the test set; the remaining interactions form the training set.
5. Filter the test set: remove any items that appear only in the test set but not in the training set.
6. Maintain new indexing: rebuild continuous user and item ID sequences, preserving the mapping. Original IDs may be non-sequential and unsuitable for modeling.
7. Process multimodal data: use OpenAI's CLIP model to convert each video's cover image into a 512-dimensional vector, then reduce it to 128 dimensions using PCA, and store the result in an LMDB database (a file-based key-value vector database).

Model Construction: Implement the five selected algorithms: LightGCN, DSSM, NextItNet, GRU4Rec, SASRec.

Similarity Calculation: Use dot product uniformly across models.

Loss Function Design: Adopt In-Batch Cross Entropy Loss.

Evaluation Process:

1. Baseline Setup: Use two baseline methods — Random recommendation and Popularity-based recommendation.
2. Metrics:
    - Hit Ratio@K (HR@K): Measures whether the ground truth test item appears in the Top-K recommendations.
    - Normalized Discounted Cumulative Gain@K (NDCG@K): Takes into account the ranking position of the correct item — higher ranks receive higher scores.
3. Evaluation Steps:
    - Store the user and item embeddings generated after training.
    - Store item vectors using FAISS for fast nearest neighbor search.
    - For each user in the test set, retrieve their corresponding user embedding and find the Top-K most similar item vectors using FAISS.
    - Treat these K items as predicted recommendations.

- Check if the ground truth item is among the Top-K and compute HR@K and NDCG@K accordingly.

Multimodal Testing:

Based on the above experiments, test the effect of adding multimodal information to the model. Specifically, concatenate the previously processed 128-dimensional video image vector into the model at an appropriate stage. After unified processing, use an MLP layer at a suitable point to convert the dimensionality of the combined vector, ensuring the output remains consistent with the original dimension.

Continue to use the same training and evaluation pipeline to assess the effect of multimodal integration.

(***Tentative)*** If time permits, migrate the optimized pipeline to the Tenrec dataset to evaluate its performance at scale. Additionally, use the rich user metadata in Tenrec to test how multimodal features can help solve the cold-start problem.

Chapter 4 Results of the Empirical Investigation

Present the results, analysis, and discussion.

Chapter 5 waiting to decide

Chapter 6 Conclusions and Future Work

References

Goldberg, D., Nichols, D., Oki, B.M. and Terry, D. 1992. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*. **35**(12), pp.61–70.

Gifford, D.K., Baldwin, R.W., Berlin, S.T. and Lucassen, J.M. An architec- ture for large scale information sys- tems. In Proceedings Tenth Symposium on Operating Systems Principles (Orcas Island, Wash., Dec. 1985), pp. 161- 170.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P. and Riedl, J. 1994. GroupLens: an open architecture for collaborative filtering of netnews *In*: *CSCW `94: Computer Supported Cooperative Work Proceedings Held in Chapel Hill, N. C. November 22-26, 1994*. New York, NY, USA: ACM, pp.175–186.

Linden, G., Smith, B. and York, J. 2003. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE internet computing*. **7**(1), pp.76–80.

Koren et al., 2009, *Matrix Factorization Techniques for Recommender Systems*

Huang, P.-S., He, X., Gao, J., Deng, L., Acero, A. and Heck, L. 2013. Learning deep structured semantic models for web search using clickthrough data *In*: *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. New York, NY, USA: ACM, pp.2333–2338.

Hidasi, B., Karatzoglou, A., Baltrunas, L. and Tikk, D. 2015. Session-based Recommendations with Recurrent Neural Networks.

Kang, W.-C. and McAuley, J. 2018. Self-Attentive Sequential Recommendation.

Sarwar, B., Karypis, G., Konstan, J. and Riedl, J. 2001. Item-based collaborative filtering recommendation algorithms *In*: *Proceedings of the 10th international conference on World Wide Web*. New York, NY, USA: ACM, pp.285–295.

Koren, Y., Bell, R. and Volinsky, C. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer (Long Beach, Calif.)*. **42**(8), pp.30–37.

Liu, Q., Zheng, K., Huang, R., Li, W., Cai, K., Chai, Y., Niu, Y., Hui, Y., Han, B., Mou, N., Wang, H., Bao, W., Yu, Y., Zhou, G., Li, H., Song, Y., Lian, D. and Gai, K. 2024. RecFlow: An Industrial Full Flow Recommendation Dataset.

Wang, S. 2023. Methodologies for Improving Modern Industrial Recommender Systems.