Wharton
University of Pennsylvania

# Decentralized Recommendation System

Matthew Kuo, Laura Li, Vivian Xiao, Megan Yang

April 22, 2025

# Problem Definition

# Problem Statement

Current recommendation systems are platform-specific and disconnected, forcing users to manually search for alternatives across different shopping websites.

**Goal:** Develop an online shopping recommendation system that leverages product metadata (e.g., images, descriptions, etc.) to provide seamless, personalized suggestions across platforms, reducing the need for repetitive searches.

# Technical Approach

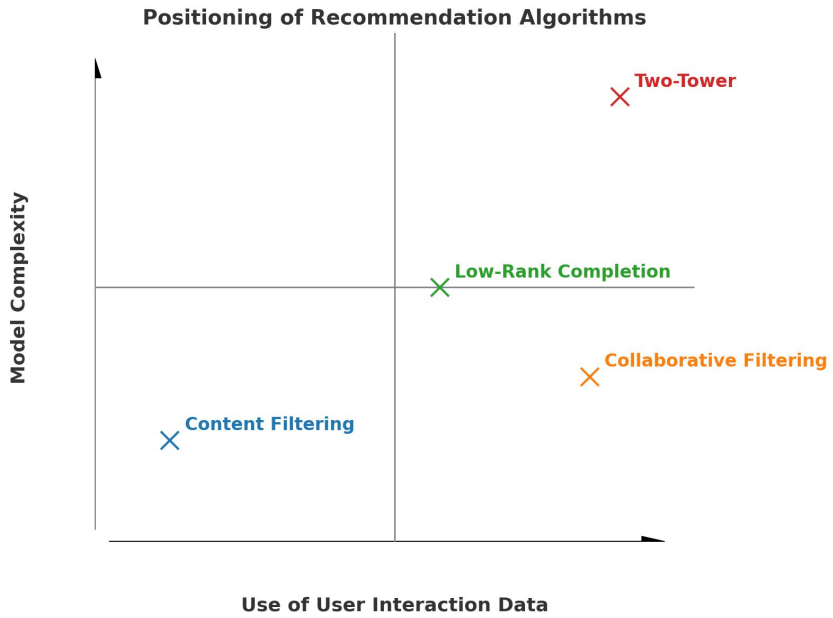
# Dataset

- **~3k images** of clothes
- Used mistral 7B to generate ratings for different personas

{"name": "Alex", "bio": "A 28-year-old graphic designer favoring Scandinavian minimalist styles. Prefers monochrome palettes (black, white, grey), high-quality natural fabrics (linen, wool), clean geometric cuts, and avoids logos or excessive detailing."}

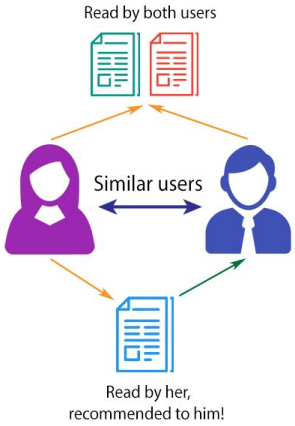
# Overall Approach

- Find algorithms that can cover recommending similar items (**consistency**) in addition to new items that the user might like (**novelty**)
- Think of low-rank matrix completion as a foundation—many modern models, including Two-Tower and CF
- Content filtering is purely based on item metadata, whereas CF and Two-Tower learn from other users.
- By using a mix of these approaches, we ensure both cold-start coverage and user-specific novelty.



# Collaborative Filtering

- **User-based:** finding similar users and suggesting what they like
- **Item-based:** finding similar items based on user interactions




## Optimization Function

$$\min_{U,V} \sum_{(i,j)\in\Omega} (R_{ij} - U_i^T V_j)^2 + \lambda(||U||^2 + ||V||^2)$$

where:

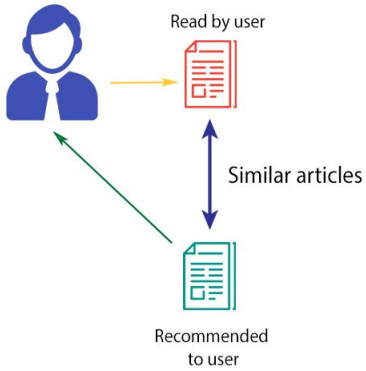
- $R_{ij}$: the observed rating for user $i$ and item $j$
- $U_i$ and $V_j$: user and item vectors
- $\lambda$: regularization parameter
- $\Omega$: set of user-item interactions

## Implementation Steps

1. Extract user & item vectors from dataset.
2. Compute cosine similarity between the target user and others.
3. Select top-N similar users/items as weights.
4. Predict item scores using weighted preferences.
5. Rank & recommend top items based on scores.

# Content-based Filtering

- Analyzes **item features** (e.g., descriptions, image embeddings) and compares them to a user's **past preferences**.
- A user preference vector ($v_u$) is created by averaging the feature representations of liked items ($L_u$).
- New items ($j$) are ranked based on **cosine similarity** ($s_j$) to the user preference vector, with the highest-scoring items recommended.




## Mathematical Formulation

$$v_u = \frac{1}{|L_u|} \sum_{i \in L_u} x_i$$

$$\arg\max_{j \notin L_u} s_j \qquad s_j = \frac{v_u \cdot x_j}{\|v_u\| \|x_j\|}$$

## Implementation Steps

1. Represent each item as a feature vector (text & image embeddings)
2. Compile a list of all "liked" items for a user
3. Calculate the cosine similarity between the preference vector and all other items.
4. Sort items by similarity.
5. Return top recommendations

# Low-rank Matrix Completion

The goal is to complete a partially observed user-item rating matrix using a low-rank factorization approach.

Given a rating matrix $R \in \mathbb{R}^{m \times n}$, where $m$ is the number of items and $n$ is the number of users, the objective is to find two low-rank matrices $U \in \mathbb{R}^{m \times k}$ and $V \in \mathbb{R}^{n \times k}$ such that:

$$\hat{R} = UV^T$$

where $k$ is the latent factor dimension. The optimization problem is formulated as:

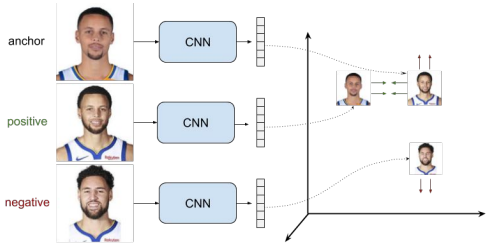
$$\min_{U,V} \sum_{(i,j) \in \Omega} (R_{ij} - (UV^T)_{ij})^2 + \lambda(||U||_F^2 + ||V||_F^2)$$

where:

- $\Omega$ is the set of observed entries in $R$
- $\lambda$ is the regularization term to prevent overfitting
- $||\cdot||_F$ denotes the Frobenius norm

## Implementation Steps:

1. Generate ratings matrix and create a mask for known ratings
2. Grid Search to find the optimal rank with minimal validation loss
3. Initialize low-rank matrices U and V using a projection layer
4. Use Adam optimizer for training
5. Compute final predictions for missing values, and recommend items with the highest predicted ratings for each user.




## Model Tuning:

- GridSearch for optimal rank, tuning for lr, lambda_reg
- Instead of pure matrix factorization, train model with a fixed projection layer on U using item embeddings
- Pairwise ranking loss (Bayesian Personalized Ranking (BPR) loss) to capture the relative ranking positions of instances (flat rmse across rank, but improved precision)

# Two-Tower

Instead of learning one large joint representation of users and items

- Use one NN (tower) to learn user reps and the other to learn item reps.
- Compare them with a similarity function.

<u>Implementation Steps & Mathematical Formulation</u>

1. Use CLIP to provide initial embeddings.
2. Obtained input for the user tower:
   a. selection vector of length of the number of items.
3. User embeddings are passed through user tower for transformation:
   a. $\tilde{Z}_u = f_U(Z_u; \theta_U)$
4. Item embeddings are fed into item tower:
   b. $\tilde{Z}_i = f_I(Z_i; \theta_I)$

Loss function (MSE) ensures that user and item embeddings of items user likes become more similar.

$$S(Z_u, Z_i) = \frac{\tilde{Z}_u \cdot \tilde{Z}_i}{\|\tilde{Z}_u\| \|\tilde{Z}_i\|}$$

$$\mathcal{L} = \frac{1}{N} \sum_{(u,i)} (S(\tilde{Z}_u, \tilde{Z}_i) - y_{ui})^2$$

# Demo!

# Results

# Define metrics

Rmse

Mae

Precision@10

Recall@10:

# Model Metrics – Collaborative filtering

## Original Ratings

RMSE: 3.2663

MAE: 2.6129

Precision: 0.0167

Recall: 0.0042

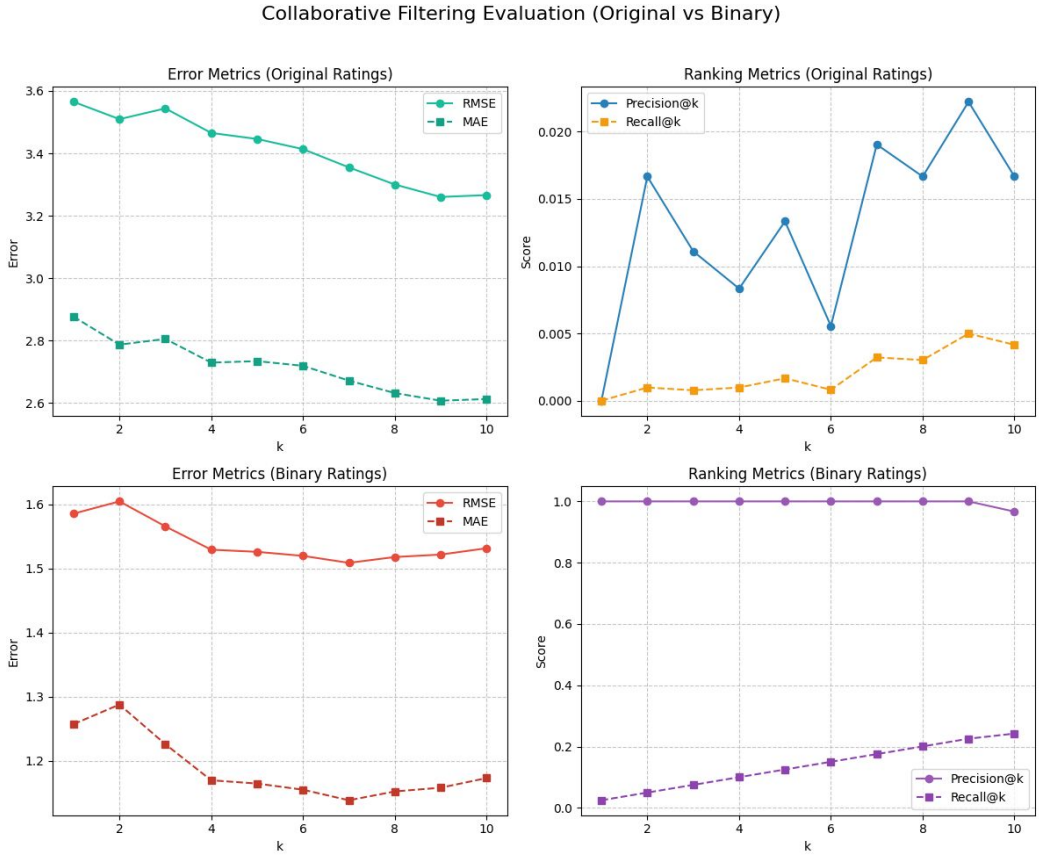
## Binary Ratings

RMSE: 1.5316

MAE: 1.1729

Precision: 0.9667

Recall: 0.2426



# Model Metrics – Content Based Filtering

## Original Ratings

RMSE: 3.1535

MAE: 2.5099

Precision: 0.0367

Recall: 0.0093

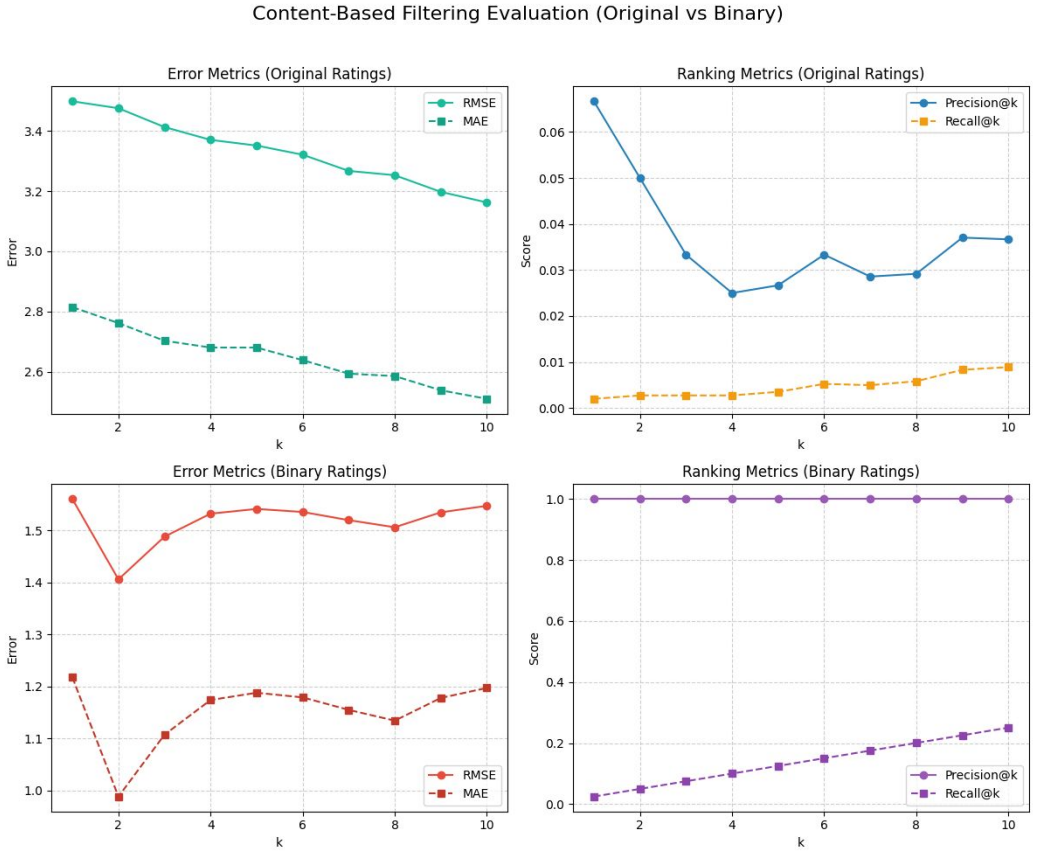
## Binary Ratings

RMSE: 1.5474

MAE: 1.1972

Precision: 1.0000

Recall: 0.2510



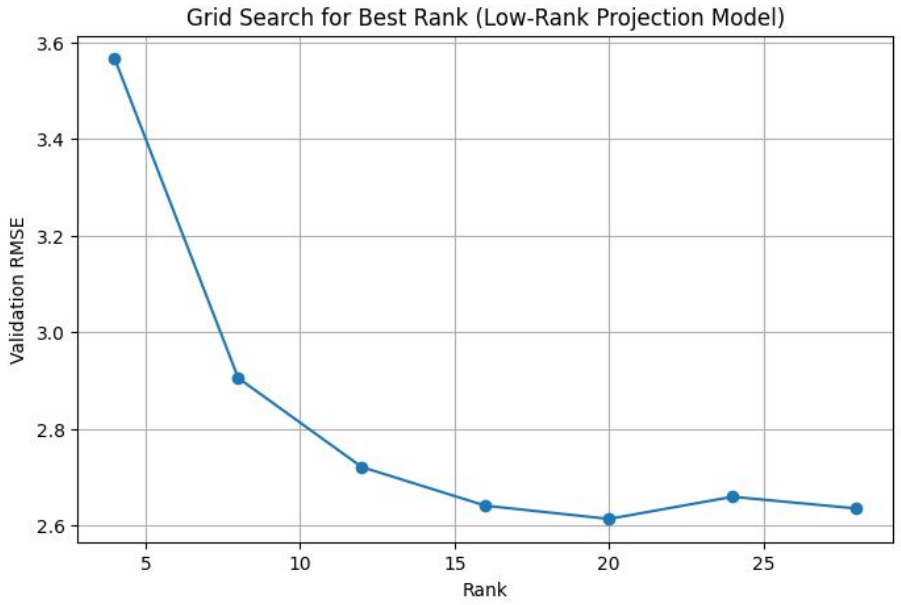
# Model Metrics – Low Rank Projection Model

## Original Ratings (with rank=20)

| | |
|---|---|
| RMSE: | 2.6131 |
| MAE: | 2.1946 |
| Precision@10: | 0.0133 |
| Recall@10: | 0.0030 |

## Binary Ratings (with rank=16)

| | |
|---|---|
| RMSE: | 0.9809 |
| MAE: | 0.9639 |
| Precision@10: | 0.0133 |
| Recall@10: | 0.0030 |



# Model Metrics – Low Rank with Pairwise Ranking Loss

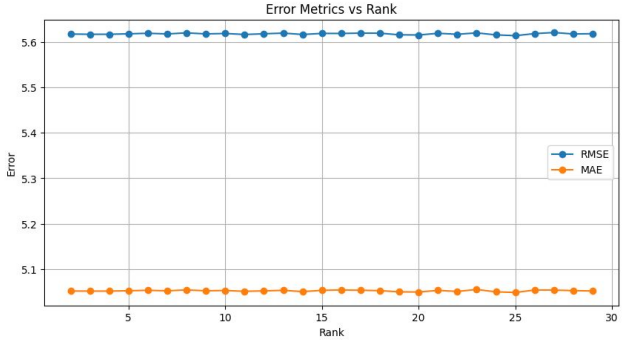
## Original Ratings (with rank=11)

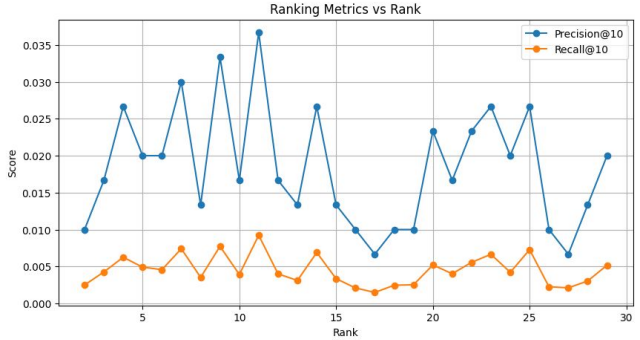
RMSE: 5.6161

MAE: 5.0515

Precision: 0.0367

Recall: 0.0092



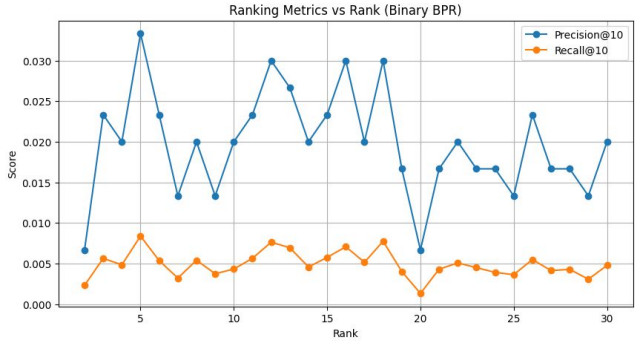

## Binary Ratings (with rank=5)

RMSE: 0.9998

MAE: 0.9989

Precision: 0.0333

Recall: 0.0084

# Model Metrics – Two Tower

## Original Ratings

RMSE: 2.8050

MAE: 2.2545

Precision@10: 0.0333

Recall@10: 0.0076

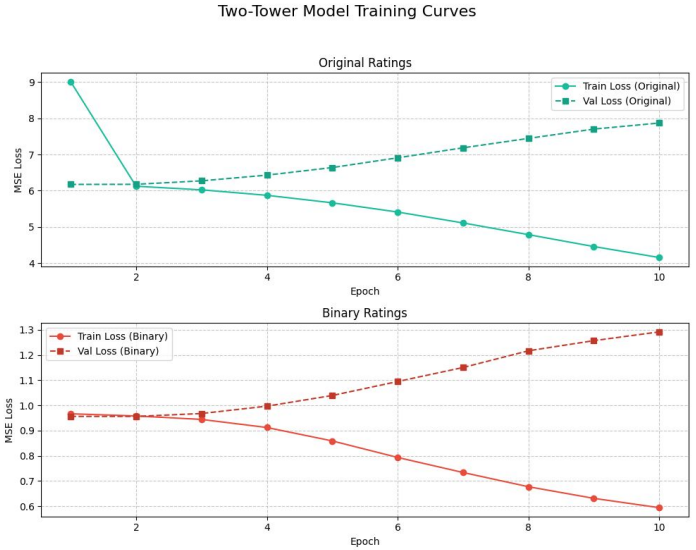
## Binary Ratings

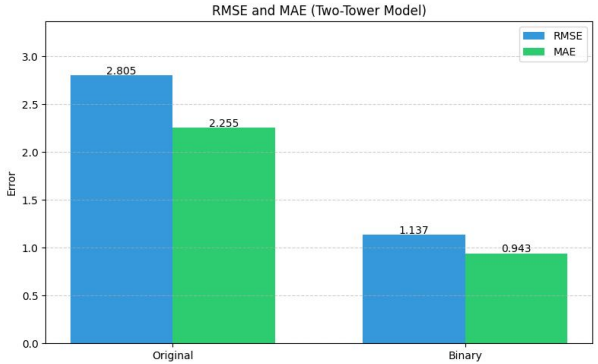
RMSE: 1.1365

MAE: 0.9426

Precision: 0.0000

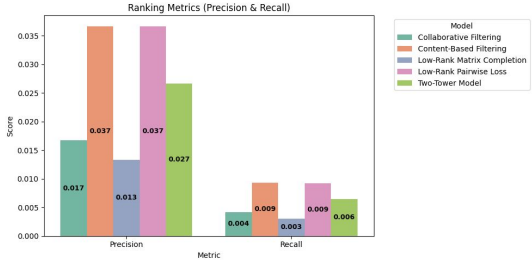
Recall: 0.0000



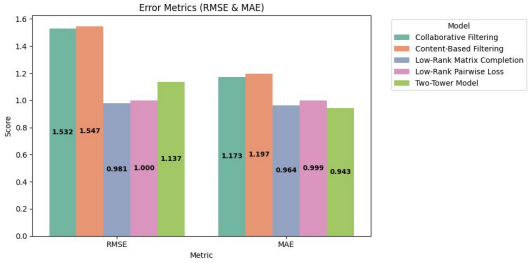



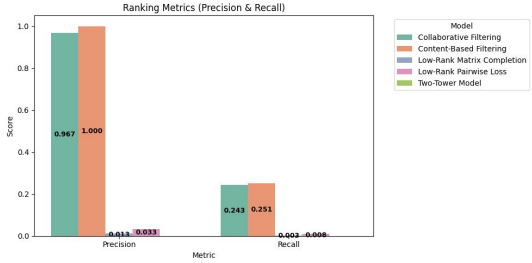
# Performance Summary

- Collaborative Filtering & Content-Based Filtering demonstrate similar performance
- Similar RMSE, CBF has better precision
- Both achieve 1.0 Precision@10 on binary ratings matrix


- Low Rank with fixed projection U: small error but poor precision
- Low Rank with BPR: high RMSE, good precision but doesn't improve on binary




- Two Tower:
- Good performance on original ratings matrix (lowest RMSE and decent precision)
- Didn't observe improvement in precision on binary

# Model Comparison & Tradeoffs

| Model | Uses Metadata | Uses Interaction Matrix | Handles Cold-Start | Recommendation Type |
|---|---|---|---|---|
| Content Filtering | ✅ | ❌ | ✅ | Uniform but personalized |
| Collaborative Filtering | ❌ | ✅ | ❌ | Novel, social-based |
| Low-Rank Completion | Optional | ✅ | ❌ (unless initialized with metadata) | Interpolative |
| Two-Tower | ✅ | ✅ | ✅ | Hybrid, rich representations |

# Reflection

# Hardest Technical/Conceptual Difficulty

We had a hard time understanding what data to feed into the two tower model.

- Initially only used the image embeddings for the item tower, then realized that including text embeddings (even though CLIP limits it to 77 tokens) provided rich information like material that might be hard to see through the image itself; thus concatenated the two embeddings.
- Realized averaging item embeddings for a “user profile” didn’t give the model any flexibility, so we switched to a +1/–1 selection vector of length 2,519 (one entry per item), letting the network learn its own mapping into the shared latent space.

# Workflow

Easier:

- Content-based filtering. Once we had our combined text+image embeddings loaded, we just computed pairwise similarities which was very fast and required minimal debugging while also giving great results.

Harder:

- Collecting data: Planned to scrape product images and recruit volunteers to “like” or “dislike” them. In reality, e-commerce sites blocked the scraping, and labeling was too manual.

# Evolving Goals

- Initial goal: Simply implement all four recommenders (content, collaborative, low-rank, two-tower) and compare their outputs.
- Mid-project pivot:
  - Simulated extra user data (personas) to see how algorithms behaved as ratings grew.
    - Appends a new column of +1/-1 or NaNs so that algorithms can use the new user.
  - Automated our data-cleaning pipeline so that any change to the raw CSVs flowed through all models without manual edits.
    - use ast.literal_eval to parse text- and image-embedding strings into real Python lists
    - skip any rows that throw parsing errors
  - Added caching for the best low-rank projection so that inference is quick in our API.
    - We train the best linear projection weights (over ranks 4, 8, 12…28), we store them in memory. Any subsequent calls to low_rank_recommend reuse the same trained projection instead of retraining every API request.

# AI Tools Assist

- Persona & ratings generation: We had GPT craft realistic user profiles and then generate simulated ±1 feedback code, which jump-started our sparse ratings matrix.
- Code writing & debugging: AI templates for PyTorch training loops, FastAPI endpoints, and data pipelines gave us a solid foundation—and its real-time tips on tensor shapes and dtype conversions unblocked our two-tower forward pass.

# Individual Contributions

# Megan

Surprising result: One of the best results were content filtering even though it was so simple.

Specific lecture: Lecture 9 helped us choose Adam over Adagrad because Adam retains Adagrad's per-parameter adaptive scaling—automatically dampening parameters with large gradients. Thus, our two-tower network reached useful recommendation quality in fewer epochs

Perspective on optimization: Thought optimization was simple and theoretical. In practice, however, nonconvex problems behave unpredictably and some practices are more practical although less optimal (for example step-size, we should be diminishing but choose a constant step-size and manually decrease it).

Two more weeks: Set up a hyperparameter-optimization pipeline (e.g., using Optuna) to explore learning rates, layer sizes, and regularization strengths for the two-tower

Restart the project: Prioritize data collection infrastructure first—designing a user-friendly labeling interface and recruitment plan—before implementing multiple algorithms.