AmazonRecommender: An End-to-End Video Game Recommendation System Built on the Amazon Reviews 2023 Dataset

Jah Chen
University of Washington
Seattle, WA
jah0311@uw.edu

Aidan Bartlett
University of Washington
Seattle, WA
aidanb@uw.edu

Asad Jaffery
University of Washington
Seattle, WA
asadjaf@uw.edu

Allen Zheng
University of Washington
Seattle, WA
azheng15@uw.edu

George Lee
University of Washington
Seattle, WA
bob888w@uw.edu

Abstract

We present *AmazonRecommender*, an end-to-end recommendation system for Amazon video game products built on top of the Amazon Reviews 2023 dataset. Our system ingests raw review and metadata files, normalizes them into a relational SQLite database with more than 4.6M product-level interactions, and exposes a Flask-based web interface for querying games and exploring recommendations. Our goal is to provide a reproducible educational infrastructure that connects large-scale real-world data to practical recommendation pipelines. We describe data processing, schema design, and system architecture, and outline the baseline recommendation strategies that can be implemented on top of this framework. Finally, we discuss the limitations of our current implementation and directions for integrating stronger representation-learning methods pretrained on the same dataset.

1 Introduction

Recommendation systems are a key component of modern digital platforms, including e-commerce, streaming, and social media. Building such systems on realistic data is challenging for students and practitioners because the available datasets are often either synthetic, heavily preprocessed, or too large to handle without substantial infrastructure investment.

The Amazon Reviews 2023 dataset (1) provides a large-scale, modern benchmark for recommendation and retrieval research. It contains more than 570M reviews and 48M items across 33 categories, with rich item metadata, fine-grained timestamps, and user–item interaction graphs. However, the raw data are distributed as

JSONL files that still require substantial engineering effort before they can be used in a production environment.

In this work, we describe *AmazonRecommender*, a project that focuses on the building an recommendation system that successfully bridges the gap between raw Amazon data and a client facing demo:

- We build a pipeline that transforms the raw Amazon Reviews 2023 video game files into a normalized SQLite database with separate tables for products and reviews.

Preprint. Under review.

- We design a simple yet flexible system architecture that connects this database to a Flask web application for querying, browsing, and recommending games.
- We implement a baseline recommendation system based on a memory-model collective filtering framework that can be extended with learned text embeddings or pretrained models.

The contributions of this project are primarily educational and engineering-focused: we demonstrate how to go from raw industrial-scale data to a functioning recommender, with clear separation between the data layer, the recommendation logic, and the user-facing interface.

2 Related Work

Amazon review datasets. Amazon review datasets have been widely used in recommendation research for over a decade. Early versions (**?**) provided tens of millions of reviews with item metadata and have been extended multiple times since. The 2023 version (**?**) significantly scales up the number of interactions and improves metadata quality, while supporting new tasks such as complex product search and language-driven recommendation.

Recommendation system baselines. Classical collaborative filtering methods, such as matrix factorization and k-nearest-neighbor (kNN) models, remain strong baselines for many recommendation tasks. Popularity-based ranking, simple heuristics over average rating, and category-aware filtering are also widely used as interpretable baselines and production fallbacks. Recent work increasingly leverages pretrained language and multi modal models to construct better item representations and to support text- and image-driven retrieval (**?**).

Educational and systems-focused projects. There is a large body of work on building recommender systems for instructional purposes, ranging from small MovieLens-based demos to full-stack projects. Our work falls into the latter category: we focus on making the Amazon Reviews 2023 dataset usable in an project setting and on providing a modular infrastructure that can host multiple recommendation approaches.

3 Data and Preprocessing

3.1 Dataset

We use the Amazon Reviews 2023 dataset (**?**), restricted to the video game domain. Concretely, we download the following files from the project site:

- `Video_Games.jsonl`: user reviews with ratings, timestamps, and review text.
- `meta_Video_Games.jsonl`: item metadata including titles, categories, pricing, and additional descriptive information.

The processed subset used in our project contains approximately 137,269 unique products and 4,624,615 reviews, as measured by queries on the final database.

3.2 Extraction pipeline

Data extraction and cleaning occurs within the `scripts` directory with an `cleaning.py` script and SQL files for data import and table creation. The SQLite database was used for initial data exploration and loading. Our final product simply used local CSV files (copies of the tables in the database). Our approach was divided into three core stages:

1. **Parsing raw JSON.** The `cleaning.py` script reads the raw JSON files, extracts a subset of fields from both reviews and metadata (e.g., ASIN, title, category, rating, and timestamp), and writes them into intermediate structured files
2. **Schema creation.** The `create_tables.sql` script defines a relational schema used in initial exploratory analysis with the following tables:

- `amazon_products`: one row per product, keyed by parent ASIN, containing metadata such as title, category and publisher.
- `product_reviews`: one row per user–item interaction, containing a unique user identifier, rating, any textual review and timestamp.

3. **Data import.** The `import_tables.sql` script loads the processed data into SQLite, performing any necessary casting, indexing, and deduplication.

3.3 Data cleaning and filtering

In practice, several additional cleaning decisions conducted using Python's Pandas library were necessary to create reliable signals for recommendation:

- **Missing or malformed fields.** Many rows in `product_reviews` contained missing values. A major issue was that pricing data was absent for over 54% of items. These values were imputed using the median price within each product's main category. Further analysis of different methods of handling this missing data would be expected in future iterations of this project.
- **Sparsity constraints.** Items with very few reviews and users with very few interactions were filtered out to reduce noise and improve the stability of collaborative signals. Formally, any item with reviews count $r_i < 5$ or any user with interaction count $u_j < 2$ was removed from the dataset.

In the current implementation, we prioritize a simple and fast pipeline over aggressive cleaning, but are aware of the limitations of our decisions. Given the scope of the project, we felt like device-specific replicated data met our needs, but if we were to attempt to scale up, or perform more complex pre-processing or model development, we would opt to make better use of rigid SQLite ETL scripts or use a cloud hosted database software. Further refinement would also examine how median-based price imputation might bias downstream recommendation signals, and provide more advanced solutions for imputation.

3.4 Data Sample Generation

To facilitate both cold-start experiments and representative testing, stratified samples of the full product dataset were generated using Python and Pandas. Several sampling strategies were applied:

- **Category-based sampling.** The five most common main categories were identified using value counts. From each category, a small, random subset of products was selected to ensure that each major category was represented.
- **Price-bin sampling.** Products were binned by price into ranges: \$0–50, \$50–100, \$100–200, \$200–500, \$500–1000, and \$1000+. From each bin, a fixed number of items were randomly selected to produce a stratified sample across the price spectrum.
- **Most-rated products.** The ten products with the highest number of reviews were included to ensure that highly rated, frequently reviewed items were present.
- **Random sampling.** A small set of ten products was randomly selected from the full dataset to provide additional diversity and unpredictable coverage.

These individual sample sets were then concatenated and deduplicated to form the final `all_samples` dataset. This dataset was used as input for the front-end recommendation interface and further model evaluation.

Visualizations such as histograms of product prices (on a log scale) and bar charts of category and price-bin distributions were also generated to inspect the representativeness and coverage of the samples.

4 System Design

4.1 Overall Architecture

Figure 1 summarizes the system architecture:

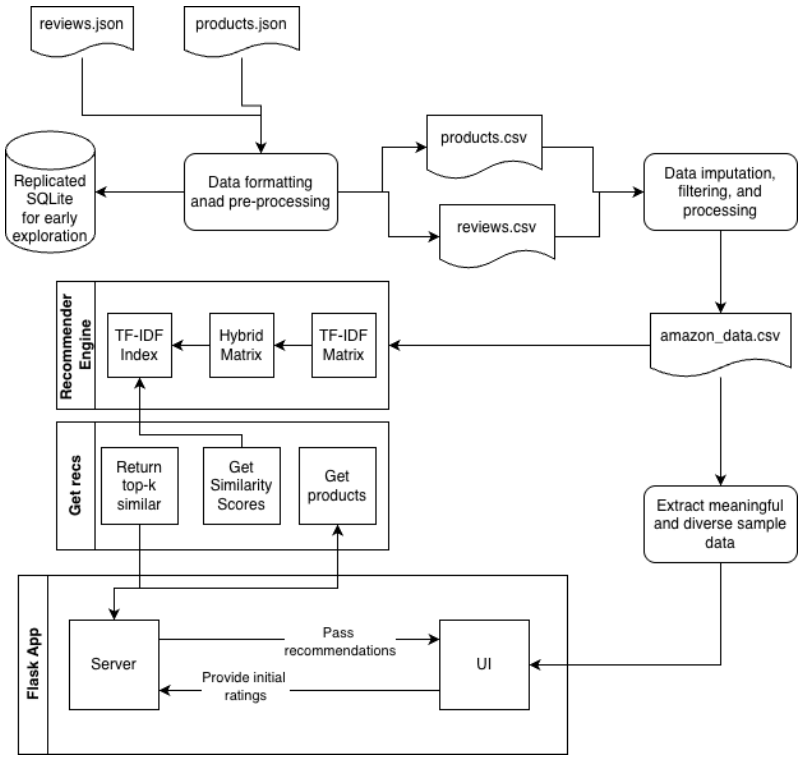

Figure 1: High-level architecture of the AmazonRecommender system. Raw JSON data is transformed into structured CSV files, which back a recommendation API and a Flask-based web UI.

- An **offline data pipeline** that transforms raw JSON into a structured CSV format for ingestion and processing into the data pipeline.
- A **recommendation index layer** that constructs the item–item similarity graph using a hybrid representation of TF-IDF text vectors and normalized numeric metadata.
- A **ranking and scoring layer** that computes similarity-weighted scores based on user ratings, normalizes them, and returns top-N recommendations while excluding previously reviewed items.
- A **data management process** that extracts a random, stratified sample from several categories of the dataset to ensure proper diversity for a user profile.
- A **Flask web application** that serves an interactive HTML and JavaScript webpage, collects user like or dislike signals, displays product recommendations, and allows for user evaluations and data collection.

4.2 Database schema and indexing

The core schema is intentionally simple:

- `amazon_products` has a primary key on the product identifier (e.g., ASIN) and indices on category and price to support browsing and filtering.
- `product_reviews` has indices on both user identifier and product identifier, enabling fast lookup of all reviews for a given product or all interactions for a given user.

This design supports multiple recommendation strategies without changing the underlying storage; new tables (e.g., `product_embeddings`) can be added as needed.

4.3 Flask Application

The Flask application provides a lightweight interface for collecting user signals and delivering model-generated recommendations. Rather than traditional navigation pages, the application exposes a small set of REST endpoints consumed by a JavaScript front end:

- **Sample retrieval endpoint** that serves stratified product samples used for cold-start preference collection.
- **Rating submission endpoints** that record the user's initial like/dislike votes and, later, their evaluation of the recommended items.
- **Recommendation endpoint** that invokes the hybrid item–item engine and returns the ranked product list.
- **Static asset delivery** for HTML, CSS, and JavaScript files.

The client-side script dynamically renders product cards, handles user interactions, and manages the rating logic. Flask's role is intentionally minimal: loading datasets, maintaining in-memory DataFrames for session-level feedback, and routing preference selections to the recommendation engine. This structure keeps the web layer lightweight: as a means to visualize and improve the item representation, similarity computation, and ranking logic within the recommender.

5 Recommendation Methods

Choosing a recommendation model was a balancing act between the memory constraints and runtime complexity of the over 6 GB dataset while also creating a model that could benefit from the rich amount of cross item-user interaction data the dataset provided. Based on these factors a Memory-Based Collaborative Filtering model was chosen.

Creation of Hybrid Matrix To construct a unified feature representation for each item, we combine textual metadata with structured numeric attributes into a single high-dimensional embedding space. Text fields are encoded using a TF–IDF vectorizer configured with a fixed vocabulary size, minimum document frequency threshold, and a unigram–bigram range. Numeric attributes—price, average rating, and rating count—are converted to valid numerical values, imputed where necessary, and standardized to zero mean and unit variance. These normalized numeric vectors are then transformed into a sparse format and concatenated with the TF–IDF matrix, producing a hybrid item matrix that captures both semantic and quantitative properties. A lookup map is preserved to associate each row of the hybrid matrix with its original item identifier.

Item-Item KNN Our ranking function used K-Nearest-Neighbors to define user similarities and provide ranked recommendations. Users provided feedback on their interactions using a binary like or dislike ratio from a random sample of items from the dataset. From there, each item's similarity to other items in the dataset was computed using cosine similarity, and the top-n items were appended to a list weighted by the initial rating from our user. After normalizing similarity score sums, the top-k items were returned in order of perceived relevance to the user. This procedure yields a similarity-weighted estimator given simply as

$$\hat{r}_j = \frac{\sum_{i \in R_u} r_{u,i} \, \mathrm{sim}(i,j)}{\sum_{i \in R_u} |\mathrm{sim}(i,j)|}. \tag{1}$$

It is implemented in our final system as follows:

Algorithm 1 Hybrid Item-Based Recommendation

procedure RETURNRECOMMENDEDITEMS(U, I, H, k, n) ▷ U: user reviews, I: item index, H: hybrid matrix
    Initialize empty maps scores and simSums
    for each review (a, r) in U **do**
        if $a \notin I$ **then**
            continue
        end if
        $p \leftarrow$ index such that $I[p] = a$
        $\boldsymbol{s} \leftarrow \text{cosSim}(H_p, H)$
        $N \leftarrow \text{TopKIndices}(\boldsymbol{s}, k)$
        for each $j \in N$ with $j \neq p$ **do**
            scores$[j] \leftarrow$ scores$[j] + r \cdot s_j$
            simSums$[j] \leftarrow$ simSums$[j] + |s_j|$
        end for
    end for
    for each item j in scores **do**
        $\hat{r}_j \leftarrow$ scores$[j]$/simSums$[j]$
    end for
    $L \leftarrow \text{TopNNotInHistory}(\hat{r}, U, n)$
    return L
end procedure

As seen, the implementation allows for the parameterization of k and n, both to return only the most relevant results to users and to reduce computation on such a large dataset.

6 Experiments

We presented our completed product to a select group of testers in order to evaluate our recommendation model and gain real user insights.

6.1 Experimental setup

User Onboarding and Cold-Start Procedure As our testers were all introduced to the system for the first time during the experiment, we prioritized designing a robust cold-start phase for new users.

Users were provided with a sequence of category choices sampled from the underlying dataset and prompted to rate products negatively or positively (thumbs down or up, respectively). The main webpage was subdivided into sections that reflected common decision factors, including primary product category, price range, highest-selling items, and a randomly selected subset intended to promote diversity in exposure.

6.2 Experimental setup

User Onboarding and Cold-Start Procedure As our testers were all introduced to the system for the first time during the experiment, we prioritized designing a robust cold-start phase for new users.

Users were provided with a sequence of category choices sampled from the underlying dataset and prompted to rate products negatively or positively (thumbs down or up, respectively). The main webpage was subdivided into sections that reflected common decision factors, including primary product category, price range, highest-selling items, and a randomly selected subset intended to promote diversity in exposure.

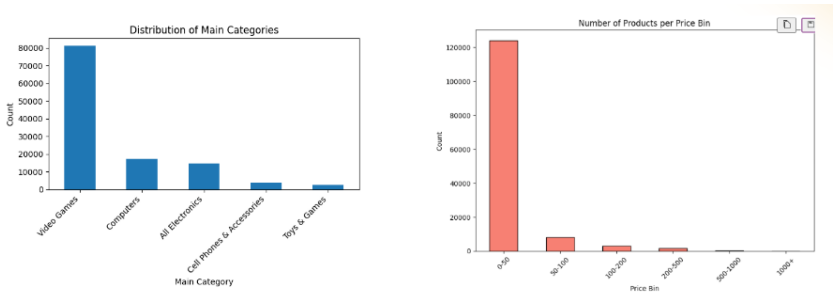

Figure 2: Makeup of the categories users would see in the recommendation sample.

Implementation details All experiments were ran using a replicated CSV of sample products obtained from the category analysis step. For reproducibility, we fixed all relevant random seeds and documented the exact queries (initial ratings in user profile) used to obtain recommendations. This setup also simplified the collection of evaluation metrics, since users were given the ability to provide binary feedback on their final recommended outcomes. These evaluations were stored locally as CSV files for subsequent aggregation and analysis.

7 Evaluation Methodology

Our evaluation process employed a user-centric approach to assess the quality of recommendations generated by our kNN item-based content-filtering system. After users completed the initial cold-start profiling phase by selecting their likes and dislikes from a curated sample of products, the system generated "personalized" recommendations. Users then provided binary feedback (like or dislike) on each recommended product, allowing us to measure subjective relevance and user satisfaction.

We conducted multiple trial runs across different product categories and user personas to ensure diverse coverage of our recommendation space. This methodology hopes to capture real-world user behavior in evaluating our recommendation system.

8 Performance Metrics

8.1 Precision and NDCG by User Scenario

Table 1: Performance metrics across three user test scenarios

User Scenario	Precision	NDCG
User looking for racing games	0.60	0.85
User looking for mouse and headset	0.67	0.72
PC gamer looking for game codes	0.10	0.441

For a subset of participants, we elicited their interests prior to the experiment and instructed them to use the recommendation system primarily to discover items similar to those interests. These users were asked to provide positive feedback on approximately ten items that closely matched their true preferences and negative feedback on roughly three items that deviated substantially from their tastes. Because the system returns items in descending order of predicted similarity, we then asked these users to re-rate the presented items, and we treated these ratings as ground-truth relevance judgments when computing the ideal DCG (IDCG). From there, we calculated the precision and NDCG of our model and present the results in Table 1.

Due to the added complexity of this deeper evaluation protocol, we were only able to apply it to three users, although they represent a diverse set of information-seeking patterns. Moreover, given the nature of our recommendation setting, identifying true negatives (TN) and false negatives (FN) is problematic, since unshown or unrated items cannot be reliably labeled without substantial computational cost and additional complexity. As a result, precision was computed using only true positives (TP) and false positives (FP) among the top-10 recommended items.

Our algorithm performed well for two of the three users, providing accurate recommendations and yielding promising potential purchases. The remaining user, who was specifically searching for game codes, emerged as an outlier. Because our dataset is platform-agnostic, the system did not distinguish between console types and instead returned codes across all platforms, rather than prioritizing those compatible with the user's own devices. This limitation highlights a clear avenue for future improvement, namely incorporating platform-specific constraints or user device profiles into the recommendation process.

8.2 User Rating Distribution

From 59 total evaluations across 11 sessions, users provided the following feedback:

Rating	Count	Percentage
Like (5.0)	42	71.2%
Dislike (1.0)	17	28.8%

Table 2: Distribution of user ratings on recommended products

The 71.2% like rate indicates that approximately seven out of ten recommendations were perceived as relevant by users, demonstrating reasonable recommendation quality for a cold-start scenario.

Metric	Value
Total Evaluations	59
Total Sessions	11
Total Likes (5.0)	42
Total Dislikes (1.0)	17
Overall Satisfaction Rate (%)	71.19
Overall Dislike Rate (%)	28.81
Average Ratings per Session	5.36
Average Session Satisfaction Rate (%)	67.32
Best Session Satisfaction Rate (%)	100.00
Worst Session Satisfaction Rate (%)	0.00
Unique Products Recommended	54
Products Recommended Multiple Times	5

Table 3: Comprehensive evaluation summary statistics

8.3 Statistical Confidence

We computed a 95% confidence interval for the overall satisfaction rate:

$$\text{CI}_{95\%} = [59.63\%, 82.74\%] \tag{2}$$

With a margin of error of 11.56%, we can state with 95% confidence that the true satisfaction rate for our recommender system lies between approximately 60% and 83%. This wide interval reflects the smaller sample size but with more user testing this interval could be narrowed even further.

9 Analysis and Discussion

9.1 Strengths

1. **Strong overall satisfaction:** The 71.19% satisfaction rate demonstrates that our hybrid TF-IDF and numerical feature approach generates relevant recommendations for the majority of cases.
2. **Diverse recommendation coverage:** With 54 unique products recommended across 59 evaluations and only 5 products appearing multiple times, the system avoids over-recommending popular items and provides varied suggestions.

3. **Effective for broad categories:** Performance metrics for general product searches (racing games, peripherals) show acceptable precision and strong ranking quality (NDCG).

9.2 Limitations

1. **High variance across sessions:** Session satisfaction rates ranged from 0% to 100%, indicating inconsistent performance that depends heavily on the user's initial profile selections.
2. **Cold-start constraints:** With only initial likes/dislikes to build the user profile, the system has limited information to personalize recommendations quickly.

10 Discussion and Limitations

Our project has several limitations:

- **Algorithmic simplicity.** The current system focuses on simple baselines rather than state-of-the-art models. This is intentional for pedagogical clarity but limits performance compared to methods that leverage text and multimodal information.
- **Domain restriction.** We restrict ourselves to the video games category. This improves coherence of recommendations but may limit generalization to other product types.
- **Cold-start and sparsity.** Users and items with very few interactions remain challenging. Addressing this would likely require side information (e.g., text, images) or explicit user profiles.
- **Scalability.** While SQLite is convenient, it is not optimized for serving large-scale, low-latency recommendation traffic. Scaling this system to production workloads would require more robust infrastructure.

Despite these limitations, the project successfully demonstrates the complete path from raw industrial-scale data to a working recommendation demo, and it provides a foundation for more advanced methods.

11 Conclusion

We have presented *AmazonRecommender*, an end-to-end recommendation system using the Amazon Reviews 2023 video game subset. The project emphasizes practical aspects of working with large real-world datasets: data ingestion, schema design, and integration with a web front end. Our framework is intentionally simple but extensible, making it suitable as a teaching tool and as a starting point for more ambitious research projects, such as integrating pretrained text encoders or sequence models.

Future work includes implementing and evaluating stronger collaborative and content-based models, adding support for text-driven search, and conducting a more thorough empirical study across multiple Amazon categories.

Acknowledgments and Disclosure of Funding

This project was developed as part of a course on recommender systems and data-intensive applications at the University of Washington. We thank the course staff for guidance, and we acknowledge the McAuley Lab for releasing the Amazon Reviews 2023 dataset and related tools.

References

[1] Y. Hou, J. Li, Z. He, A. Yan, X. Chen, and J. McAuley. Bridging Language and Items for Retrieval and Recommendation. *arXiv preprint arXiv:2403.03952*, 2024.

[2] J. McAuley, C. Targett, Q. Shi, and A. van den Hengel. Image-based Recommendations on Styles and Substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2015.

Amazon Reviews 2023 dataset.
https://nijianmo.github.io/amazon/index.html