A Comparative Study of Traditional and LLM-based Recommendation Systems on Amazon Review Data

Candidate Nr. 50450
The London School of Economics and Political Science
London, United Kingdom

Candidate Nr. 46476
The London School of Economics and Political Science
London, United Kingdom

Candidate Nr. 41195
The London School of Economics and Political Science
London, United Kingdom

Candidate Nr. 40781
The London School of Economics and Political Science
London, United Kingdom

Figure 1: Amazon Recommendation System

ABSTRACT

This project investigates e-commerce product recommendation by combining collaborative filtering with LLM-based techniques. We use the Amazon Reviews 2023 dataset and implement three main pipelines: ALS for collaborative filtering, LLM-based user preference summarization, and embedding-based retrieval with explanation generation (LLM-only). Our modular system extracts user profiles, encodes products, and retrieves relevant items using approximate nearest neighbor search. LLMs are also used to generate personalized recommendation reasons. While LLMs improve interpretability, they face scalability challenges. Our findings suggest that hybrid models offer a flexible and effective framework for large-scale personalized recommendations.

KEYWORDS

Big Data, Distributed Computing, Recommendation System, Amazon, ALS, LLM, Collaborative Filtering

ST446 Distributed Computing for Big Data, May 06, 2025, London, UK

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Candidate Nr. 50450, Candidate Nr. 46476, Candidate Nr. 41195, and Candidate Nr. 40781. 2025. A Comparative Study of Traditional and LLM-based Recommendation Systems on Amazon Review Data. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (ST446 Distributed Computing for Big Data).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In the era of digital commerce, the volume of user-generated data has grown at an unprecedented rate. Product reviews, user interactions, and browsing history now form vast datasets that present both an opportunity and a challenge: delivering personalized, relevant recommendations at scale. As traditional recommendation algorithms such as collaborative filtering reach their limitations in capturing nuanced user intent, especially in cold-start or sparse-data scenarios, attention is turning toward more sophisticated methods powered by language models.

This project explores and compares two fundamentally different approaches to recommendation: Alternating Least Squares (ALS), a widely used collaborative filtering technique suited for distributed environments, and Large Language Model (LLM)-based recommendation, an emerging paradigm that leverages semantic understanding of text. Using the Amazon Reviews 2023 dataset, we implement and evaluate both methods in a scalable big data pipeline built with PySpark and Google Cloud Dataproc, examining not only performance metrics but also recommendation diversity and interpretability.

This comparative framework is appropriate for highlighting the trade-offs between scalability and semantic richness in recommender systems. If the models perform as anticipated, we expect to observe that LLM-based methods may provide enhanced diversity and explainability, while traditional ALS remains more scalable and efficient within distributed environments.

2 RELATED WORK

The related work demonstrates an evolution in recommendation systems, from traditional collaborative filtering techniques like ALS (Meng et al., 2016) to more advanced deep learning methods such as Neural Collaborative Filtering (He et al., 2017). Further, semantic-based approaches such as DeepCoNN (Zheng et al., 2017) and BERT4Rec (Sun et al., 2019) leverage text data to enhance personalization. Recently, LLM-based recommendation systems (e.g., Li et al., 2023; Wang et al., 2023) have emerged, using language models for unified recommendation generation and natural language justifications. This report extends these ideas by comparing the scalability of ALS with the semantic richness and flexibility offered by LLM-based methods, directly addressing the strengths and weaknesses identified in existing research.

Collaborative Filtering and Matrix Factorization

- Neural Collaborative Filtering introduces deep models to learn non-linear user-item interactions, improving over traditional matrix factorization (He et al., 2017).
- The implementation of Alternating Least Squares (ALS) in Apache Spark MLlib is described by Meng et al. (2016), emphasizing scalability for large datasets.

Semantic and Hybrid Approaches

- McAuley et al. (2015) infer networks of complementary and substitutable products using product metadata and reviews, highlighting the value of textual data.
- DeepCoNN employs dual convolutional neural networks to extract semantic features from reviews for both users and items (Zheng et al., 2017).
- BERT4Rec applies BERT-style transformers for sequential recommendation, modeling user behavior as a language task (Sun et al., 2019).

LLM-based Recommendation Systems

- Li et al. (2023) – GPT4Rec proposes a generative framework for personalized recommendations by generating hypothetical user searches from item titles in their history, enhancing personalization and diversity [Li et al., 2023].
- Wang et al. (2023) – GeneRec introduces a generative recommender system using AI-generated content and user instructions, addressing limitations of traditional recommendation approaches [Wang et al., 2023].

3 DATA AND EDA

3.1 Data source

We will use the Amazon Reviews 2023 dataset published by the McAuley Lab, available on Hugging Face: Link.

The dataset is divided into two main components:

- **User Reviews**: Contains information about individual product reviews, with the following columns:
  - `rating`, `title`, `text`, `images`, `asin`, `parent_asin`, `user_id`, `timestamp`, `helpful_vote`, `verified_purchase`
- **Item Metadata**: Includes product-related information, with the following columns:
  - `main_category`, `title`, `average_rating`, `rating_number`, `features`, `description`, `price`, `images`, `videos`, `store`, `categories`, `details`, `parent_asin`, `bought_together`, `subtitle`, `author`

We downloaded the dataset in `.json` format using Google Cloud Platform (GCP).

3.2 EDA

3.2.1 *I. User Review Dataset EDA.*

The user review dataset contains 701,528 reviews from 631,986 unique users on 115,709 distinct beauty products. The accompanying metadata provides context such as category, brand, and price for each product.

The most reviewed product received 1,962 reviews, while the top reviewer submitted 165 reviews. Ratings are skewed toward higher values, with many items averaging 5.0 stars. Review text length varies considerably, with an average of 173 characters, and a range from 0 to 14,989 characters. Most reviews fall into the "Medium" category (50–200 characters).

Verified purchases show a slightly lower average rating (3.95) compared to non-verified reviews (4.08), suggesting verified users may be more critical. Helpfulness voting is sparse: the average number of helpful votes per review is 0.92, with 73% of reviews receiving none. Only 3% of reviews receive more than five helpful votes, indicating a highly right-skewed engagement pattern.

Figure 2: Distribution of user ratings for beauty products.

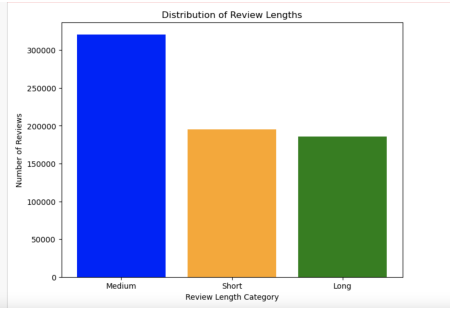

Figure 3: Distribution of review text lengths in the dataset.

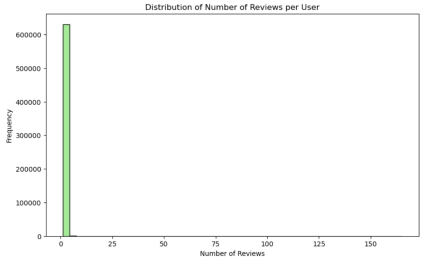

Figure 4: Distribution of number of reviews per user

The distribution is highly right-skewed, indicating that most users write only a few reviews. This suggests user activity is imbalanced.

3.2.2 II. Item Metadata Dataset EDA.

The metadata dataset includes 17 columns describing product-level features. While several fields, such as `details`, `average_rating`, and `main_category`, are fully populated, others exhibit significant missingness. In particular, the `author`, `subtitle`, `bought_together`, and `categories` fields are entirely null. Additionally, `features`, `price`, and `description` are missing in over 80% of entries, with `features` having the highest missing rate at 84.57%. The `store` field contains 11,337 missing values (10.07%), and even the `title` field, which is generally expected to be complete, has 12 missing entries.

Two dominant categories emerge:

- All Beauty: 112,135 products

Table 1: Missing Value Summary in Item Metadata Dataset

Column	Missing Count	Total	Missing Rate (%)
author	112,590	112,590	100.00
bought_together	112,590	112,590	100.00
categories	112,590	112,590	100.00
subtitle	112,590	112,590	100.00
features	95,213	112,590	84.57
price	94,886	112,590	84.28
description	93,428	112,590	82.98
store	11,337	112,590	10.07
title	12	112,590	0.01
details	0	112,590	0.00
average_rating	0	112,590	0.00
images	0	112,590	0.00
parent_asin	0	112,590	0.00
main_category	0	112,590	0.00
rating_number	0	112,590	0.00
videos	0	112,590	0.00

- Premium Beauty: 455 products

Despite being fewer in number, Premium Beauty products receive higher average ratings (4.03) than All Beauty products (3.94). Pricing mirrors this distinction:

- All Beauty: average price of $27.09
- Premium Beauty: average price of $48.36

These insights suggest that Premium Beauty items are not only better rated but also significantly more expensive, potentially reflecting a perceived or actual quality premium. The overall data sparsity and skewness emphasize the need for robust recommendation algorithms capable of handling such distributions. To better understand product themes, we also analyzed the most common terms in product titles.

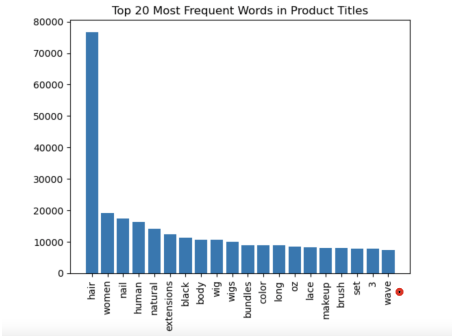

Figure 5: Top 20 Most Frequent Words in Product Titles.

The most frequently occurring words in the product titles are related to haircare and beauty products, with "hair" being the most common term, followed by other key terms like "women," "nail," "human," and "natural." Words such as "wig," "extensions," and "bundles" suggest a focus on hair products, while terms like "makeup," "brush," and "set" reflect a broader beauty and cosmetics theme.

Additionally, specific descriptors like "black," "color," and "long" are common, indicating that product titles often emphasize color and length attributes.

These findings from our exploratory analysis help guide the approach we take in building our recommendation models, which we explain next in the methodology section.

4 METHODOLOGY

4.1 Data Storage & Infrastructure

Our infrastructure includes Google Cloud Storage (GCS) for handling large-scale datasets, Google Cloud Dataproc combined with HDFS for distributed data storage and computation, and Google Colab with GPU support for implementing LLM-related work.

4.2 Modeling : ALS

The model-based collaborative filtering algorithm called Alternating Least Squares is a traditional form of applying a recommendation system based on user and item data. In this example, we have a user matrix with its relative dimensional features and a item matrix with its relative dimensional features. ALS in its essence is filling up a partially filled matrix which holds user ratings (n) for specific products (m). So our main user_item matrix A is of dimension mxn is not filled but only holds some ratings of some users for some items. Based on this partial knowledge, we now want to create a filled matrix with all user item ratings filled up. In order to achieve this kind of imputation, we are using what we already know, the so far given ratings from our users for the respective products.

First of all, we are creating two separate matrices, one matrix representing the users $X \in \mathbb{R}^{n\times k}$ and another item matrix with the dimension $Y \in \mathbb{R}^{m\times k}$. These two seperate matrices are representing our initial matrix A by setting $A = XY^T$. The latent factors/features are representing commonalities between the two matrices being the users and the items, for example representing a specific class of products which is only related to a specific age or gender group, or a product which is only bought for a specific need. The choice of the number k is arbitrarily and acts therefore a s a hyperparameter which can be tuned and also optimized by using cross validation.

Coming to the ALS algorithm and how the missing values are imputed by using the two matrices being connected with the latent features, we are setting up a Loss function in order to track the accuracy of XY^T approximating A for its given values. So now we need to find X and Y which miinimze the a loss function: in our case we choose a squared error loss function and add L2 regularization in order to prevent overfitting.

We have a partially observed rating matrix $A \in \mathbb{R}^{n\times m}$, and we wish to factor it as

$$A \approx X\,Y^T,$$

where

- $X = [x_1, \ldots, x_n]^T \in \mathbb{R}^{n\times k}$ are the user–feature vectors,
- $Y = [y_1, \ldots, y_m]^T \in \mathbb{R}^{m\times k}$ are the item–feature vectors,
- $\Omega \subseteq \{1, \ldots, n\} \times \{1, \ldots, m\}$ is the set of observed entries.

We define the regularized squared–error loss

$$f(X, Y) := \sum_{(i,j)\in\Omega} \left(A_{i,j} - x_i^T y_j\right)^2 + \lambda\left(\sum_{i=1}^{n} \|x_i\|_2^2 + \sum_{j=1}^{m} \|y_j\|_2^2\right), \quad (1)$$

where $\lambda \geq 0$ is a regularization hyper-parameter that penalizes large feature-vector norms to prevent overfitting.

The key variables used throughout this report are defined as follows. Let $A \in \mathbb{R}^{n\times m}$ denote the user–item ratings matrix, where n is the number of users and m is the number of items. We introduce k to represent the number of latent factors. The matrix $X \in \mathbb{R}^{n\times k}$ corresponds to the latent factor representation of users, and the matrix $Y \in \mathbb{R}^{m\times k}$ corresponds to the latent factor representation of items.

4.2.1 *Alternating Least Squares (ALS) Algorithm Outline.*

(1) *Initialization.*
- Randomly initialize the user–factor matrix $X \in \mathbb{R}^{n\times k}$ and the item–factor matrix $Y \in \mathbb{R}^{m\times k}$.
- Define $\Omega = \{(i, j) : A_{ij}$ is observed$\}$ and choose a regularization parameter $\lambda > 0$.

(2) *Iterative alternating updates.* Repeat until convergence (e.g. change in loss below a threshold or reaching `maxIter`):

(a) *Phase U (user update).* For each user $i = 1, \ldots, n$, solve

$$x_i \longleftarrow \left(\lambda I + \sum_{j:(i,j)\in\Omega} y_j y_j^\top\right)^{-1} \left(\sum_{j:(i,j)\in\Omega} A_{ij}\, y_j\right),$$

where x_i is the ith row of X and y_j the jth row of Y.

(b) *Phase V (item update).* For each item $j = 1, \ldots, m$, solve

$$y_j \longleftarrow \left(\lambda I + \sum_{i:(i,j)\in\Omega} x_i x_i^\top\right)^{-1} \left(\sum_{i:(i,j)\in\Omega} A_{ij}\, x_i\right),$$

updating the jth row of Y.

(3) *Termination.* Stop when one of the following is met:
- The decrease in the regularized squared-error objective falls below a preset tolerance.
- The number of iterations reaches `maxIter`.

(4) *Computational complexity.*
- Updating a single user x_i costs $O(m_i k^2 + k^3)$, where $m_i = |\{j : (i, j) \in \Omega\}|$.
- Updating a single item y_j costs $O(n_j k^2 + k^3)$, where $n_j = |\{i : (i, j) \in \Omega\}|$.

4.2.2 *ALS adaptions for training.*

During training of our ALS model on the full Amazon reviews dataset, the Spark driver began crashing due to excessive lineage growth and memory pressure. We addressed this instability through a combination of four strategies. First, we reduced the number of latent factors k, thereby shrinking the dimensions of the user (X) and item (Y) factor matrices and lowering the overall memory footprint. Second, we limited the number of iterations by decreasing the `maxIter` parameter, which shortened the training pipeline and reduced intermediate lineage. Third, we repartitioned the training RDD into 200 slices (`rdd.repartition(200)`), ensuring that ALS's user and item blocks are evenly distributed across executors rather than being recomputed on a single node. Finally, we enabled checkpointing every two iterations via `als.setCheckpointInterval(2)`, which truncates the RDD lineage and breaks long dependency chains. Together, these adjustments stabilized the driver, controlled memory usage, and allowed the ALS algorithm to converge successfully.

4.2.3 ALS cross-validation for hyperparameter choice. During training for the first model, as already discussed above, the kernel crashed several times and multiple cluster had to be created again to try to train the first model. Even after one successful training, the cluster and HDFS were not able to store multiple models subsequently and train multiple models by using a parameter-grid, which did not allow us to tune the hyperparameters accordingly due to the extremely large data size. In order to continue with evaluation of the model, the first model has been chosen with 3 latent factors, maximum iterations of 5, and a L2 regularization of $\lambda = 0.1$ to avoid overfitting. This setup led to a RMSE (Root mean squared error) of approximately $RMSE = 5.05$.

4.2.4 Solving kernel crashes partially. We have faced nonstop kernel crashes when installing the datasets API into our GCP cluster and then running the ALS model. The code has been working correctly, however the kernel crashed several times but at different spots during the code, sometimes when loading the data into the DataFrame and sometimes during the ALS training and sometimes during the generation of recommendations. WE have tried to launch different cluster over and over again, and we have additionally launched clusters in different regions in order to avoid peak usage clusters. Additionally, we tried to reduce the number of latent factors and other hyperparameters to a minimum in order to reduce the data inside the RAM. Moreover, using as few code as possible so we generate less RDD partitions was also another approach of how to minimize the likelihood of kernel crashes for the ALS model in GCP. All of these efforts unfortunately have not helped in order to avoid kernel crashes, which have continued to occur randomly at random stages when executing the ALS Model notebook. This notebook in the github is called "ALS Model HDFS.ipynb" and shows the process of loading the code.

Lastly, as the ultimate solution we downloaded the data locally in form of two json files, being the user_review data and the item_metadata for the Amazon products. After loading these inside a dedicated bucket in the GCP and launching them into the notebook pyspark environment into a DataFrame, the notebook had less kernel crashes. We succeeded to train the ALS model once again, retrieve the RMSE of a similar score (around 5.05) and were able to create first user recommendations and product recommendations. After that, we pursued with model adaptions and improving the model using Cross-validation by choosing the best hyperparameters of the model, which are being presented in the numerical section. Overall it can be stated that a major limitation has been the GCP environment which has has not been very reliable and compatible with launching huggingsface's dataset API as well as working without kernel crashes. The main notebook with the successful training, evaluation, and hyperparameter tuning of the ALS model and its recommendations can be found in the notebook "ALS Model Local.ipynb".

4.3 Modeling : LLM-Based Recommendation System

Our LLM-based recommendation system is designed in a modular and scalable fashion, comprising components for data preprocessing, user profile extraction, candidate retrieval and reranking, personalized recommendations with catchy ad-slogan style reasons. This modularity facilitates the seamless integration of different LLMs and supports efficient experimentation across tasks. We divided the whole system into the following parts with brief decision process for different models and methods.

Environment Setup As the core LLM inference and recommendation tasks were fully executed to Google Colab, the integration with GCS introduced unnecessary complexity and friction. As a result, we streamlined the entire workflow within Colab, enabling a smoother and more efficient pipeline for preprocessing, embedding, profiling, recommendation generation, and evaluation. The GPU RAM requires at least 8GB for running through the whole system, which is the eding limit for an LLM we implemented in this system.

Data Downloading and Processing, Define cold start users The pipeline begins by downloading raw datasets from the Hugging Face Hub, specifically `McAuley-Lab/Amazon-Reviews-2023`, and applies filtering rules to clean the review data. Then the processed user review data is saved into `user_reviews_batch.jsonl`, which serves as input for the user profile module. Any user not appearing in this json file will be labeled as a cold-start user in later recommendation and evaluation stage with fallback strategies or treated as a special group.

Product Embedding `BAAI/bge-small-en-v1.5`, an encoder-only model is used for product embeddings. Product titles, descriptions, and features are concatenated - for there are missing values in features and descriptions as is shown in EDA - and encoded into dense vectors, which are indexed using FAISS (`IndexFlatL2`) for efficient nearest neighbor retrieval. The index, ASIN mappings, and product metadata are saved to disk for downstream use.

Top-K ngrams are generated by extracting the most frequent word or phrase patterns (1–3 words) from product titles and descriptions, highlighting the most common product-related terms in the dataset. In our system, K is set to 200. We didn't use some available vocabulary sources such as nltk because that works more as a general dictionary, while in domain field of beauty products, it is better to focus more on the targeted sources so that meaningful extractions can be made.

LLMs for User Profiling and Recommendation Ad-slogans For User Profiling We firstly explored several LLMs from small to large, including `TinyLlama-1.1B-Chat-v1.0` and `Falcon-7B`, but ultimately selected `Nous-Hermes-2-Mistral-7B-DPO` for its robustness, availability, and high-quality output in summarization tasks. This is the largest model in this approach, costing 8GB GPU RAM to load.

For recommendation ad-slogen generation, we tested both template-based and LLM-based approaches, selecting `google/flan-t5-large` for its strong balance between expressiveness and computational efficiency.

Generating Recommendations Our pipeline first retrieves the top 200 candidate products for each user using a FAISS index based on vector similarity. It then applies a second filtering stage, where candidates are re-ranked by semantic similarity (cosine similarity) to the user's preferences and filtered for a minimum rating threshold (e.g., 3.3 stars). For non-cold-start users, the top 5 diverse, high-quality items are selected, and ad slogans are generated. For cold-start users (those lacking sufficient profile data), the system

bypasses the preference filters and provides fallback recommendations based on top-rated products, ensuring that all users receive five suggestions.

Evaluation for the system We evaluate the system both from system functioning perspective, as well as different stages of generating output. These metrics are also shown with experimental results in Numeric Results in the same order.

First we check Cold Start Ratio among users waiting for recommendations. Then we check Average Specificity Score for the LLM-generated user profile, showing how many concrete, product-relevant terms appear in user profiles, indicating the clarity and usefulness of extracted preferences. The concrete, product-relevant terms is defined in TopK ngrams.

For recommendations, we check mainly three metrics. Average Semantic similarity (**AvgCosSim**) measures how closely the recommended products match the user's preferences by computing the average cosine similarity between user profile embeddings and product embeddings. The higher, the better. Only non-cold start users will be taken into account for this metric as for cold start users, there is no user profile for them.

Ad Diversity (**AdDupRatio**) measures the proportion of unique ad slogans among all generated recommendations, reflecting how varied or repetitive the outputs are. The lower, the better. In numeric results, this metric will also be grouped in cold start and non-cold start users to see the performace of recommendation strategies toward these two groups.

Average Item Rating (**AvgRating**) calculates the mean product rating (typically from user reviews) across all recommended items, reflecting the overall quality level of the recommendations. The higher, the better. In numeric results, this metric will also be grouped in cold start and non-cold start users.

Overall, by combining small, medium, and large models across encoder, decoder, and encoder-decoder architectures, the pipeline achieves a robust balance of scalability, precision, and user engagement.

4.4 Modeling : LLM-Only Pipeline

In this section, we propose a recommendation pipeline that relatively more relies on Large Language Models (LLMs) without incorporating any collaborative filtering or traditional learning-based recommender methods. The pipeline is designed to emulate a consumer analyst that interprets user-generated content and ranks relevant products accordingly. The process consists of six stages, as stated in Figure 6, combining natural language understanding and semantic search. The first stage involves extracting user's reviews from the dataset and linking them with the corresponding product titles from the metadata. This creates a human-readable set of product experiences for each user. In the second stage, these reviews are passed into an LLM to summarize the user's preferences. We experimented with two LLMs for this summarization: `TinyLlama/TinyLlama-1.1B-Chat-v1.0` and `NousResearch/Nous-Hermes-2-Mistral-7B-DPO`. While TinyLlama is a lightweight model capable of fast inference on limited hardware, it suffers from a constrained input token limit (512 tokens), making it suboptimal when handling long review histories. In contrast, the Mistral-based model offers significantly higher input capacity

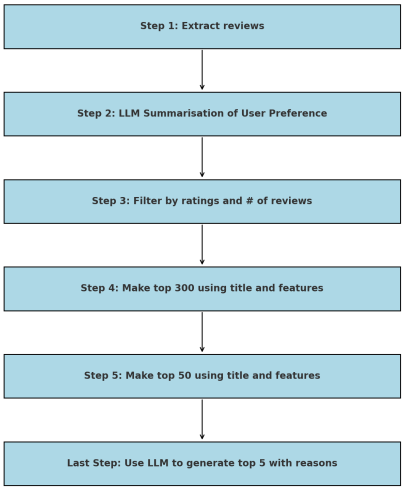

Figure 6: LLM-Only Pipeline Description

(up to 8192 tokens), which enables more robust summarization by capturing fuller context. Consequently, we adopt the Mistral model for our final implementation.

Following summarization, the third step filters the item pool based on predefined thresholds for average rating ($\geq$ 4.0) and number of reviews ($\geq$ 5), significantly reducing the candidate space. In the fourth and fifth stages, SBERT-based semantic matching is used to further narrow down the candidates: the user's summary is first matched against product titles (to retrieve top 300), then against title+feature concatenations (to retrieve top 50). Finally, in the sixth stage, the refined product list and the user summary are input into the LLM for reranking.

Two major uses of prompt engineering occur in this pipeline: (1) during user summarization, and (2) during reranking. In the summarization step, structure prompting and role prompting are employed. The model is instructed to write exactly three sentences that address: (i) common patterns in purchased items, (ii) what the user prioritizes, and (iii) what the user is likely to buy in the future. This formulation leads to clearer, non-redundant preference profiles. In the reranking step, a structured answer format is enforced using a response scaffold with explicit constraints (e.g., "Respond in the following format: Rank 1: Product: … Reason: …"). Moreover, instructions explicitly emphasize that the ranking should be based strictly on alignment with the user summary, preventing the model from being distracted by superficial product features.

This end-to-end pipeline demonstrates how LLMs can be integrated into both user modeling and item selection, using language alone to drive personalized product recommendations.

4.5 Case Study: Example output of models

Customer:
AEEOGN7VRJZEAGIGS3OZZDXS4YAQ
Product Bought:
Wowlife 15inch / 40cm Long Wig Two Tone Gradually Varied One Piece Straight Hair Replacement Brown Dark Blue

User Review:
"beautiful wig, very soft. bangs are WAY too long, going to have to cut them which is a bummer. (picture is after I cut the bangs)"

1. ALS Recommendations:

- Multi-functional Portable Carrying Bag for Microsoft Surface Pro 4
- 6-Pack Rhinestone Non-slip Wave Headbands
- 3-Piece Nail Drill Bit Set for Manicure/Pedicure
- Aloe Vera Juice Carotene Lipstick (RED)
- BBL 7pcs Champagne Gold Makeup Brushes Set

2. LLM-Based Pipeline:
User Profile (Extracted):

- **Preferred products:** wigs
- **Liked features:** softness, bangs
- **Dislikes:** long bangs
- **Potential interests:** cutting tools

Top Recommendations (with ad copy):

- **B015IK0NJ4|wigs A-BU036G** – *Wigs Are Easy To Style.*
- **B07V55KGH2|wigs** – *Hair that loves you back, wigs, softness, bangs.*
- **B08M5NLFHF|Long Wigs** – *Hair that loves you back, wigs, softness, bangs.*
- **B01F2WQUBI|wigs A-BU066** – *You will love the softness of this wig.*
- **B08HXPFZPL|headband wigs** – *Jieyan's curly wigs are the perfect choice for the woman who likes softness and bangs.*

3. LLM-Only Pipeline:
User Preference Summary:

(1) The customer is concerned with the quality and softness of the wig, but found the bangs to be too long.
(2) The customer values softness and appearance in their wig purchases.
(3) The customer is likely to purchase wigs in the future, possibly focusing on finding the right bang length.

Top Recommendations (with explanations):

- **Rank 1:** Narrative Cosmetics Alcohol Palette Refill – *Customization and replacement fit user's preference for control and quality.*
- **Rank 2:** Reyane Tradition R2B2 Fragrance – *Confidence-boosting scent with unique top notes.*
- **Rank 3:** Benefit Cosmetics Blush Bunch Set – *Fun-sized variety encourages exploration and personalization.*
- **Rank 4-5:** TOO FACED Natural Lust Eye Palette – *30 versatile shades for diverse makeup needs.*

5 NUMERICAL RESULTS

5.1 ALS

After switching from HDFS to locally downloaded files and uploading these files into a dedicated bucket, the ALS model was being trained for first arbitrarily chosen hyperparameters (k = 5, maxiter = 5, regularization parameter = 0.1). This model yielded a RMSE of around 5.05 which leaves plenty room of improvement. Therefore

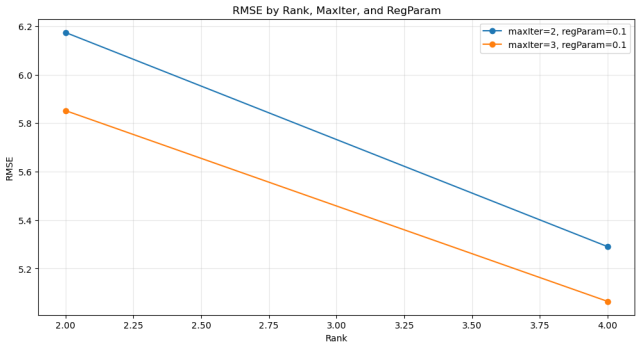

Figure 7: Cross-validation of ALS model

we chose to use cross validation in order to find the best setup for the ALS model, such that the rank(number of latent factors), the number of maxiaml iterations and the regularization parameter are chosen in a way which minimises the RMSE possibly best. In Figure 7 we are able to observe the chart of or cross validation for our ALS model:

Here we can observe that a higher number of latent factors and a higher amount of maximum iterations inside the ALS model yield a significantly lower RMSE and are able to improve the ALS model. This is reasonable, since:

(1) **Greater representational capacity.** Each latent factor can be thought of as a hidden "dimension" along which users and items can align (for example, make-up preference, price sensitivity, brand affinity, etc.). By increasing the rank, the model has more dimensions in which to separate and relate users and items, allowing it to capture subtler distinctions and more complex co-occurrence patterns, thus reducing prediction error.
(2) **Finer-grained similarity structure.** With only a few factors, very different items or users may be forced into the same coarse cluster. Adding factors lets the model express multiple orthogonal notions of similarity (e.g. "bold colors vs. subtle colors" and "lipstick vs. lip gloss"), so that two users who buy similar beauty products but rate differently can still end up close together in factor space—improving recommendation accuracy.
(3) **Better convergence through more iterations.** ALS alternates between fixing user factors and fixing item factors, solving a least-squares problem at each step. If we stop too early, the alternating updates have not fully propagated information, yielding a suboptimal factorization. Allowing more iterations lets the factors "settle" into a stable solution that more fully minimizes the squared error.
(4) **Trade-off and diminishing returns.** Each extra factor or iteration adds compute cost and, if pushed too far without proper regularization, risks overfitting to noise. In practice, RMSE often drops sharply moving from very low ranks (e.g. 5→20) or few iterations (e.g. 5→20) and then plateaus, marking the point where most signal has been captured.

(5) **Interaction with regularization.** Higher-rank models can memorize idiosyncrasies, so it is common to adjust the regularization parameter (e.g. increase λ slightly as rank grows). With well-chosen regularization, the model uses extra degrees of freedom to reduce bias without incurring too much variance, and additional iterations ensure convergence to that optimal factorization.

In summary, increasing the rank provides richer latent spaces to better approximate the user–item rating matrix, and allowing more iterations ensures that alternating optimization fully exploits those dimensions. When paired with proper regularization, this leads to a lower RMSE on held-out data.

5.2 LLM-based

Metric	Value
Overall	
Cold Start Ratio	54%
Average Specificity Score	50%
AvgCosSim	71.79%
AdDupRatio	61.60%
AvgRating	4.11
Cold Start Users	
AdDupRatio	99.26%
AvgRating	3.95
Non-Cold Start Users	
AdDupRatio	17.39%
AvgRating	4.31

Table 2: Summary of Recommendation System Evaluation Metrics with User Group Breakdown

The pipeline demonstrates strong overall performance. We evaluated our LLM-based recommendation pipeline over a sample of 50 users, where 54% are cold start ones. This is due to invalid review texts or LLM didn't extract valid features from the reviews. This number is the highest among all the experiements up till now but still acceptable as samples differ.

The **Average Specificity Score** among non-cold-start users was 50%, suggesting that approximately half of the extracted profile terms were concrete, product-relevant entities. This highlights both the strength of the LLM-generated profiles and the opportunity to further enhance specificity, which could improve recommendation precision. But right now it's much better than a random guess.

The average semantic similarity (**AvgCosSim**) between user profiles and recommended products reached an average of 71.79% among non-cold-start users, which is slightly good, more optimization work can be

The overall ad diversity (**AdDupRatio**) was 61.60%, this metric doesn't look promising because there are many cold start users in it, as for cold start users it reaches 99.26%, leaving non-cold start users only 17.39%, which means for each non-cold start user, there will be about 1 to 2 out of 5 ad slogans are the same, keeping the slogans diversed enough. Sometimes it is acceptable to have 2 to 3 the same as a repetition to the users to stimulate them to buy in real e-commerce platform strategies. Also for cold start users the AdDupRatio is almost 100% due to the fallback strategies are too simple: recommend popular products without preference filtering, this could lead to the same category of products only, but the slogan needs input of user profile, also our fallback strategy is not diversed enough.

The average item rating (**AvgRating**) across all users was 4.11, confirming the quality of the recommendation pool to some extent. When looking into it deeper, AvgRating for cold start users is lower than non-cold ones, 3.95 to 4.31, underscoring the effectiveness of personalized recommendations.

5.3 LLM-only

The LLM-only pipeline requires significantly more computational resources than both the ALS and hybrid LLM-based approaches. Specifically, it demands a GPU more powerful than NVIDIA L4 and took approximately 3 hours and ~~3 minutes~~ to process a batch of 50 sample user IDs, making it the least scalable option among the three.

In terms of performance, the LLM-only method achieved an average **AdDupRatio** of 0.056 and an average **AvgCosSim** of 0.363. Compared to the LLM-based pipeline (AdDupRatio: 0.616, AvgCosSim: 0.717), the LLM-only approach shows a drastically lower AdDupRatio, indicating much higher diversity and minimal duplication in the recommended item lists. However, the lower AvgCosSim suggests that the recommended items are less semantically aligned with user preferences, implying a trade-off between recommendation diversity and precision.

This divergence can be attributed to the architecture of the pipeline: by relying solely on LLMs for both user profile generation and product embedding, the system explores a wider product space without the collaborative filtering signals that typically anchor recommendations more tightly to user behavior. As a result, the LLM-only method tends to generate more exploratory recommendations, albeit at the cost of reduced alignment with core user interests.

While the user summary is often well-structured and reflective of actual preferences, the final top-5 recommendations frequently fail to match the user's expressed interests. This disconnect suggests limitations in the end-to-end coherence of the LLM-only pipeline, likely stemming from the lack of task-specific fine-tuning and the constraints of running relatively small open-weight models in a local environment.

Despite its limitations in efficiency and precision, the LLM-only pipeline excels in generating personalized, natural language explanations for each recommendation. This enhances interpretability and user engagement, making it more appropriate for applications where transparency or persuasive communication is prioritized over recommendation accuracy or scalability.

Overall, the LLM-only pipeline represents a highly expressive but computationally demanding approach. Its strength lies in explainability and diversity, but without further tuning or scaling, it currently falls short as a reliable standalone recommender system.

6 CONCLUSION

This report presented a comprehensive study on e-commerce product recommendation by combining both traditional collaborative

filtering methods and cutting-edge LLM-based approaches. Specifically, we implemented the Alternating Least Squares (ALS) algorithm for collaborative filtering and developed two modular LLM-based pipelines to extract user preferences, generate product embeddings, retrieve relevant items, and produce personalized recommendations with reasons or even ad slogans. By integrating these two paradigms, we aimed to compare their strengths and explore their complementary benefits in improving recommendation quality. We highlighted key methodological decisions, including the selection of open-source models, the design of modular components, and the use of approximate nearest neighbor search to efficiently handle large-scale product datasets.

Despite the promising results, several limitations remain in LLM-based systems. For user profile construction, we relied solely on the Mistral-7B model, which, although robust in its outputs, incurs high time costs when applied to large user datasets. For example, generating profiles for approximately 10,000 users required over 10 hours, making it impractical for real-time or enterprise-scale applications. Future work should explore lighter-weight models or investigate non-open-source or commercial LLMs to improve scalability.

For product embedding, while we adopted the BGE-small model for its speed and stability, many alternative LLM-as-encoder models are available and could be explored in future experiments to improve embedding quality and downstream retrieval performance. For recommendation reason generation, we found that lightweight models were generally sufficient for most tasks, though fine-tuning model parameters and prompt engineering remains time-intensive and warrants further investigation.

Finally, for the recommended product list itself, future work could explore integrating additional business constraints, diversity measures, or user feedback loops to further refine and personalize recommendations. Overall, this work provides a flexible foundation that combines both collaborative filtering and LLM-based techniques, offering multiple avenues for extension and improvement in large-scale recommendation systems.

7 BIBLIOGRAPHY

- He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T. S. (2017). Neural collaborative filtering. *Proceedings of the 26th International Conference on World Wide Web (WWW).*
- Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., ... & Zaharia, M. (2016). MLlib: Machine learning in Apache Spark. *Journal of Machine Learning Research, 17*(34).
- McAuley, J., Pandey, R., & Leskovec, J. (2015). Inferring networks of substitutable and complementary products. *Proceedings of the 21st ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD).*
- Zheng, L., Noroozi, M., & Yu, P. S. (2017). Joint deep modeling of users and items using reviews for recommendation. *Proceedings of the 10th ACM International Conference on Web Search and Data Mining (WSDM).*
- Sun, F., Liu, J., Wu, J., Pei, C., Lin, X., Ou, W., & Jiang, P. (2019). BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM).*
- Li, J., Zhang, W., Wang, T., Xiong, G., Lu, A., & Medioni, G. (2023). GPT4Rec: A generative framework for personalized recommendation and user interests interpretation. arXiv preprint arXiv:2304.03879.
- Wang, W., Lin, X., Feng, F., He, X., & Chua, T.-S. (2023). Generative recommendation: Towards the next-generation recommender paradigm. arXiv preprint arXiv:2304.03516.

8 STATEMENT OF INDIVIDUAL CONTRIBUTIONS

Task	50450	46476	41195	40781
Data Acquisition and Preparation	0%	0%	45%	55%
EDA	0%	95%	0%	5%
Modeling	33.3%	0.0%	33.3%	33.3%
Evaluation	33.3%	0.0%	33.3%	33.3%
Notebook Re-organization	25%	25%	25%	25%
Report Writing	25%	25%	25%	25%

Table 3: Individual Contributions by Task

REFERENCES

Received 01 May 2025