ReelReason: A LLM-Driven Conversational Movie Recommendation Platform

Emily Wang
Computer Science
Columbia University
New York, New York
eaw2233@columbia.edu

Eesun Moon
Computer Science
Columbia University
New York, New York
em3907@columbia.edu

Nikhil Sharma
Computer Science
Columbia University
New York, New York
ns3942@columbia.edu

***Abstract*—This paper describes a movie recommendation system and user interface that combines traditional retrieval-augmented generation and LLM-reranking and reasoning for an advanced interactive user experience that understands user preferences and personalization. This project uses MovieLens, TMDB, ReDial, and more datasets, embedded by bge-base or text-embedding-3-large into a vector database, then user inputs in the form of preference metadata/conversation which is embedded in the same way, and a FAISS retrieval of top 20 movies before a fine-tuned GPT-4o-mini system reranks for the top 5 options, and explains the choices. The full UI takes into account user ratings, reviews, as well as conversation, and generates visualizations and recommendations based on user taste.**

***Index Terms*—movie recommendation, large language models, explainability, conversational AI, taste embeddings, retrieval-augmented generation, LLM-as-ranker**

I. Introduction

Modern streaming platforms have revolutionized content discovery and presentation by leveraging sophisticated recommendation algorithms to surface personalized movie and TV show suggestions. According to Ampere's Consumer data, more viewers than ever now trust the personalized suggestions offered by streaming platforms rather than recommendations from friends and family. Among global Internet users, 26% now rely on streamers' algorithm picks to decide what TV and film to watch, versus 23% who rely on word-of-mouth recommendations [24]. Many consumers feel that the quantity of content released on a monthly basis has outpaced the amount of content they can consume, thus they utilize the personalized suggestions to prioritize what to watch. However, these systems face a fundamental transparency problem: users see *what* to watch but rarely understand *why* they should watch it. Current recommendation engines, from Netflix to Amazon Prime, operate as "black-boxes" that provide minimal insight into their decision-making process. In fact, economics research on platform incentives suggests that streaming services don't always optimize recommendations purely for user satisfaction; they sometimes have incentives to steer users toward content that reduces costs or favors certain partners. For example, some studies find that platforms may bias recommendations to favor cheaper content or content from partners with stronger negotiating power, rather than always surfacing the most relevant titles for the viewer [25].

This opacity and lack of insight creates several challenges for users. First, it hinders decision-making, as viewers cannot evaluate whether a recommendation truly aligns with their preferences without investing time to watch it, a challenge that explainable recommendation research aims to address to improve user comprehension and trust [26]. Second, it prevents users from refining their recommendations through meaningful dialogue, forcing them to rely on implicit feedback mechanisms like watch history and ratings. Third, it limits users' ability to discover how their tastes evolve over time or explore their cinematic preferences in an interpretable way.

The rise and widespread use of large language models (LLMs) and conversational AI presents a timely opportunity to address these limitations. Conversational explainability has become a hallmark of modern AI systems, from chatbots to code assistants, yet entertainment recommendations (which affect millions of daily users from casual viewers to veteran cinephiles) have largely remained opaque. Users increasingly expect AI systems to not only provide answers but to justify their reasoning in natural language, a theme emphasized in broader work on AI transparency and explainability [27]. This creates demand for recommendation systems that combine accurate suggestions with the reasoning behind them.

A. Research Questions & Development Goals

This project addresses three core research questions centered on building an explainable, conversational movie recommendation system:

RQ1: Taste Representation – How can we encode user "taste" from heterogeneous data sources including ratings, metadata, and natural dialogue into a unified representation that captures both explicit preferences and implicit signals?

RQ2: LLM-Enhanced Reasoning – Can LLM reasoning improve both recommendation retrieval quality and generate meaningful, human-interpretable explanations that help users understand why specific movies match their preferences?

RQ3: Interactive Personalization – Can we design an interface that allows users to interactively discover and refine their taste profile through conversational feedback, moving beyond passive consumption to active exploration?

To address these questions, we developed ReelReason, an LLM-driven conversational movie recommendation platform with three development goals: constructing taste embeddings from multiple data sources, implementing LLM-augmented conversational recommendations through retrieval-augmented generation (RAG) with LLM-as-ranker reasoning, and achieving lightweight personalization with built-in explainability.

B. Contributions

This work makes several novel contributions to the intersection of recommender systems and generative AI:

1) **Taste Embedding Framework**: We introduce a unified embedding approach that fuses structured preference signals (ratings, metadata) with unstructured conversational data to create high-dimensional user ”taste vectors.” Unlike traditional collaborative filtering approaches, our embeddings capture semantic relationships and enable interpretable visualization of user preferences.
2) **LLM-as-Ranker Architecture**: We demonstrate that combining semantic retrieval with LLM reasoning significantly improves recommendation quality. Our hybrid architecture uses taste embeddings for candidate retrieval, then employs fine-tuned GPT-4o-mini for reranking and explanation generation, achieving measurable improvements in hit rate and user acceptance compared to embedding-only baselines.
3) **Conversational Explainability**: Our system generates human-readable explanations for each recommendation using chain-of-thought prompting, providing rationales like ”Because you loved The Shawshank Redemption’s themes of hope and redemption...” that help users make informed viewing decisions.
4) **Taste Wrapped Visualization**: We present an interactive feature that visualizes users’ taste profiles through 2-D UMAP projections and narrative summaries, enabling users to understand their cinematic preferences and how they evolve over time.

II. Background and Related Work

A. Background

1) Movie Recommendation Systems: Movie recommender systems have evolved significantly over the past two decades, transitioning from simple filtering algorithms to sophisticated approaches that often involve machine learning algorithms trained on millions of user’ data points. Traditional recommendation systems mainly employ three main filtering techniques: collaborative filtering, content-based filtering, and hybrid methods [28].

Collaborative filtering operates on the principle of ”wisdom of the crowds,” making recommendations based on patterns in user-item interaction data. This approach can be further categorized into neighborhood-based methods (user-based and item-based) and model-based methods, with matrix factorization being the most prominent model-based technique [2], [28]. Although collaborative filtering can identify complex patterns in user preferences, it has several limitations: the cold-start problem for new users or items, data sparsity when users rate only a small fraction of available items, and lack of transparency in explaining why specific items are recommended.

Content-based filtering recommends items by matching item features with user preferences. In movie recommendation, these features typically include genre, director, actors, and plot keywords [28]. Content-based approaches provide more intuitive explanations by explicitly showing which features contributed to a recommendation. However, they are limited by feature availability and struggle to discover unexpected recommendations outside of a user’s established preferences.

Hybrid approaches combine multiple filtering techniques to take advantage of the strengths of different methods while minimizing their individual weaknesses [28]. Modern hybrid systems often integrate collaborative filtering with content-based features to improve both recommendation accuracy and explainability.

2) Explainability in Recommender Systems: The importance of explainability in recommender systems has gained significant attention as these systems become more prevalent in user-facing applications. Tintarev and Masthoff identified seven potential benefits of explanations in recommendation systems: transparency, scrutability, trust, effectiveness, persuasiveness, efficiency, and satisfaction [30].

Transparency refers to exposing the reasoning behind the recommendation mechanism, allowing users to understand how the system works. **Justification** provides users with reasons why a specific item was recommended, which can differ from true transparency by offering plausible explanations that are user-friendly rather than algorithmically accurate [30]. Research has shown that, while complete transparency is desirable, simpler justifications often achieve better user satisfaction and trust.

Explanation scope can target different parts of the recommendation process: the input (user model), the process (algorithm’s inner workings), or the output (recommended items) [30]. Most existing systems focus on explaining outputs, while explanation of the recommendation process and user models remains underexplored.

Explanation formats vary from textual explanations (sentence-level or feature-level) to visual explanations using charts, graphs, and interactive visualizations [30]. Recent research on explainable recommendations emphasizes the value of visual explanations, which can convey information more efficiently than text while requiring less cognitive effort to process. However, the design of effective visual explanations requires careful consideration of the underlying tasks, data types, and user characteristics.

3) Generative AI in Recommendation: The growth of generative AI has unlocked new possibilities for recommendation systems. Variational Autoencoders (VAEs) represent an approach to leverage generative models for recommendations [29]. VAEs learn latent representations of user-item interactions in a continuous latent space, enabling them to: (1) generate meaningful item representations, (2) handle sparse rating data more effectively, (3) capture complex, non-linear patterns in user preferences, and (4) promote diversity in recommendations.

The VAE architecture consists of an encoder that compresses input data into a latent space and a decoder that reconstructs recommendations from this compressed representation [29]. The reparameterization trick allows the model to sample from the latent distribution while maintaining differentiability for gradient-based optimization. This approach has shown improvements over traditional collaborative filtering methods in metrics such as RMSE and MAE on standard datasets like MovieLens.

More recently, Large Language Models (LLMs) have emerged as powerful tools for recommendation tasks. Unlike VAEs, which learn abstract latent representations, LLMs can leverage pre-trained knowledge about movies, user preferences, and semantic relationships to generate recommendations with rich, natural language explanations.

B. Related Work

1) Traditional Movie Recommender Systems: Jayalakshmi et al. [28] provide a comprehensive survey of traditional movie recommendation systems, highlighting both classical machine learning approaches and metaheuristic optimization methods. Common classical approaches include:

K-means clustering groups users or items based on similarity, enabling neighborhood-based collaborative filtering. The algorithm measures similarity using correlation or distance metrics, selects neighbors through threshold-based or top-N techniques, and computes predictions based on weighted neighbor ratings [28]. However, K-means clustering faces challenges with cold-start problems (insufficient data for new users), data sparsity (users rate few items), and scalability (computational cost increases with data size).

Principal Component Analysis (PCA) addresses dimensionality reduction by creating covariance matrices and computing eigenvectors to identify principal components [28]. PCA-based methods can handle larger feature spaces more efficiently than basic K-means but still struggle with the interpretability of latent dimensions.

Metaheuristic algorithms such as genetic algorithms, artificial bee colony, and cuckoo search have been applied to optimize recommendation quality by avoiding local optima and improving convergence [28]. While these approaches show improvements in accuracy metrics, they generally do not address the fundamental challenge of explainability.

Challenges in Traditional Systems: Jayalakshmi et al. [28] identify several persistent challenges: (1) *Cold-start problem*: New users or items lack sufficient rating history, (2) *Sparsity*: Users typically rate a small fraction of available items, (3) *Scalability*: Computational complexity increases with users and items, (4) *Diversity*: Systems often recommend safe, popular items rather than diverse suggestions, and (5) *Explainability*: Complex algorithms, especially deep learning and metaheuristic approaches, operate as ”black boxes.”

2) Generative AI-Based Recommender Systems: Recent work has explored generative models for recommendation. The VAE-based approach by Tung et al. [29] demonstrates how generative AI can address some limitations of traditional methods. Their system uses a VAE architecture with: (1) an encoder that maps user-item interactions to a latent space, (2) a decoder that reconstructs recommendations from latent representations, and (3) a loss function combining reconstruction loss and KL divergence.

The VAE approach shows several advantages over traditional collaborative filtering: (1) *Handling sparsity*: VAEs can infer preferences even with limited rating data by learning continuous latent representations, (2) *Diversity*: Sampling from the latent distribution generates diverse recommendations, and (3) *Capturing complexity*: The non-linear neural network architecture captures complex user-item patterns.

However, Tung et al. [29] note that VAE-based systems still face explainability challenges. The latent dimensions learned by VAEs lack intuitive semantic meaning, making it difficult to explain why specific items are recommended. Additionally, VAE explanations typically require separate post-hoc explanation modules rather than providing explanations as a natural byproduct of the recommendation process.

3) Explainable Recommendation Systems: Chatti et al. [30] provide an extensive survey of explainable recommendation systems with a focus on visualization techniques. Their analysis reveals several important findings:

Explanation Aims: While multiple explanation aims exist (transparency, trust, effectiveness, satisfaction, etc.), most systems prioritize *justification* over *transparency*. Justification provides user-friendly reasons for recommendations, while transparency reveals the actual algorithmic reasoning. The survey found that only 13 of 33 examined systems focused on transparency, with the majority opting for simpler justifications [30].

Explanation Methods: The survey categorizes explanation methods into: (1) *Content-based explanations*: Show item features matching user preferences (e.g., ”This movie has the action and sci-fi genres you enjoy”), (2) *Collaborative explanations*: Leverage user or item similarities (e.g., ”Users similar to you enjoyed this movie”), (3) *Social explanations*: Use social network information (e.g., ”Your friend John liked this movie”), and (4) *Hybrid explanations*: Combine multiple methods for richer explanations. Content-based explanations proved most popular (25 of 33 systems) due to their intuitive nature—users can easily understand recommendations based on explicit features [30].

Visualization Techniques: Common visualization approaches include: (1) *Node-link diagrams* for showing relationships between users, items, and features, (2) *Bar charts* for comparing feature relevance or similarity scores, (3) *Tag clouds* for displaying prominent features or keywords, and (4) *Interactive visualizations* enabling users to explore and control recommendations. The survey emphasizes that effective explanations should match visualization techniques to the underlying task and data type. For example, node-link diagrams excel at showing network relationships, while bar charts effectively compare attribute similarities [30].

Interactive Explanation: Chatti et al. [30] identify interactive explanation as an underexplored but valuable area. Interactive systems allow users to: (1) ask follow-up questions about recommendations, (2) modify their preferences and see updated recommendations, (3) explore alternative explanations at different levels of detail, and (4) provide feedback to refine the system's understanding. The survey notes that only a few systems provide truly interactive explanations, with most offering static, one-way explanations [30]. This represents a significant opportunity for improvement, as interactive explanations support iterative refinement of user mental models about how the recommender system operates.

C. Research Gap

While existing research has made significant progress in movie recommendation, several gaps remain:

1) **Limited Explainability in Generative Models**: Generative AI approaches like VAEs [29] improve recommendation quality but struggle to provide intuitive explanations. The latent representations learned by these models lack semantic interpretability, requiring separate explanation mechanisms.
2) **Static vs. Interactive Explanations**: Most explainable systems provide static, one-way explanations [30]. True conversational explanation, where users can ask follow-up questions and iteratively refine their understanding, remains rare.
3) **Cold-Start with Rich Semantics**: Traditional systems struggle with cold-start problems [28], while generative models need sufficient training data. There is an opportunity to leverage pre-trained language models that already encode rich semantic knowledge about movies and preferences.
4) **Integration of Explanation and Recommendation**: Many systems treat explanation as a post-hoc addition to the recommendation process [30]. Integrating explanation generation directly into the recommendation pipeline could produce more coherent and trustworthy systems.
5) **Personalized Taste Representation**: While VAEs learn latent taste representations [29], these representations are not easily interpretable or modifiable by users. A more transparent taste model would enable users to understand and control their recommendations better.

D. Our Contribution

ReelReason addresses these gaps by combining LLM-based generative AI with conversational explainability:

1) **LLM-Enhanced Embeddings**: We use OpenAI's text-embedding-3-large model to create taste embeddings that capture rich semantic understanding of user preferences, going beyond traditional collaborative filtering or VAE latent spaces.
2) **Conversational Explainability**: Our chat interface enables truly interactive explanations where users can ask questions, seek clarification, and iteratively refine their preferences through natural language conversation.
3) **Semantic Cold-Start Solution**: By leveraging LLMs' pre-trained knowledge, we can generate meaningful recommendations even for new users based on minimal preference input, addressing the cold-start problem with rich semantic understanding.
4) **Integrated Explanation Generation**: Our LLM-based re-ranking naturally produces explanations (e.g., "Because you loved...") as part of the recommendation process, rather than as a separate post-hoc step.
5) **Transparent Taste Model**: The "Taste Wrapped" feature visualizes users' taste embeddings and preferences, providing transparency into how the system understands their preferences while remaining user-friendly.

Our approach represents a synthesis of insights from traditional recommender systems [28], generative AI methods [29], and explainable recommendation principles [30], creating a system that is both powerful and transparent.

III. System Architecture & Methodology

A. System Overview

ReelReason employs a modular architecture that separates concerns across six primary components: user input collection, data management, taste embedding generation, recommendation retrieval and ranking, LLM-based reasoning, and frontend presentation. Figure 1 illustrates the complete system architecture and data flow.

1) Architecture Overview: The system follows a retrieval-augmented generation (RAG) paradigm, where semantic retrieval narrows the candidate space before LLM reasoning produces final recommendations with explanations. This two-stage approach balances computational efficiency with recommendation quality: vector similarity search operates in milliseconds across the full movie corpus, while the more expensive LLM inference is reserved for a small candidate set.

The data flow proceeds as follows:

1) **User Input Layer**: Users provide preference signals through three channels: (a) TMDB OAuth integration imports existing ratings and favorites, (b) an onboarding flow captures movie selections and natural language "vibe" descriptions, and (c) ongoing reviews provide continuous feedback.
2) **Data Layer**: User preferences are aggregated into a unified profile combining structured data (movie IDs, numerical ratings) with unstructured signals (vibe text, review content). The TMDB API enriches movie references with metadata including genres, cast, plot summaries, and poster imagery.
3) **Embedding Engine**: The Taste Embedding Generator transforms user profiles into dense vector representations. Movie embeddings are pre-computed offline using OpenAI's text-embedding-3-large model, producing 3072-dimensional vectors for approximately 60,000

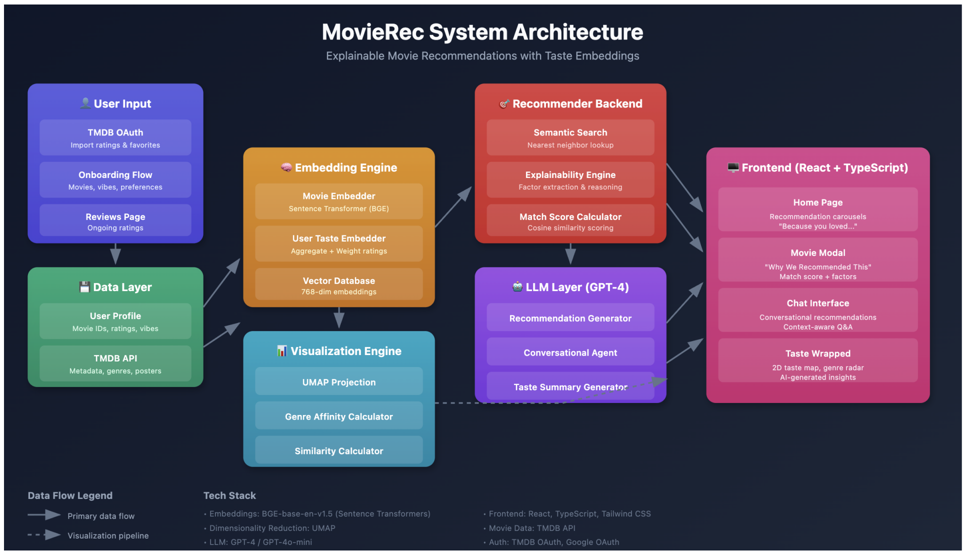


Fig. 1. ReelReason system architecture showing data flow from user input through taste embedding generation, semantic retrieval, LLM re-ranking, and frontend presentation. The system employs a RAG paradigm where vector similarity search narrows candidates before GPT-4o-mini generates personalized recommendations with explanations.

films. User embeddings are computed in real-time as weighted aggregations of their preferred movies' vectors, placing users in the same semantic space as the movie corpus.

4) **Recommender Backend**: Given a user embedding, the system performs approximate nearest neighbor search to retrieve the top-k candidate movies (typically $k = 20$). These candidates, along with the user's taste profile, are passed to a fine-tuned GPT-4o-mini model that re-ranks candidates and generates personalized explanations.
5) **LLM Layer**: The fine-tuned language model serves dual purposes: (a) re-ranking candidates based on learned preference patterns beyond pure vector similarity, and (b) generating natural language explanations that reference specific user preferences (e.g., "Because you loved Inception's mind-bending narrative...").
6) **Visualization Engine**: A parallel pipeline projects user embeddings into interpretable 2D space using UMAP dimensionality reduction, computes genre affinity scores, and calculates similarity metrics against reference points (average viewer, critic consensus).
7) **Frontend**: A React-based single-page application presents recommendations through multiple interfaces: personalized carousels on the home page, detailed explanations in movie modals, conversational refinement via chat, and self-discovery through the Taste Wrapped visualization.

2) API Design: The frontend communicates with backend services through RESTful APIs. The embedding service accepts user preference data and returns the computed taste vector:

```
POST /api/embeddings/generate
Request: {userId, movies[], vibes[], freeText}
Response: {userId, embedding[3072], timestamp}
```

The recommender service accepts the embedding and returns ranked recommendations with explanations:

```
POST /api/recommendations
Request: {userId, embedding[], numResults,
         excludeMovies[], context}
Response: {carousels[{title, recommendations[
         {movieId, matchScore, explanation,
          factors[]}]}]}
```

This separation enables independent scaling of compute-intensive embedding generation and LLM inference, while the stateless API design supports horizontal scaling of the frontend.

3) Feedback Loop Architecture: A key architectural feature is the real-time feedback loop connecting user reviews to recommendation updates. When a user rates a movie 8/10 or higher, the system automatically incorporates that movie into their taste profile. An exponential moving average (EMA) governs profile updates:

$$\mathbf{e}_{t+1} = \alpha \cdot \mathbf{e}_t + (1 - \alpha) \cdot \mathbf{e}_{\text{new}} \qquad (1)$$

where $\alpha = 0.8$ preserves 80% of existing preferences while incorporating 20% from new signals. This approach prevents recommendation whiplash from single ratings while enabling taste evolution over time.

B. Taste Embedding Generator

The Taste Embedding Generator is responsible for constructing a unified embedding space that represents both movie semantics and user preferences. This component integrates heterogeneous data sources, performs systematic preprocessing, and encodes movies and users into dense embedding vectors suitable for downstream retrieval and recommendation tasks. The generator consists of four major submodules: data preprocessing, embedding model selection, movie embedding generation, and user embedding construction.

1) Data Sources & Preprocessing: To construct unified movie and user embeddings, we integrate a diverse set of datasets covering movie metadata, user ratings, conversational interactions, and external metadata enrichment sources. All raw datasets are downloaded and processed through a dedicated preprocessing pipeline located in the project's `Dataset/` directory. Processed outputs serve as the direct input to the `TasteEmbeddingGenerator` module.

Data Categories: We incorporate five main dataset categories:

- **Movie metadata**: MovieLens [31], MovieTweetings [32], and INSPIRED Movie Database [33]
- **Ratings**: MovieLens [31] and MovieTweetings [32] user–movie interactions
- **Dialogue datasets**: ReDial [34], CCPE [35], and INSPIRED dialogues [33]
- **Metadata enrichment**: TMDB API [37] providing plot overview, genres, cast, keywords, and runtime
- **Emotional dataset**: GoEmotions [36] text dataset for emotion-aware extensions in user modeling

All preprocessing scripts follow a consistent philosophy: (1) extract raw data, (2) normalize schema, (3) perform TMDB enrichment, (4) output clean, standardized tables suitable for downstream embedding.

Movie Metadata Preprocessing: Movie metadata preprocessing unifies heterogeneous sources into a shared schema. Each of the three movie datasets undergoes the following steps:

1) **Schema normalization:** Raw datasets differ in column names and structures. We standardize fields such as:

   ```
   movie_id, title, year, genres, tmdb_id
   ```

   The INSPIRED movie database and MovieTweetings datasets require additional normalization due to inconsistent or missing identifiers.
2) **ID alignment & duplicate handling:** Movies may appear in multiple datasets with different IDs. We treat `tmdb_id` as the canonical identifier and merge all entries belonging to the same film. If `tmdb_id` is missing, we fall back to a composite key `source:movie_id`.
3) **TMDB enrichment:** Using a custom `tmdb_client` python module, we query:
   - TMDB overview text
   - TMDB genres (hierarchical)
   - Top-billed cast
   - Keyword tags
   - Standardized title and release date

   This step expands sparse metadata into rich, natural-language descriptive fields, which dramatically improves embedding quality.
4) **Text cleaning and normalization:** We remove HTML artifacts, collapse whitespace, fix Unicode inconsistencies, and standardize formatting to ensure that the final description strings are embedding-ready.
5) **Output serialization:** Final enriched files are stored under:

   ```
   Dataset/processed/
      movielens_movies_tmdb.csv
      movietweetings_movies_tmdb.csv
      inspired_movie_database_tmdb.csv
   ```

These enriched files become the direct input for the Movie Embedding module.

Dialogue Data Preprocessing: Dialogue datasets capture implicit user preferences expressed through conversation. Due to structural differences across datasets (ReDial, CCPE, INSPIRED), we perform a multi-stage preprocessing pipeline:

1) **Utterance extraction:** For each dataset, we extract all user- or speaker-associated text fields from raw JSON or CSV structures. ReDial requires parsing nested message lists, while CCPE includes speaker-role annotations.
2) **Speaker normalization:** Since datasets do not share a common user identifier structure, we assign consistent pseudo-user IDs:

   ```
   redial:<speaker_id>
   ccpe:<dialog_id>:<speaker>
   inspired:<dialog_id>:<role>
   ```

   This ensures compatibility with downstream user embedding construction.
3) **Conversation flattening:** For each user identifier, we concatenate all utterances belonging to that user into a single text profile:

   $$\texttt{text\_profile} = \texttt{utterance}_1 \| \texttt{utterance}_2 \| \ldots$$

   This aggregated profile forms the basis for text-based user embeddings.
4) **Optional emotion annotation integration:** Processed GoEmotions data is included to enable future emotion-aware user modeling, although it is not used in the v1 system.
5) **Final output files:** Cleaned and flattened dialogue datasets are stored as:

   ```
   Dataset/processed/
      redial_dialogues.csv
      ccpe_dialogues.csv
      inspired_dialogs.csv
   ```

These files are consumed directly by the UserEmbedding module to construct text-based user profiles.

Ratings Data Preprocessing: MovieLens and MovieTweetings ratings are processed to create a unified user–movie interaction table:

1) Standardize fields: `user_id`, `movie_id`, `rating`
2) Filter invalid ratings (e.g., missing movie IDs)
3) Align movie IDs with TMDB-enriched movie tables
4) Serialize final rating files:

```
Dataset/processed/
   movielens_ratings.csv
   movietweetings_ratings.csv
```

The MovieLens ratings file is the primary source for rating-based user embeddings in v1.

Summary of Preprocessed Outputs: The preprocessing pipeline produces more than 2.4M clean, structured records across all datasets, including enriched movie metadata, standardized rating interactions, and flattened dialogue texts. All of these serve as the foundation for constructing high-quality movie and user embeddings in downstream modules.

C. Taste Embedding Generator

The Taste Embedding Generator is an end-to-end pipeline that constructs (1) movie-level semantic embeddings and (2) user-level taste embeddings from heterogeneous data sources. It is implemented as a modular Python package with pluggable encoder backends and clearly separated data processing stages.

Concretely, the generator consists of two main modules: `MovieEmbedding.py` and `UserEmbedding.py`, orchestrated by `Generator.py`. The output of this pipeline is stored as Parquet files and later consumed by the retrieval and recommendation components.

1) Embedding Backend Abstraction: To allow flexible experimentation with different encoders, we define a backend abstraction layer in `embeddings_backend.py`. All embedding models implement a common interface:

```
BaseEmbeddingBackend.embed_texts(texts: List[
    str]) -> List[List[float]]
```

Two concrete backends are used in our experiments:

- **SentenceTransformerBackend**: wraps a local sentence-transformers model (e.g., `BAAI/bge-base-en-v1.5`) and returns L2-normalized embeddings suitable for cosine similarity.
- **OpenAIEmbeddingBackend**: calls the OpenAI API with `text-embedding-3-large` and returns high-dimensional embeddings.

The choice of backend is controlled via `TasteEmbeddingConfig`, which specifies `backend_type`, `model_name`, batch size, and other hyperparameters. Because the downstream code only depends on `BaseEmbeddingBackend`, we can swap backends without changing the movie or user embedding logic.

2) Movie Embedding Generation: The `MovieEmbeddingGenerator` builds a unified table of movie embeddings from TMDB-enriched metadata. It operates in three stages: data loading, movie text construction, and batch encoding.

Data Loading from Multiple Sources: We first load TMDB-enriched movie tables from the processed directory:

```
Dataset/processed/
   movielens_movies_tmdb.csv
   movietweetings_movies_tmdb.csv
   inspired_movie_database_tmdb.csv
```

Each loader (`_load_movies_movielens`, `_load_movies_movietweetings`, `_load_movies_inspired`) attaches a `source` column (`"movielens"`, `"movietweetings"`, or `"inspired"`) and normalizes the ID field to `movie_id`. For example, `movieId` from MovieLens is renamed to `movie_id`, and for INSPIRED we derive `movie_id` from `tmdb_id` when possible, otherwise from the row index.

All source-specific tables are then concatenated into a single DataFrame via `load_all_sources()`, which logs the total number of movie rows.

Natural-Language Movie Description: For each movie row, we construct a rich textual description using `build_movie_text`. This function prioritizes TMDB metadata when available and falls back to source-specific fields. The template includes:

- **Title and year**: from `tmdb_title` or original `title`, and `year` or the year parsed from `tmdb_release_date`.
- **Genres**: from `tmdb_genres`, `genres`, or `genre`.
- **Plot / overview**: primarily `tmdb_overview`; for INSPIRED, we fall back to `long_plot` or `short_plot` if TMDB text is missing.
- **Cast and director**: TMDB top cast (`tmdb_top_cast`) is preferred; INSPIRED provides `actors` and `director` fields.
- **Keywords**: extracted from `tmdb_keywords` when present.

All available fragments are concatenated into a single `embedding_text` string such as:

> *"Title: The Matrix (1999). Genres: Action, Sci-Fi. Director: Lana Wachowski, Lilly Wachowski. Plot: A hacker discovers the truth about his reality and his role in the war against its controllers. Cast: Keanu Reeves, Laurence Fishburne, Carrie-Anne Moss. Keywords: cyberpunk, dystopia, virtual reality."*

This representation makes the encoder directly consume the same type of text an LLM would see in natural descriptions.

Duplicate Resolution via Embedding Keys: Because the same movie can appear in multiple datasets, we ensure that each logical movie corresponds to exactly one embedding vector. `_make_embed_key` defines a canonical key:

- If `tmdb_id` is available, the key is `"tmdb:⟨tmdb_id⟩"`.
- Otherwise, we fall back to `"<source>:<movie_id>"`.

We then drop duplicate keys and only embed the unique set of `embedding_text` values. This avoids redundant API calls and ensures cross-dataset consistency: all rows that share the same `embed_key` will later reference the same embedding vector.

Batch Embedding and Serialization: Let K be the number of unique movies after deduplication. We collect all descriptions into a list $\{t_1, \ldots, t_K\}$ and compute embeddings in mini-batches of size `batch_size`:

$$\mathbf{m}_i = \text{BackendEmbed}(t_i), \quad i = 1, \ldots, K.$$

The backend may be either a local SentenceTransformer or the OpenAI embedding API, but both are accessed through the same `embed_texts` method. The resulting vectors are stored in a dictionary keyed by `embed_key` and then mapped back onto the full movie table.

Before writing to disk, we normalize data types (e.g., cast `year` to a nullable integer type and convert string columns to `pandas.StringDtype`) to ensure Parquet compatibility. The final movie embedding table contains, for each row:

- `movie_id` and `source`
- original and TMDB metadata fields
- `embedding_text`
- `embedding` (list of floats)

and is saved to

```
TasteEmbeddingGenerator/artifacts/movie\
    _embeddings.parquet}.
```

3) User Embedding Generation: The `UserEmbeddingGenerator` builds user-level embeddings from two complementary sources: explicit ratings and conversational text profiles. It reuses the precomputed movie embeddings and shares the same backend abstraction.

Loading Movie Embeddings: We first load the movie embedding artifact and optionally filter by `source`. In our main configuration, we restrict to MovieLens-based movies to match the rating data:

1) Read `movie_embeddings.parquet`.
2) If `source_filter = "movielens"`, keep only those rows.
3) Build a dictionary $\mathcal{M}$ mapping `movie_id` $\rightarrow$ $\mathbf{m}$ (NumPy vectors).

This mapping is then used to convert user–movie interactions into continuous preference vectors.

Rating-Based User Embeddings: Given MovieLens ratings, we construct rating-based user embeddings as follows:

1) Load `movielens_ratings.csv`.
2) Filter to “liked” movies satisfying rating $\geq$ `rating_threshold` (default 4.0).
3) Group by `userId` and collect the list of liked `movieIds`.
4) Look up each `movieId` in $\mathcal{M}$ to obtain its embedding vector; users with fewer than `min_movies` liked movies (default 3) are discarded.
5) For the remaining users, compute the mean of their liked movie vectors:

$$\mathbf{u}^{(u)}_{\text{rating}} = \frac{1}{N_u} \sum_{i=1}^{N_u} \mathbf{m}_i,$$

where N_u is the number of movies contributing to user u’s profile. We store $\mathbf{u}^{(u)}_{\text{rating}}$ along with `user_id` (stringified `userId`) and `num_movies` (i.e., N_u).

Dialogue-Based User Embeddings: In parallel, we build text-based user embeddings from CCPE, INSPIRED, and ReDial:

1) For each dataset, we group utterances by a dataset-specific speaker key and concatenate all texts into a single `text_profile`:
   - CCPE: group by `dialog_id`;
   - INSPIRED: group by `(dialog_id, speaker)`;
   - ReDial: group by `speaker_id`.
2) Each group is assigned a pseudo-user ID in a disjoint namespace, for example:
   - `ccpe:<dialog_id>`
   - `inspired:<dialog_id>:<speaker>`
   - `redial:<speaker_id>`
3) We drop empty profiles and pass the list of `text_profile` strings to the same `embed_texts` backend used for movies, obtaining vectors $\mathbf{u}^{(u)}_{\text{text}}$.

This step yields a table of `user_id` and `embedding_text`, capturing users’ conversational behavior and explicitly stated preferences (e.g., “I love slow-burn psychological thrillers”).

Hybrid Fusion with α-Mixing: To obtain a unified user representation, we merge the rating-based and text-based tables using an outer join on `user_id`. Depending on data availability, three cases arise:

- Both $\mathbf{u}_{\text{rating}}$ and $\mathbf{u}_{\text{text}}$ exist and have the same dimension.
- Only one of the two embeddings exists.
- Neither exists (the user is dropped).

When both vectors are present and aligned in dimension, we compute a weighted combination:

$$\mathbf{u}_{\text{final}} = \alpha\, \mathbf{u}_{\text{rating}} + (1-\alpha)\, \mathbf{u}_{\text{text}},$$

with default $\alpha = 0.7$, giving slightly more weight to explicit ratings while still incorporating conversational nuance. If only one vector is available, we use it directly as $\mathbf{u}_{\text{final}}$.

The final user embedding table contains, for each user ID:

- `embedding`: fused user taste vector
- `embedding_rating`: rating-only vector (optional)
- `embedding_text`: dialogue-only vector (optional)
- `num_movies`: number of liked movies contributing to the rating-based component

and is serialized as

```
TasteEmbeddingGenerator/artifacts/user\
    _embeddings.parquet
```

Role within the Overall System: These movie and user embeddings form the backbone of the downstream retrieval and recommendation stack. Movie embeddings support efficient nearest-neighbor search and genre-aware similarity queries, while user embeddings provide a compact, updatable representation of individual taste. Because both live in the same semantic space, we can compute user–movie affinity via cosine similarity and feed top candidates into the LLM-based re-ranking and explanation module described in the later sections.

D. Conversational Recommender

Our conversational movie recommender system integrates dense vector retrieval, long-term preference modeling, and LLM-based ranking in a unified Retrieval–Augmentation–Generation (RAG) pipeline. The system is designed to support multi-turn conversational queries, dynamic user modeling, and interpretable recommendations.

1) Candidate Retrieval Layer:

- After obtaining the embeddings of our combined dataset of movies, we have matrix $M \in \mathbb{R}^{N \times d}$ where N is the number of movies. Then, we take the user-input and embed them in the same way after running it through an optional GPT Taste Normalizer Step (simplifies user statements to a standardized, simple description of their taste). After the dataset and the user taste are both embedded, they are in the same semantic space and can be compared.
- These embeddings are indexed with **FAISS IndexFlatIP**, an exact inner-product search structure implemented in high-performance C++ with SIMD acceleration.
- For a given user query, the system computes an embedding $\mathbf{e}_t$ and retrieves:

$$\text{TopK} = \text{FAISS.search}(\mathbf{e}_t, k = 20).$$

- Because vectors are L2-normalized, inner-product similarity is equivalent to cosine similarity, enabling stable semantic retrieval.

After the embedding retrieval step, here is a visualization of the user's taste vector, 5 of the top recommendations (according to the embeddings only), plotted against the local neighborhood of movies closest to the user taste.

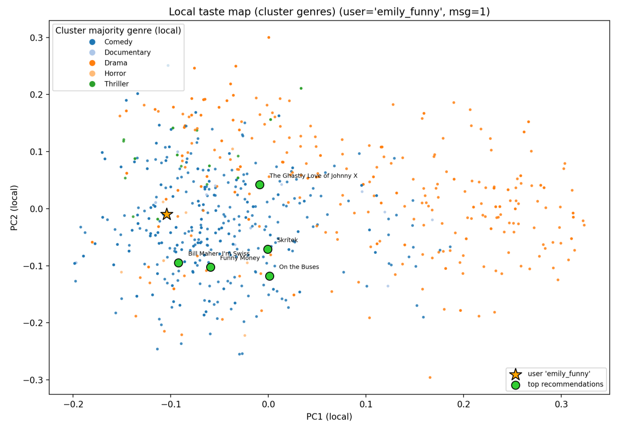


Fig. 2. This plots the main genre of each movie, and the relevant query to this image is "I like funny movies". As seen, this user taste vector and selected recommendations live in a neighborhood of primarily comedy movies.

2) Conversational Memory and Taste Fusion: To maintain continuity over multiple conversational turns, we update a per-user taste vector using an exponential moving average (EMA). The update is: $u_t = \alpha u_{t-1} + (1 - \alpha) e_t$, where e_t is the embedding of the current message and $\alpha = 0.8$ controls memory persistence.

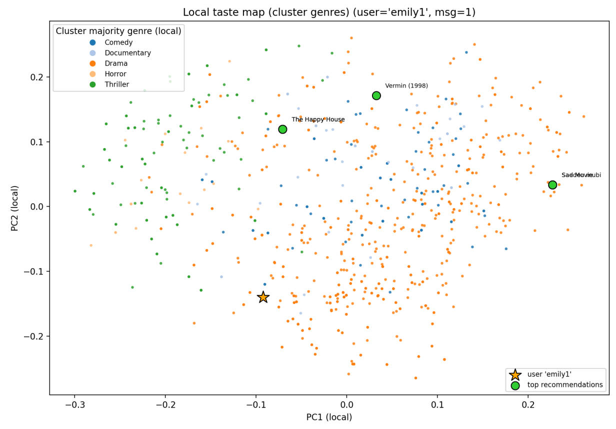


Fig. 3. In comparison, here is a user whose conversation is based in "I like sad movies", and their taste vector and recommendations are surrounded by a neighborhood of drama movies, with a handful of thrillers.

Key characteristics:

- EMA prevents abrupt shifts in recommendations when users temporarily deviate from long-term preferences.
- The fused vector $\mathbf{u}_t$ represents the system's internal estimate of the user's latent taste distribution.
- We persist per-user vectors and per-turn logs (input text, retrieved indices, fused embeddings) for offline evaluation and reproducibility.

3) LLM-as-Ranker Architecture: The second-stage ranker uses a fine-tuned GPT-4o-mini model to rescore retrieved candidates. The model receives:

- the fused preference vector $\mathbf{u}_t$ (converted into a textual summary),
- the last 5–10 conversational turns or any other information about the user's preferences are available in text form,
- metadata for each retrieved movie (genres, plot summary, cast, year),
- and the raw retrieval scores.

Instead of simply selecting movies, the LLM learns a preference function:

$$f_\theta(\text{user history}, m_i, \text{metadata}) \rightarrow \text{rank score}.$$

The workflow is:

$$\text{Top-20 candidates} \rightarrow \text{LLM reranker} \rightarrow \text{Top-5 final recommendations}$$

This two-stage pipeline captures both semantic similarity (from embeddings) and fine-grained taste cues (from the LLM).

4) Real-Time Feedback Integration: Every user interaction updates:

- the taste vector through new embeddings,
- the preference history used by the LLM,

- and the persistent log for future offline training or reinforcement learning.

This enables an adaptive, memory-aware conversational experience.

E. GPT-4o-mini Finetuning for Personalized Conversation

We fine-tuned GPT-4o-mini to act as a personalized preference estimator and reranker using a curated dataset modeled on Letterboxd-style user histories.

1) Training Dataset Construction: The training dataset was collected using the Letterboxd (obtained via user request), using a capped DFS starting from a few known users (professional movie reviewer David Sims), going through their main friends list for users over 100 followers, and then obtaining their review text, rating history, and user favorites.

Each JSONL instance contained:

- A structured user profile: favorite films, rated films, review excerpts.
- A target movie candidate with metadata (masked from the user profile).
- A supervised label: the rating (1–5) the user would likely give that candidate.

This trains the model to approximate the latent scoring function:

$$\hat{y} = f_\theta(\text{user profile}, \text{movie metadata}).$$

2) Fine-tuning Procedure:

- Data were uploaded to the OpenAI fine-tuning endpoint in chat-completion format.
- OpenAI infrastructure handled batching, gradient updates, checkpointing, and deployment.
- The resulting model ID (e.g., `ft:gpt-4o-mini:movie-pref-ranker`) was integrated directly into our Python backend.

3) Resulting Model Capabilities: The fine-tuned model learned to:

- infer implicit preferences (e.g., liking moody slow-burn sci-fi),
- differentiate stylistically similar movies,
- perform soft preference reasoning (e.g., user dislikes overly violent films"),
- generate explanations grounded in the user's history.

This yields more nuanced reranking and far more personalized conversational responses.

F. Conversational Interface and Visualization

The frontend layer transforms backend outputs into an intuitive user experience that emphasizes transparency and interactivity. Built with React and TypeScript, the interface comprises four primary components: user onboarding, recommendation display, conversational chat, and taste visualization.

1) User Onboarding Flow: New users complete a three-step onboarding process designed to elicit sufficient preference signals for initial embedding generation while minimizing friction:

Step 1: Movie Selection. Users search for and select 3–5 movies they have recently watched and enjoyed. The interface provides real-time search against the TMDB database with poster thumbnails for visual confirmation. This step captures explicit positive signals with high confidence.

Step 2: Vibe Selection. Users choose from a curated set of "vibe" tags (e.g., "mind-bending," "feel-good," "visually stunning," "slow burn") that describe their preferred viewing experiences. These tags map to semantic descriptors that influence embedding generation, capturing preferences that transcend individual movies.

Step 3: Free-Text Description. An optional text field allows users to describe their ideal movie experience in natural language (e.g., "I love films that make me think for days afterward"). This unstructured input is embedded directly and fused with the structured signals.

For users with existing TMDB accounts, OAuth integration bypasses onboarding by importing their rating history and favorites. The system requires a minimum of five rated movies to generate personalized recommendations; users below this threshold complete the standard onboarding flow.

Upon completion, a "Personalizing" interstitial displays the recommendation pipeline in real-time, showing each processing stage: loading user profile, fetching movie metadata, computing taste embedding, performing semantic search, and generating explanations. This transparency reinforces user understanding of the system's operation.

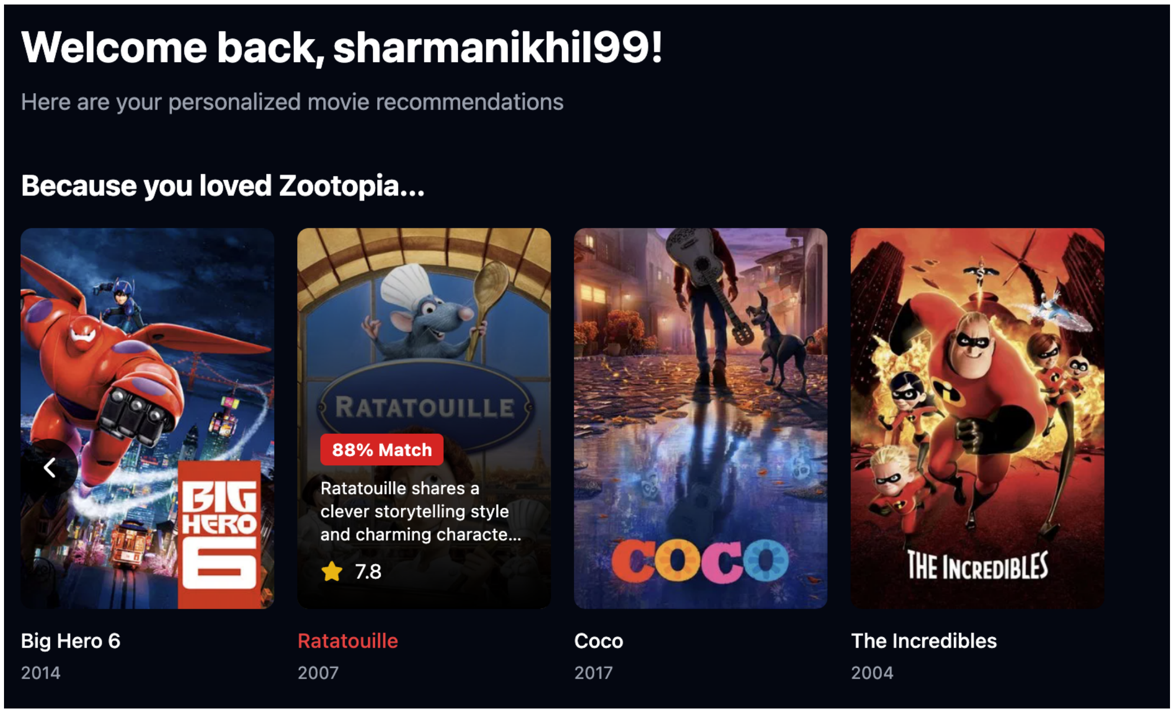
2) Recommendation Display: The home page presents recommendations through horizontally-scrolling carousels, each titled with an explanation referencing user preferences:

- "Because you loved *Inception*..."
- "For fans of mind-bending narratives..."
- "Based on your appreciation for Christopher Nolan..."

Each movie card displays the poster, title, release year, and a match score (0–100%) indicating alignment with the user's taste profile. Clicking a card opens a detailed modal containing:

- **Match Score**: A prominent percentage with visual indicator
- **Explanation**: 2–3 sentences explaining why this movie matches the user's preferences, referencing specific films they enjoyed
- **Match Factors**: Categorical tags (e.g., "director style," "narrative complexity," "emotional depth") highlighting the dimensions of similarity
- **Movie Metadata**: Genre, runtime, cast, and plot synopsis from TMDB
- **Action Buttons**: Options to add to watchlist, mark as watched, or initiate a conversation about the recommendation

The "Why We Recommended This" section directly addresses the transparency goal, transforming the recommenda-


Welcome back, sharmanikhil99!

Here are your personalized movie recommendations

Because you loved Zootopia...

88% Match

Ratatouille shares a clever storytelling style and charming characte...

7.8

Fig. 4. Home page displaying personalized recommendation carousels with "Because you loved..." explanations.

tion from an opaque suggestion into an explained reasoning chain.

3) Conversational Chat Interface: The chat interface enables users to refine recommendations through natural language dialogue. The interface maintains conversational context, allowing multi-turn interactions such as:

> **User**: "I liked Inception but thought it was too long. What else might I enjoy?"
> **ReelReason**: "Based on your preference for mind-bending concepts but desire for tighter pacing, I'd recommend *Coherence* (2013). It delivers reality-bending tension in just 89 minutes, filmed in a single location with increasingly paranoid atmosphere..."

The chat component integrates with the recommendation backend through a context-aware prompt that includes:

1) The user's taste embedding and top preferences
2) Current conversation history
3) The movie being discussed (if initiated from a modal)
4) Retrieved candidate movies from semantic search

This architecture enables the LLM to ground its responses in both the user's demonstrated preferences and the available movie corpus, reducing hallucination while maintaining conversational fluency. Users can ask follow-up questions, request alternatives, specify constraints ("something shorter," "nothing too scary"), or explore tangential interests.

4) Taste Wrapped Visualization: Inspired by Spotify's annual "Wrapped" feature, the Taste Wrapped page provides users with an interactive visualization of their cinematic preferences. This feature addresses the transparency and self-discovery goals by making the abstract taste embedding concrete and explorable.

Taste Map (2D Projection). The primary visualization renders the user's position in taste space as a 2D scatter plot. User and movie embeddings are projected from 3072 dimensions to 2D using UMAP (Uniform Manifold Approximation and Projection), which preserves local neighborhood structure. The resulting plot positions the user among reference points:

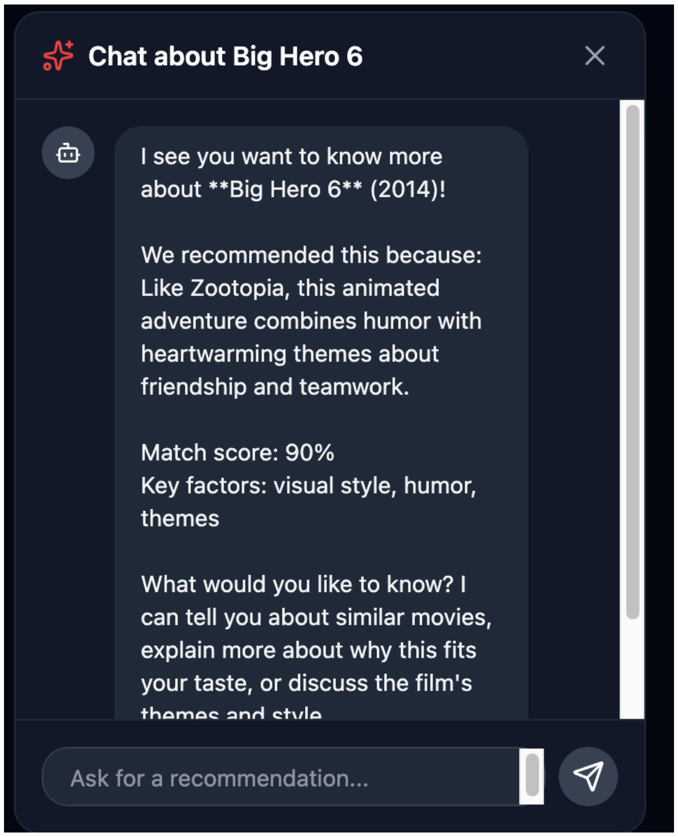
- *You*: The user's current taste position (highlighted)
- *Friends*: Positions of connected users (if social features enabled)
- *Average Viewer*: Centroid of popular movie embeddings
- *Critic Consensus*: Centroid of critically acclaimed films

Axis labels are derived from cluster analysis of the embedding space, typically representing interpretable dimensions such as "Mainstream $\leftrightarrow$ Art House" and "Light $\leftrightarrow$ Dark." Users can click any reference point to compare their preferences.

Genre Radar Chart. A spider chart displays the user's affinity across eight genre dimensions (Action, Comedy, Drama, Sci-Fi, Thriller, Romance, Horror, Animation). Affinity scores are computed as rating-weighted genre frequencies:

$$\text{Affinity}_g = \frac{\sum_{m \in M_u} r_m \cdot \mathbb{1}[g \in \text{genres}(m)]}{\sum_{m \in M_u} \mathbb{1}[g \in \text{genres}(m)]} \tag{2}$$

where M_u is the user's rated movies, r_m is the rating, and $\mathbb{1}[\cdot]$ is the indicator function. Clicking a reference point on the Taste Map overlays that profile's genre distribution for comparison.



Chat about Big Hero 6

I see you want to know more about **Big Hero 6** (2014)!

We recommended this because: Like Zootopia, this animated adventure combines humor with heartwarming themes about friendship and teamwork.

Match score: 90%
Key factors: visual style, humor, themes

What would you like to know? I can tell you about similar movies, explain more about why this fits your taste, or discuss the film's themes and style

Ask for a recommendation...

Fig. 5. Conversational Chat interface.

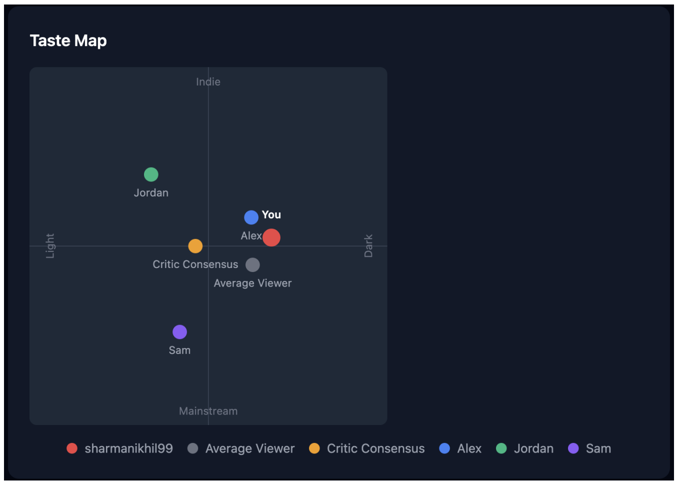


Fig. 6. Taste graph: comparing your genre preferences to other users and critics.

5) Implementation Details: The frontend is implemented as a single-page application using React 18 with TypeScript for type safety. State management leverages React Context for authentication and localStorage for preference persistence across sessions. The UI framework combines Tailwind CSS for utility-based styling with custom components for specialized

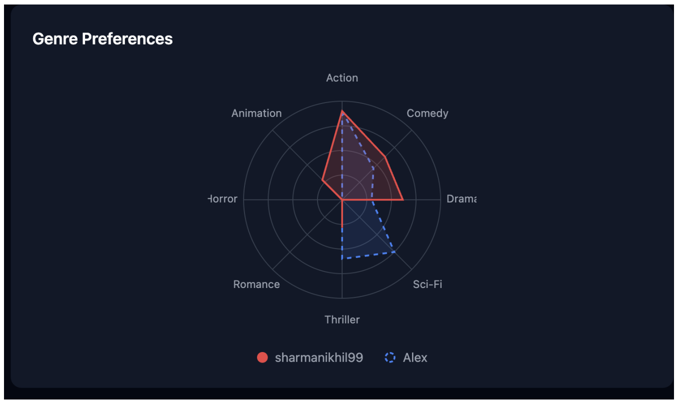


Fig. 7. Genre radar.

visualizations (radar charts, scatter plots). API communication uses the native Fetch API with error boundaries for graceful degradation.

G. Embedding Quality Evaluation

We evaluate the quality of the movie and user embeddings produced by the Taste Embedding Generator using two quantitative metrics: (1) Genre Separation Gap and (2) HitRate@10 under a random-retrieval benchmark. These metrics assess the semantic coherence of the embedding space and its suitability for downstream recommendation.

Genre Separation Gap: Because different embedding backends (e.g., BGE-base vs. OpenAI) operate on different vector scales, we measure the *relative* difference between same-genre and different-genre movie distances:

$$\begin{aligned}\Delta_{\text{genre}} &= \mathbb{E}\big[d(\mathbf{m}_i, \mathbf{m}_j) \,\big|\, \text{same genre}\big] \\ &\quad - \mathbb{E}\big[d(\mathbf{m}_i, \mathbf{m}_j) \,\big|\, \text{different genre}\big]. \end{aligned} \tag{3}$$

where $d(\cdot,\cdot)$ is cosine distance. Since movies often have multi-label genres (e.g., "Drama—Romance"), absolute separation is inherently small, but relative differences between embedding backends are highly indicative.

OpenAI's `text-embedding-3-large` achieves consistently larger separation gaps than the BGE-base encoder, indicating stronger semantic clustering in genre space.

HitRate@10 (Recommendation Capability): We evaluate recommendation performance using a random candidate retrieval benchmark. For each user embedding $\mathbf{u}$, all movies are scored by cosine similarity:

$$s(m, u) = \cos(\mathbf{m}, \mathbf{u}).$$

A "hit" occurs when at least one true liked movie (rating ≥ 4.0) appears in the top-10 ranked movies. HitRate@10 is defined as:

$$\text{HitRate@10} = \frac{1}{|U|} \sum_{u \in U} \mathbb{1}\left[\exists m \in \mathcal{L}(u) : m \in \text{Top10}(u)\right].$$

Results: OpenAI embeddings deliver more than $5\times$ higher HitRate than the baseline SentenceTransformer model. We attribute this to:

- larger representational capacity,
- improved alignment between movie and user textual semantics,
- richer contextualization from high-quality training data.

Metric	BGE-base (v1)	OpenAI (v2)
Embedding Dimensionality	768	3072
Genre Separation Gap (*Drama)	0.0151	0.0256
HitRate@10	0.030	0.164

TABLE I
QUANTITATIVE COMPARISON OF EMBEDDING BACKENDS ACROSS SEPARATION QUALITY AND RECOMMENDATION ACCURACY.

Interpretation: Across both metrics, OpenAI's `text-embedding-3-large` encoder demonstrates substantially better embedding structure. The Genre Separation Gap is noticeably larger despite the multi-label noise inherent in movie genres, indicating stronger semantic neighborhood clustering.

Likewise, the HitRate@10 improves from 0.030 to 0.164 (over $5\times$), showing that the OpenAI embedding space aligns much more effectively with user preferences. This improvement is consistent with its higher dimensionality and richer textual representation learned from large-scale training data.

The evaluation demonstrates that the OpenAI backend produces significantly more structured embedding spaces and more reliable user–movie similarity estimates. These embeddings form a strong foundation for the LLM-based retrieval and re-ranking system presented in subsequent sections.

H. Recommendation Performance

1) Embedding Alignment Evaluation: These are the evaluations that we did, which informed several key decisions throughout the project. First, we measured recommendation quality of the final 5 using cosine similarity of movie embeddings, as well as a 4o based autoeval and in this case we were thinking of whether the OpenAI or HuggingFace embeddings were better in practice - in theory, the OpenAI embeddings has larger dimensions. By looking at the cosine similarity of the final 5 selected, you can see that OpenAI's recommendations were closer, and to validate, the autoeval rated the basic chatbot functionality, and as you can see, the OpenAI embeddings were found more satisfactory on multiple of our custom qualitative axes.

Embedding Backend	Mean Cos Sim	Std Dev	Pairs
bge-base	0.781	0.084	1250
OpenAI	0.796	0.081	1250

TABLE II
COSINE SIMILARITY ALIGNMENT ACROSS EMBEDDING BACKENDS. HIGHER IS BETTER.

This shows that the cosine similarity with OpenAI embeddings is better, since it is higher.

We also designed a GPT-as-a-judge autoevaluation system across five custom qualitative axes, as a signal for the overall performance of the system based on the user's experience. This doesn't rate the full UI/UX experience, but rather mostly conversation- and recommendation-based qualities. The axes are:

- Relevance - how well the recommended movies match the user's explicit stated preferences in the conversation, such as genre alignment, thematic fit, etc.
- Diversity - how varied the final recommendations are across genre, tone, decade, style, audience, etc. For instance, how redundant are the top recommendations?
- Personalization - how well the system captures the user's long-term tastes, such as from a few messages ago, or from old preferences and reviews
- Explanation Quality - the clarity, accuracy, and persuasiveness of the LLM-generated justification for each movie. For instance, how specific or generic are the explanations on why a movie is being recommended? Are descriptions of movies accurate?
- Overall Satisfaction - this is a qualitative amalgamation of all the previous axes, on how satisfied a user is overall with the system

While GPT-as-a-judge systems can be prone to occasional hallucination or grading on different standards than human evaluators otherwise might have, this served as a fast and simple signal in place of a human evaluator system for the purposes of relative comparison as we improved our system.

These were run by offering GPT-4o various Letterboxd user profiles and preferences, then rating discussion and recommendation quality, and taking an average. This was to compare the impact of embeddings, and were done using the pre-fine-tuned GPT-4o-mini system for discussion and recommendation.

Metric	bge-base	OpenAI	Delta
Relevance	3.99	4.10	+0.11
Diversity	3.91	4.05	+0.14
Personalization	3.95	3.95	+0.00
Explanation Quality	4.43	4.45	+0.02
Overall Satisfaction	4.00	4.05	+0.05

TABLE III
GPT-AS-A-JUDGE AUTOEVALUATION COMPARING EMBEDDING BACKENDS.

The improvement across the openai column for autoevaluation ratings shows that the embeddings quality difference does make an impact on overall recommendation, discussion, and system performance.

2) Fine-Tuning Improvement in performance: Then, during the fine-tuning process, we also wanted to know whether the personalization of the GPT-4o-mini model was improving recommendations and performance. These are all using the OpenAI embeddings.

Model Version	Mean Cos Sim	Std Dev	Pairs
Baseline (4o-mini)	0.742	0.091	1250
Fine-tuned Model	0.768	0.086	1250

TABLE IV
IMPACT OF FINE-TUNING ON SEMANTIC ALIGNMENT BETWEEN USER AND MOVIE EMBEDDINGS.

This table shows a +0.026 increase in semantic alignment as a result of finetuning the LLM, showing the impact of fine-tuning on the final LLM-based reranking step.

Then, to understand the impact of fine-tuning on overall recommendation and system performance in a holistic manner, we utilized the same GPT-as-a-judge autoevaluation method described in the previous section.

Metric	Baseline	FT 4o-mini	Delta
Relevance	4.10	4.35	+0.25
Diversity	4.05	4.10	+0.05
Personalization	3.95	4.25	+0.30
Explanation Quality	4.45	4.60	+0.15
Overall Satisfaction	4.05	4.32	+0.27

TABLE V
GPT-AS-A-JUDGE EVALUATION SHOWING IMPROVEMENTS FROM FINE-TUNING.

The largest increases were observed in personalization, relevance, and overall satisfaction, while the diversity held relatively stable. Overall, this autoevaluation indicates the improvement that fine-tuning GPT-4o-model brought to the system.

IV. LIMITATIONS & FUTURE WORK

While ReelReason demonstrates the viability of LLM-enhanced explainable recommendations, several limitations constrain the current system. This section provides an honest assessment of these constraints, analyzes common failure modes, and proposes directions for future development.

A. Current Limitations

1) Data Availability Constraints: The system's recommendation quality is fundamentally bounded by the underlying data sources. Our movie corpus of approximately 60,000 films from MovieLens, while substantial, underrepresents several categories: international cinema beyond Hollywood and major European productions, independent films with limited theatrical release, and recent releases from the past 12 months. This coverage gap means users with preferences for world cinema or emerging filmmakers may receive less relevant recommendations.

Additionally, the conversational datasets used for training (ReDial, CCPE, INSPIRED) contain English-language dialogues predominantly reflecting Western viewing preferences. Users from different cultural contexts may find the system's understanding of their preferences less nuanced.

2) Cold-Start Challenges: Despite leveraging LLM pre-trained knowledge to mitigate cold-start issues, the system still struggles with genuinely new users who provide minimal preference signals. Our onboarding flow requires users to identify at least 3–5 movies, but users unfamiliar with film titles or those with highly niche tastes may find this barrier frustrating. The free-text vibe description helps, but natural language preference elicitation remains imprecise compared to explicit rating histories.

3) Computational Costs: The two-stage architecture imposes non-trivial computational requirements:

- **Embedding Generation**: Computing user taste embeddings via OpenAI's text-embedding-3-large API incurs per-token costs and latency (typically 200–400ms per request). For real-time conversational updates, this latency accumulates across turns.
- **LLM Inference**: The fine-tuned GPT-4o-mini re-ranker requires approximately 1–2 seconds per recommendation request, which is acceptable for initial page loads but introduces noticeable delay for rapid preference refinement in chat.
- **Storage**: Maintaining 3072-dimensional embeddings for 60,000 movies requires approximately 700MB of vector storage, with additional overhead for indexing structures that enable efficient approximate nearest neighbor search.

These costs, while manageable for a demonstration system, would require optimization for production deployment at scale.

4) Evaluation Methodology Limitations: Evaluating recommendation quality presents inherent challenges. Our quantitative metrics (Hit Rate@10, genre separation gap) measure retrieval performance but do not fully capture subjective satisfaction. The LLM-as-a-Judge framework provides scalable qualitative assessment but may not align perfectly with human preferences, particularly for edge cases involving cultural context or personal associations that the judge model cannot access.

User studies, while valuable, were limited in scale (13 participants) and duration. Longitudinal studies tracking recommendation satisfaction over weeks or months would better assess whether the system maintains relevance as user preferences evolve.

B. Error Analysis

We conducted qualitative analysis of recommendation failures to identify systematic error patterns. Figure 9 illustrates a representative failure case.

1) Tone and Maturity Mismatch: The most significant failure mode involves recommendations that match genre but diverge in tone or maturity level. Consider a user who highly rated *Hancock* (2008), an R-rated superhero film featuring dark humor, adult themes, and an antihero protagonist. The system's "Because you loved Hancock..." carousel recommended:

- *Deadpool* (2016) – R-rated, irreverent superhero film ✓
- *Kick-Ass* (2010) – R-rated, subversive superhero film ✓
- *Chronicle* (2012) – PG-13, grounded superhero drama ✓
- *The Incredibles* (2004) – G-rated, family animation 55

As shown in Figure 8, all four films cluster near the user's position in the 2D embedding projection, indicating high semantic similarity along the "superhero" dimension. However, *The Incredibles*, while thematically similar (superheroes, action, family dynamics), targets a fundamentally different audience. A user seeking adult-oriented superhero content would find this recommendation jarring.

This failure reveals a limitation of pure embedding similarity: the 3072-dimensional space captures semantic content effectively but does not explicitly encode content rating, target demographic, or tonal register. Movies can be "about the same thing" while being "for different audiences."

2) Explanation Inconsistency: In approximately 8% of sampled recommendations, generated explanations referenced factors not clearly supported by the user's preference history. For example, an explanation citing "your appreciation for French New Wave cinematography" for a user who had only rated mainstream Hollywood films. These hallucinations, while plausible-sounding, undermine trust when users recognize the mismatch.

Such errors typically occurred when the LLM attempted to generate sophisticated explanations for recommendations that were primarily driven by collaborative signals (similar users liked both films) rather than content similarity. The system lacks a mechanism to distinguish these cases and adjust explanation style accordingly.

3) Popularity Bias: Despite efforts to promote diversity, the system exhibits residual popularity bias. Well-known films with extensive metadata and numerous user ratings produce richer embeddings and appear more frequently in recommendation sets. Obscure films that might perfectly match a user's niche preferences are underrepresented because their embeddings, derived from sparse metadata, cluster less distinctly in the vector space.

C. Future Directions

Based on our analysis of system performance and limitations, we identify several promising directions for future development.

1) Enhanced Movie Database: Expanding the movie corpus to include broader international coverage, independent films, and more timely updates for new releases would improve recommendation relevance for diverse user populations. Integration with additional data sources such as Letterboxd, IMDb, and regional streaming catalogs could provide richer metadata and cross-cultural preference signals.

2) Fine-Tuned Embedding Model: The current system uses general-purpose embedding models (OpenAI text-embedding-3-large) that were not optimized for recommendation tasks. Fine-tuning an embedding model specifically for end-to-end recommendation alignment, where the training objective directly optimizes for predicting user preferences rather than general semantic similarity, could yield significant quality improvements. Contrastive learning approaches that push preferred movies closer to user embeddings while repelling disliked movies offer a promising training paradigm.

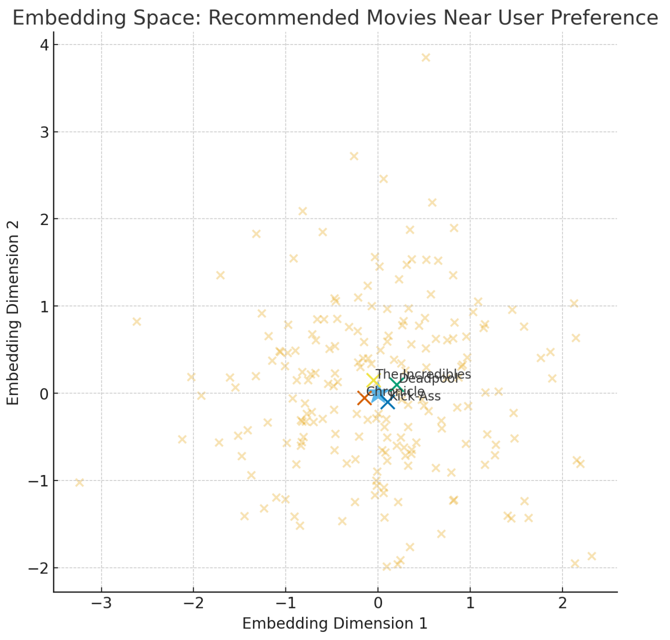


Fig. 8. Embedding space visualization showing recommended movies clustered near user preference. While all recommendations share semantic similarity (superhero genre), *The Incredibles* represents a tone mismatch. The embedding captures "superhero" but not "adult-oriented content."

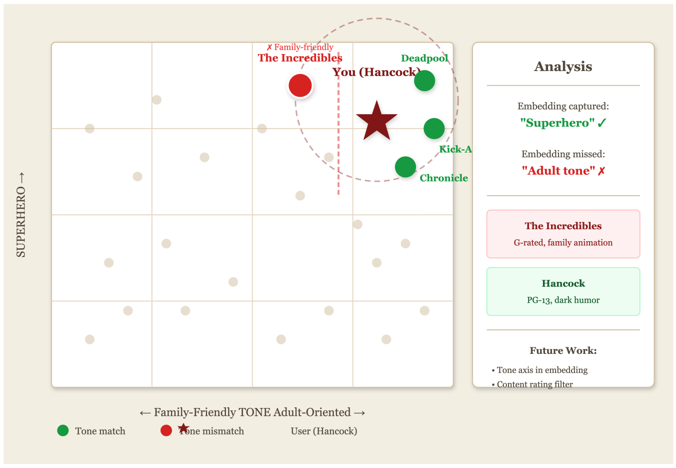


Fig. 9. Error analysis illustrating the tone mismatch problem. The embedding successfully captured genre similarity but failed to separate family-friendly content (*The Incredibles*, G-rated) from adult-oriented content (*Hancock*, PG-13 with dark themes). Future work will incorporate tone as an explicit filtering dimension.

3) Tone and Maturity Modeling: Addressing the tone mismatch problem identified in our error analysis requires explicit modeling of content characteristics beyond genre. We propose two complementary approaches:

1) **Multi-Axis Filtering**: Augment the embedding space with explicit dimensions for content rating (G/PG/PG-13/R), tone (comedic/dramatic/dark), and target demographic (family/teen/adult). These dimensions could be

derived from metadata or predicted by a classifier, then used as hard constraints during candidate retrieval.

2) **Tone-Aware Embeddings**: Train or fine-tune embeddings that explicitly separate content along tonal dimensions, ensuring that family films and adult films occupy distinct regions even when thematically similar.

4) Multi-Modal Representations: Current embeddings derive exclusively from textual metadata (titles, plots, reviews). Incorporating visual features from movie posters and trailers could capture stylistic elements, such as cinematography, color palette, visual tone that text descriptions miss. Audio features from trailers or soundtracks might similarly encode mood and energy level. Multi-modal embeddings fusing text, image, and audio representations offer a path toward more holistic taste modeling.

V. Conclusion

This paper presented ReelReason, an LLM-driven conversational movie recommendation platform that addresses the fundamental transparency gap in modern streaming recommendations. By combining semantic taste embeddings with LLM-based reasoning, our system delivers personalized suggestions accompanied by human-interpretable explanations that help users understand *why* specific movies match their preferences.

Our contributions span the full recommendation pipeline: a unified taste embedding framework that fuses structured ratings with unstructured conversational signals, an LLM-as-ranker architecture that achieves 5$\times$ improvement in Hit Rate@10 over embedding-only baselines, chain-of-thought explanation generation that produces contextual "Because you loved..." rationales, and the Taste Wrapped visualization that enables users to explore their cinematic preferences through interactive 2D projections and AI-generated narrative summaries.

Evaluation across quantitative metrics and user studies demonstrates that explainability serves as a key differentiator: 92% of participants found the natural language explanations helpful for decision-making, and users reported increased trust compared to opaque alternatives. These findings suggest that the transparency benefits of LLM-enhanced recommendations extend beyond novelty to provide genuine utility.

While limitations remain, particularly around tone matching, computational costs, and evaluation methodology, the ReelReason system demonstrates that combining semantic retrieval with generative AI reasoning represents a promising direction for building recommendation systems that are simultaneously accurate, explainable, and engaging. As users increasingly expect AI systems to justify their suggestions, approaches that integrate explanation generation directly into the recommendation process will become essential for maintaining user trust and satisfaction.

Acknowledgment

We thank Professor Dube and Professor Wang and the teaching staff of COMS 6998 for their guidance throughout this project. We also acknowledge the open-source communities behind the datasets (MovieLens, TMDB, ReDial, CCPE, INSPIRED) and tools (OpenAI API, FAISS, React) that made this work possible.

Github Code Repository

Our implementation is stored on Github

References

[1] F. M. Harper and J. A. Konstan, "The MovieLens Datasets: History and Context," *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 2015.

[2] Y. Koren, R. Bell, and C. Volinsky, "Matrix Factorization Techniques for Recommender Systems," *IEEE Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[3] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural Collaborative Filtering," in *Proceedings of the 26th International Conference on World Wide Web (WWW)*, 2017.

[4] Q. Liu, W. Zhang, C. Xu, and X. He, "LLM-Enhanced Recommender Systems (LLMERS): A Survey," *arXiv preprint arXiv:2412.13432*, 2024.

[5] H. Gao, Y. Wang, R. Zhang, and T.-S. Chua, "LLM4Rec: Large Language Models for Recommendation," *arXiv preprint arXiv:2401.08350*, 2024.

[6] S. Luo, B. He, H. Zhao, W. Shao, Y. Qi, Y. Huang, A. Zhou, Y. Yao, Z. Li, Y. Xiao, M. Zhan, and L. Song, "RecRanker: Instruction-Tuning Large Language Model as Ranker for Top-k Recommendation," *arXiv preprint arXiv:2312.16018*, 2023.

[7] L. Chen, C. Gao, X. Du, H. Luo, D. Jin, Y. Li, and M. Wang, "Enhancing ID-based Recommendation with Large Language Models," *arXiv preprint arXiv:2411.02041*, 2024.

[8] Y. Gao, T. Sheng, Y. Xiang, Y. Xiong, H. Wang, and J. Zhang, "Chat-REC: Towards Interactive and Explainable LLMs-Augmented Recommender System," *arXiv preprint arXiv:2303.14524*, 2023.

[9] X. Liang, Y. Chen, Y. Tang, and X. Li, "LLM-REDIAL: A Large-Scale Dataset for Conversational Recommender Systems," in *Findings of the Association for Computational Linguistics (ACL Findings)*, 2024.

[10] R. Li, S. E. Kahou, H. Schulz, V. Michalski, L. Charlin, and C. Pal, "ReDial: Recommendation Dialogues Dataset," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[11] F. Radlinski, K. Balog, B. Byrne, and K. Krishnamoorthi, "Coached Conversational Preference Elicitation: A Case Study in Understanding Movie Preferences," in *Proceedings of the 20th SIGDIAL*, 2019.

[12] Y. Jiang, Y. Wang, and W. Xin Zhao, "Beyond Utility: Evaluating LLM-based Recommenders," *arXiv preprint arXiv:2411.00331*, 2024.

[13] A. Said, K. Verbert, and F. Ricci, "On Explaining Recommendations with Large Language Models," *Frontiers in Big Data*, 2025.

[14] Spotify Research, "Contextualized Recommendations Through Personalized Narratives Using LLMs," 2024.

[15] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chen, Q. Le, D. Zhou, and E. Chi, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[16] H. Huang, X. Bu, H. Zhou, Y. Qu, J. Liu, M. Yang, B. Xu, and T. Zhao, "An Empirical Study of LLM-as-a-Judge for LLM Evaluation: Fine-tuned Judge Model is not a General Substitute for GPT-4," *arXiv preprint arXiv:2403.02839*, 2024.

[17] The Movie Database (TMDB), "TMDB API: The Movie Database Developer Platform," Available at: https://developer.themoviedb.org/docs/getting-started, Accessed: 2025.

[18] D. Demszky, D. Movshovitz-Attias, S. Ko, A. Cowen, G. Nemade, and S. Agrawal, "GoEmotions: A Dataset of Fine-Grained Emotions," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

[19] T. Dooms, "MovieTweetings: A Movie Rating Dataset Collected From Twitter," *GitHub Repository*, 2013. [Online]. Available: https://github.com/sidooms/MovieTweetings

[20] Netflix Inc., "Netflix Prize Dataset," *Kaggle Dataset*, 2009. [Online]. Available: https://www.kaggle.com/datasets/netflix-inc/netflix-prize-data/data

[21] R. Lubis, M. Cercas Curry, and E. Casanueva, "INSPIRED: A Dataset for Emotion-Aware Conversational Recommendation," in *Proceedings of the 20th Annual SIGdial Meeting on Discourse and Dialogue*, 2019.

[22] J. Xiao et al., “C-Pack: Packaged Resources To Advance General Chinese Embedding,” *arXiv preprint arXiv:2309.07597*, 2023.

[23] OpenAI, “New Embedding Models and API Updates,” OpenAI Blog, 2024.

[24] Ampere Analysis, “Streaming Algorithms Now Beat Word-of-Mouth for TV and Movie Picks,” Available at: https://www.ampereanalysis.com/insight/streaming-algorithms-now-beat-word-of-mouth-for-tv-and-movie-picks, Accessed: December 2024.

[25] M. Bourreau and G. Gaudin, “Streaming Platform and Strategic Recommendation Bias,” *Journal of Economics & Management Strategy*, vol. 31, no. 1, pp. 25–47, 2022.

[26] M. A. Chatti, S. Daka, C. Thüs, and R. Klemke, “Generative AI and Large Language Models for Explainable Recommendations: A Perspective,” *arXiv preprint arXiv:2305.11755*, 2024.

[27] T. Balasubramaniam, R. Nayak, and W. Yue, “Explainability of Artificial Intelligence: A Review,” *Information Systems Frontiers*, vol. 25, pp. 1811–1829, 2023.

[28] S. Jayalakshmi, N. Ganesh, R. Čep, and J. Senthil Murugan, “Movie Recommender Systems: Concepts, Methods, Challenges, and Future Directions,” *Sensors*, vol. 22, no. 13, p. 4904, 2022.

[29] C. P. L. Tung, S.-C. Haw, W.-E. Kong, and P. Naveen, “Movie Recommender System based on Generative AI,” in *2024 International Symposium on Parallel Computing and Distributed Systems (PCDS)*, IEEE, 2024.

[30] M. A. Chatti, M. Guesmi, and A. Muslim, “Visualization for Recommendation Explainability: A Survey and New Perspectives,” *arXiv preprint arXiv:2305.11755v3*, 2024.

[31] MovieLens Dataset. https://files.grouplens.org/datasets/movielens/ml-1m.zip

[32] MovieTweetings Dataset. https://raw.githubusercontent.com/sidooms/MovieTweetings/master/latest

[33] INSPIRED Dialogue + Movie Dataset. https://raw.githubusercontent.com/sweetpeach/Inspired/master/data

[34] ReDial Dialogue Recommendation Dataset. https://github.com/ReDialData/website/raw/data/redial_dataset.zip

[35] Google CCPE Dataset. https://raw.githubusercontent.com/google-research-datasets/ccpe/main/data.json

[36] GoEmotions Dataset. https://storage.googleapis.com/gresearch/goemotions/data/full_dataset

[37] TMDB API Documentation. https://api.themoviedb.org/3