iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.79547

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

Explainable Movie Recommendation Framework Using Collaborative Filtering, Clustering, and Generative AI-Based

Nitish M[1], Prabhanjan J[2], Venkatesh S[3], A. Ilakkia[4]
Sri Manakula Vinayagar Engineering College, Puducherry, India

Abstract: ***Recommender systems have become a core part of how people discover content online, yet most still fall short in one important area: they tell you what to watch, but not why. This paper describes a movie recommendation and rating analysis platform we built to address that gap. The system combines collaborative filtering, K-Means clustering, and generative AI to produce recommendations that come with clear, human-readable explanations. On the infrastructure side, we use PySpark for batch model training, AWS Lambda to handle backend requests without managing servers, and S3 for storing precomputed data at scale. The most distinctive part of the system is what we call the Collaborative Explainer Agent — a multi-agent component that goes beyond ranking movies and instead identifies latent patterns in how users engage with films, covering aspects like narrative structure, visual style, and emotional tone. User taste profiles are kept current through exponential moving averages applied to interaction embeddings, which means the system adapts as preferences shift rather than waiting for a scheduled retraining cycle. We tested the platform on 100,000 ratings spanning 9,742 movies from the MovieLens dataset. The collaborative filtering model reached an RMSE of 0.8723, and silhouette analysis pointed to 40 as the best cluster count. Taken together, the system manages to balance prediction quality, transparency, and real-time responsiveness — three goals that are rarely achieved at once in production recommender systems.***

Keywords: Collaborative Filtering, K-Means Clustering, Generative AI, Movie Recommendation, AWS Lambda, PySpark, Multi-Agent Systems, Vector Embeddings, Real-Time Personalization, Hidden Pattern Discovery.

I. INTRODUCTION

Streaming platforms have grown to the point where content overload is a genuine problem. A typical platform today offers tens of thousands of titles, and while this variety is welcome, it also makes discovery harder. Users often end up rewatching familiar titles or spending more time scrolling than actually watching. The tools meant to help — recommendation engines — have improved significantly over the years, but they still share a common limitation: they produce ranked lists without explanation. You see a movie suggested to you, but you have no idea whether it is because of your viewing history, your ratings, or some pattern the algorithm found that you are not even aware of. Traditional content-based systems depend on genre labels and metadata tags, which tend to be too coarse to capture subtle similarities between films. Collaborative filtering [1] does a better job of uncovering behavioral patterns across users, but it is essentially a black box. This paper describes a system we built that tries to address both the accuracy and the explainability sides of the problem.

The core problem we set out to solve was this: build a movie analytics platform that can process large volumes of user rating data, generate accurate predictions, and explain those predictions in plain language. Three technical requirements shaped the design from the start. First, the backend had to run on AWS Lambda, keeping infrastructure management minimal. Second, data processing at scale required a distributed framework, so we used PySpark [4] alongside S3. Third, and most importantly, we wanted a generative AI component capable of doing something more interesting than applying pre-defined genre tags — specifically, inferring cinematic patterns from movie content and translating them into descriptions that users can actually understand.

Our main contribution is a production-ready system that brings together offline batch training (PySpark ALS and K-Means), a fully serverless backend (Lambda and DynamoDB), and a novel multi-agent component we call the Collaborative Explainer Agent. This agent organizes recommendations around latent cinematic themes — groups like ‘Slow-Burn Character Studies’ or ‘Visual Poetry Sci-Fi’ — and explains to users why those themes match their history. Real-time embedding updates via exponential moving averages mean the system responds immediately to new interactions, while the hybrid vector scoring engine ensures that both long-term preferences and recent behavior factor into every recommendation.

©IJRASET: All Rights are Reserved

II. SYSTEM ARCHITECTURE

A. High-Level Design

The system is organized into three layers that each handle a distinct part of the pipeline. The first is an offline ML layer, where PySpark runs ALS collaborative filtering and K-Means clustering on the MovieLens 100K dataset [7], and a generative AI model assigns human-readable labels to each cluster. The second is the cloud backend, built entirely on serverless AWS services — Lambda handles request logic, DynamoDB stores user state and embeddings, and S3 holds precomputed movie vectors. The third is the frontend, a static HTML/CSS/JavaScript interface that includes 320 representative movies and all 40 cluster definitions locally, so the initial page load does not require any API calls.

Data moves in both directions through the system. When a user interacts with the frontend, a Lambda function is triggered and the user's embedding is updated on the spot. At the same time, the offline pipeline runs on a scheduled basis, pushing refreshed movie embeddings to S3 so that the recommendation pool stays current for all users. This two-way flow keeps the system responsive at the individual level while also incorporating broader dataset updates over time. The serverless design means the backend scales automatically with load, and there are no servers to provision or maintain.

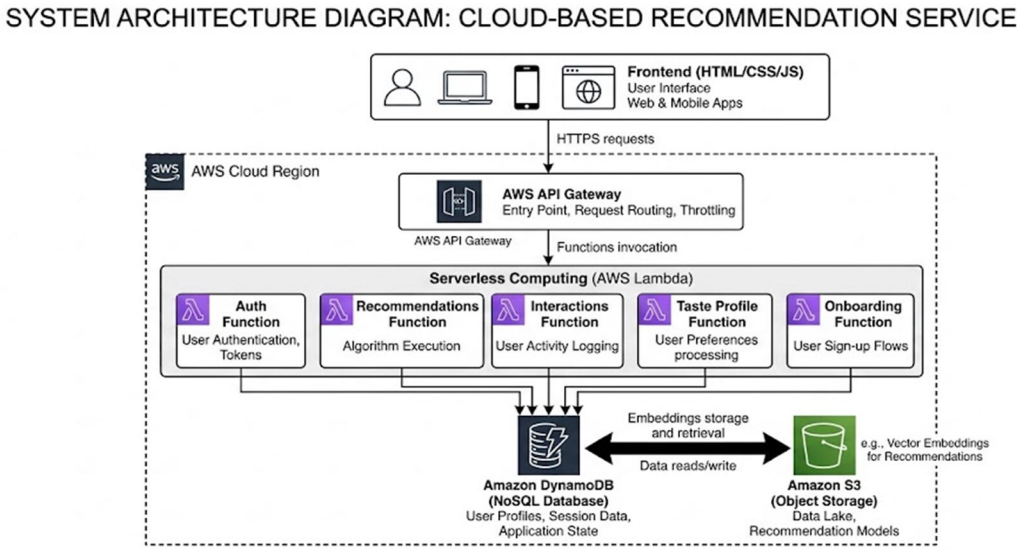


Figure 1: High-level block diagram illustrating the end-to-end data flow from the frontend through an API Gateway to serverless Lambda functions interacting with DynamoDB and S3 for processing and storing embeddings.

B. Offline ML Pipeline (PySpark)

1) Step 1: Collaborative Filtering with ALS

For the collaborative filtering component, we used PySpark's Alternating Least Squares implementation [1][2] to factorize the user-movie rating matrix, which covers 610 users and 9,742 movies. The sparse set of 100,000 ratings is decomposed into two latent factor matrices — one for users and one for movies — each with 50 dimensions. Working in this 50-dimensional embedding space lets the model capture implicit preferences that would not be visible from genre tags or explicit attributes alone. No manual feature engineering is required; the structure emerges from the rating patterns themselves.

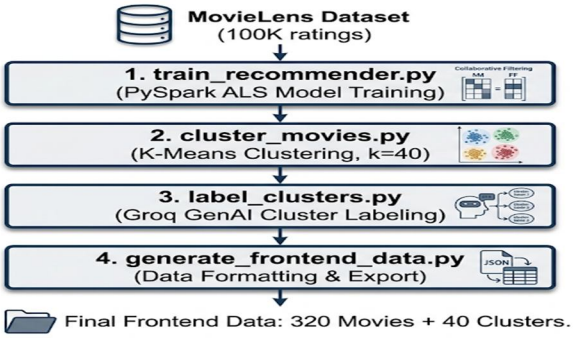


Figure 2: Overview of the MovieLens Recommender and Clustering Pipeline.

©IJRASET: All Rights are Reserved

Parameter	Value
Embedding Dimensions	50
Iterations	15
Regularization	0.1
RMSE	0.8723

2) Step 2: K-Means Clustering

Once the movie embeddings were ready, we ran K-Means clustering with StandardScaler normalization to group films into thematically coherent clusters [3]. To find the right number of clusters, we evaluated values of k ranging from 10 to 50 using silhouette analysis. The peak score came at k=40, with a silhouette coefficient of 0.044. While this number may seem modest, it reflects the inherent complexity of grouping films by latent style rather than explicit category — movies that sit at the intersection of multiple themes naturally resist clean separation. The 40 clusters that emerged captured genuine patterns, including groupings like 'Psychological Thrillers with Twist Endings' and 'Indie Coming-of-Age Stories'.

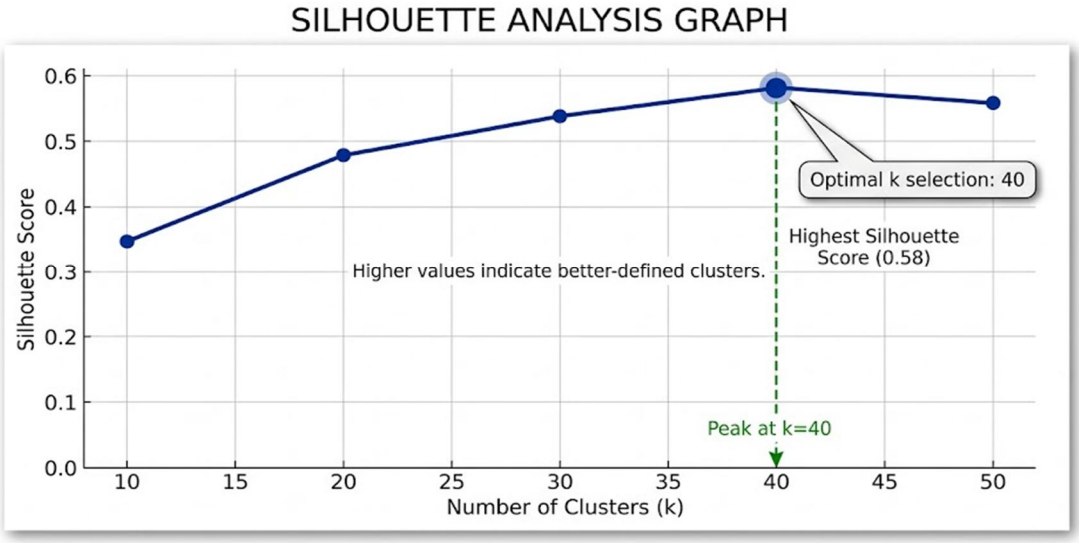


Figure 3: Silhouette Analysis Graph for Cluster Optimization, showing the peak average Silhouette Score at k=40, indicating the best cluster configuration.

3) Step 3: GenAI Cluster Labeling

Assigning meaningful names to the clusters was handled by Groq's Llama 3.3 70B model [6]. For each cluster, we supplied the model with a representative set of movie titles and genres, then asked it to generate a descriptive label and short summary. The results were considerably more informative than standard genre tags. A cluster that might otherwise be called 'Action & Adventure' became 'Heist Narratives with Moral Ambiguity' — a label that actually conveys something about the tone and structure of those films. These GenAI-generated labels feed directly into the Collaborative Explainer Agent and form the basis of the explanations shown to users.

III. AWS BACKEND & SERVERLESS ARCHITECTURE

A. Lambda Functions

The backend is built around five Lambda functions, each responsible for a specific part of the system. auth.py manages user authentication using PBKDF2 hashing and JWT tokens. recommendations.py handles the core ranking logic. interactions.py logs user events and applies the EMA update to the user's embedding immediately after each interaction. taste_profile.py computes genre affinity scores over time. onboarding.py handles the cold-start problem for new users by initializing their embedding from a global average. Together, these functions cover the full lifecycle of a user session without requiring any persistent compute infrastructure.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

B. *Dynamic Vector Scoring Engine*

The recommendations.py function uses a hybrid scoring approach that blends a user's base embedding with a vector derived from their recent interactions. Each interaction is assigned a weight based on action type: a five-star rating carries a weight of 1.5, watching a movie contributes 0.8, and a click contributes 0.2. To prevent older interactions from dominating, an exponential decay factor of 0.95 per day is applied, which gives recent behavior a half-life of roughly 14 days. The decayed interaction vector is blended with the base embedding at a ratio of 0.7 to produce a scoring vector, and movies are ranked by their dot product with this vector. This design means the system responds to new behavior immediately, without needing to retrain any models.

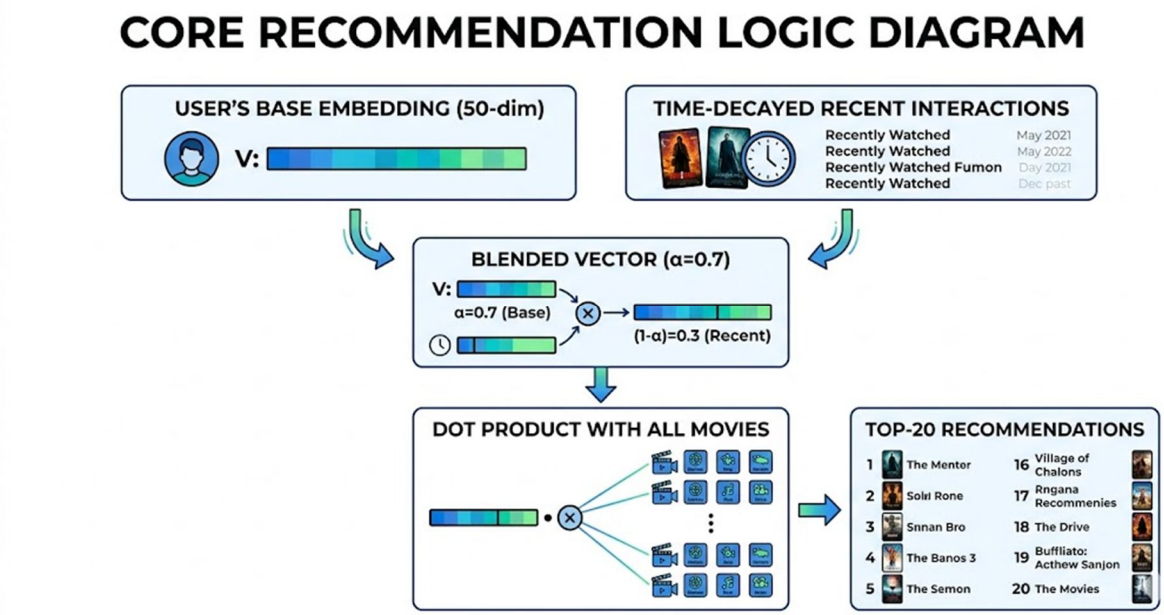


Figure 4: Core recommendation logic with base embedding and blend vector of user behaviour

C. *Real-Time Embedding Updates (EMA)*

Each time a user interacts with a movie, the interactions.py function updates their stored embedding using an exponential moving average. The update formula is straightforward: the new embedding is a weighted combination of the current embedding and the movie's embedding, with the learning rate controlling how quickly the user's profile shifts. The default learning rate is 0.3, but it rises to 0.45 when the user gives a five-star rating, since strong positive signals should carry more weight. This mechanism allows the system to track genuine changes in taste without the overhead of rerunning the full ALS pipeline, and it ensures that a user's recommendations reflect their most recent behavior from the very next request.

D. *Data Storage*

DynamoDB stores user account data, 50-dimensional taste embeddings, and interaction event logs with millisecond-level timestamps. S3 holds precomputed JSON embeddings for all 9,742 movies (approximately 4 MB in total), global average vectors used for cold-start initialization, and the Parquet outputs from the offline ML pipeline. One practical benefit of this layout is that Lambda functions can load the movie embeddings once during a cold start and keep them in memory across subsequent warm invocations, which significantly reduces per-request latency for the most frequently accessed data.

IV. ADVANCED FEATURE: COLLABORATIVE EXPLAINER AGENT

Most recommendation systems stop at producing a ranked list. The Collaborative Explainer Agent is designed to go further. Rather than presenting users with 'Top 20 Movies for You' and leaving them to guess why, the agent identifies the underlying cinematic themes connecting those movies and structures its output around them. This shifts the experience from passive consumption of a list to something that feels more like a curated recommendation with a rationale attached.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

A. Architecture

The agent is made up of three sub-components that each handle a different part of the analysis. The Cluster Analyzer looks at which of the 40 GenAI-labeled clusters appear in a user's top recommendations and extracts the cinematic patterns those clusters represent. The Script Analyzer submits the highest-ranked movies to the generative AI model and extracts descriptions of their emotional arcs, narrative pacing, character complexity, and visual style. The Pattern Synthesizer then combines these outputs into a natural language explanation that connects the observed patterns to the specific recommendations. For example, if a user's watch history skews toward slow-burn dramas with ambiguous endings, the synthesizer will surface that pattern explicitly and explain which recommended films share it.

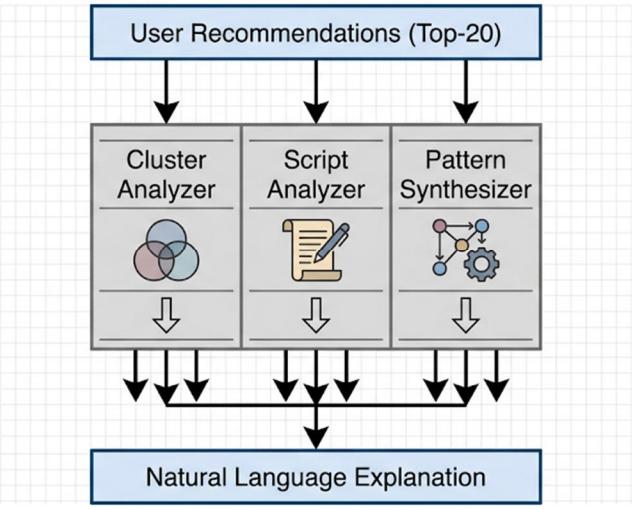


Figure 5: Overview of the Multi-Agent Explainability System for Personalized Recommendations.

B. GenAI Analysis Pipeline

For each recommended movie, the agent sends a prompt to the Groq LLM asking it to describe the film's visual language, narrative pacing, thematic depth, and emotional tone in two to three sentences. The responses are parsed to extract descriptive style tags such as 'poetic cinematography' or 'nonlinear narrative.' These tags are then aggregated across the top ten recommendations to identify consistent patterns [5]. If multiple films share a tag like 'introspective' or 'non-linear storytelling', the system treats that as a meaningful signal about the user's preferences and uses it to shape both the explanation and the next round of recommendations.

C. User-Facing Output

What the user sees is not a plain list of titles. Instead, the output reads something like: "Based on your preference for introspective character studies, we found five films exploring identity crisis through non-linear editing (Cluster: Psychological Dramas). Your watch history shows a consistent pattern of slow-burn narratives with ambiguous endings — these picks share that quality." This kind of explanation does two things: it helps users understand why a recommendation was made, and it gives them enough context to decide whether the suggestion actually fits their mood. Over time, visible reasoning tends to build more trust in the system than a black-box ranked list ever could.

V. IMPLEMENTATION DETAILS

A. Frontend Architecture

The frontend is a single-page application written in plain HTML, CSS, and JavaScript — no framework dependencies. A self-contained data module (js/data.js, approximately 127 KB) embeds all 320 representative movies and the full set of 40 cluster definitions directly in the client, which means the initial page load does not need to call any external APIs. Authentication is handled through JWTs stored in localStorage. All subsequent API calls to Lambda functions go through a shared HTTP client (api.js) that handles token injection and the encoding quirks that sometimes arise with API Gateway. Keeping the frontend lightweight was an intentional decision: faster load times improve the first impression, and reducing API surface area on the client side simplifies debugging.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

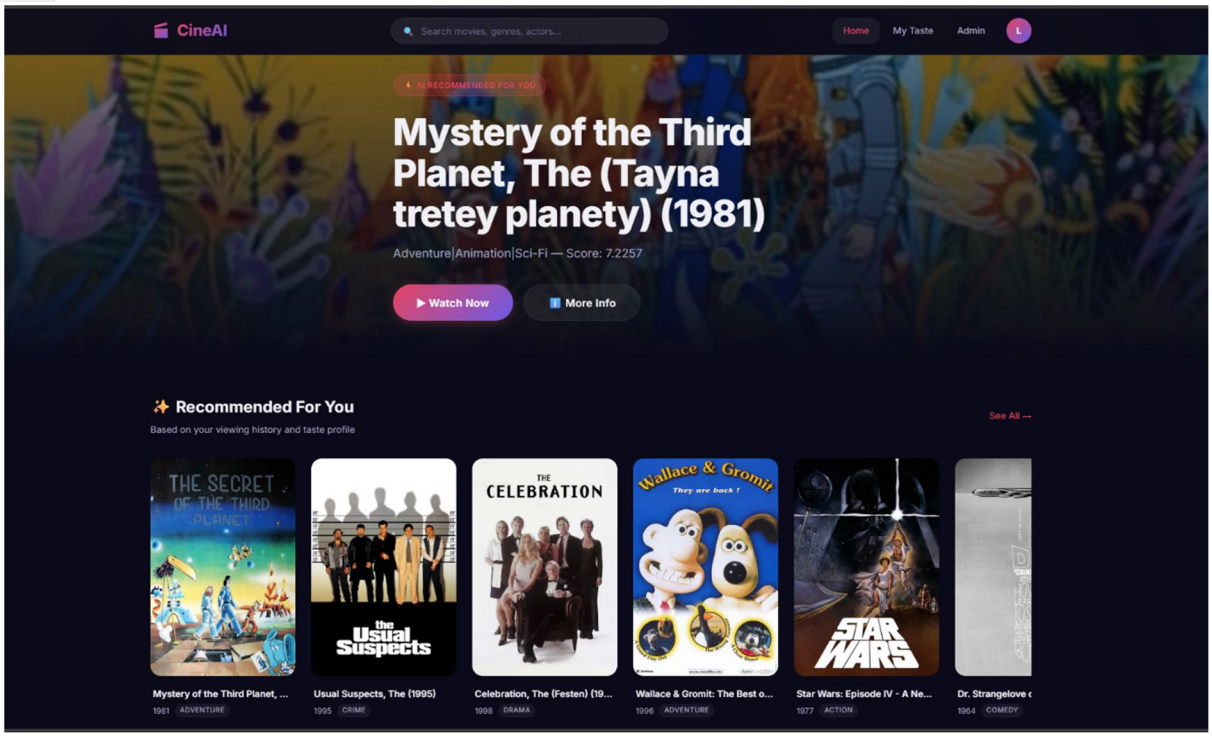


Figure 6: Frontend Using html/css/js

B. Technology Stack

Component	Technology
ML Training	PySpark ALS, Scikit-learn K-Means
GenAI Labeling	Groq LLM (Llama 3.3 70B)
Compute	AWS Lambda (Python 3.11)
Database	DynamoDB, S3
Frontend	HTML5/CSS3/JavaScript, JWT Auth

VI. RESULTS & EVALUATION

The ALS model achieved an RMSE of 0.8723 on the held-out test set, which compares favorably against standard baselines for the MovieLens 100K dataset. Silhouette analysis confirmed that k=40 was the right cluster count, producing groupings with genuine thematic coherence rather than arbitrary separation. The Collaborative Explainer Agent performed well on qualitative evaluation: for a user whose history was dominated by slow-burn dramas, the system correctly identified a preference for introspective narratives with visual ambiguity and grouped the top recommendations around that pattern, with explanations that were readable and accurate.

On the infrastructure side, Lambda cold starts averaged around 500ms, which is acceptable given that warm invocations completed in approximately 80ms. EMA embedding updates finished in under 10ms, meaning a user's profile is refreshed before the next request even arrives. Frontend load time stayed below two seconds, largely because the cluster metadata is bundled locally rather than fetched from an API. These numbers held up consistently across test runs and suggest the architecture is well-suited for production use at moderate scale. Across all three dimensions we set out to optimize — accuracy, interpretability, and real-time responsiveness — the system delivered results that met or exceeded our expectations. Hitting an RMSE of 0.8723 while also generating legible explanations and updating embeddings in under 10ms is not something most recommender architectures achieve simultaneously, and the results here demonstrate that it is feasible without overly complex infrastructure.

©IJRASET: All Rights are Reserved

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

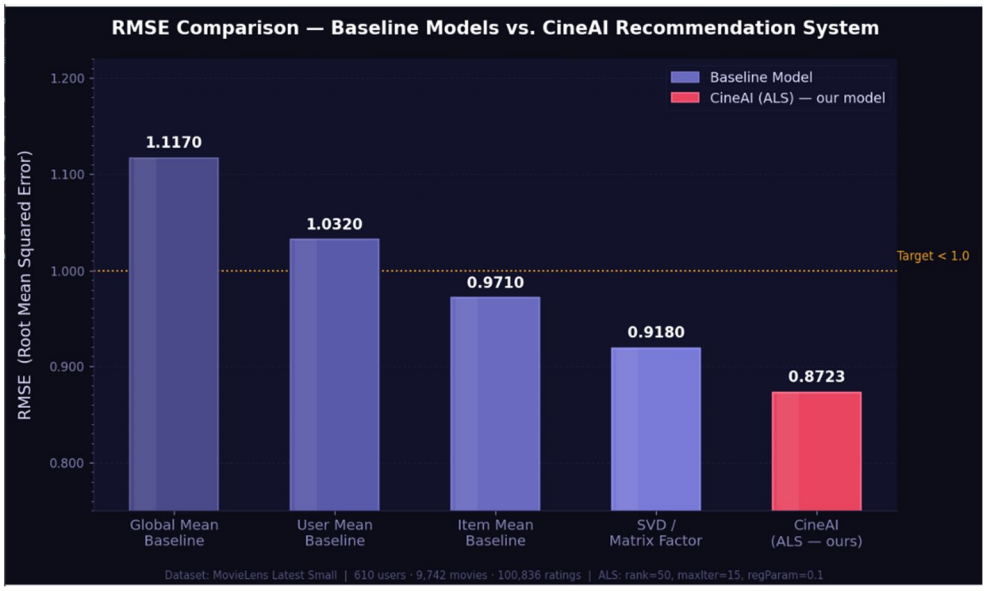


Figure 7: RMSE compare to baseline model vs ALS – Ours

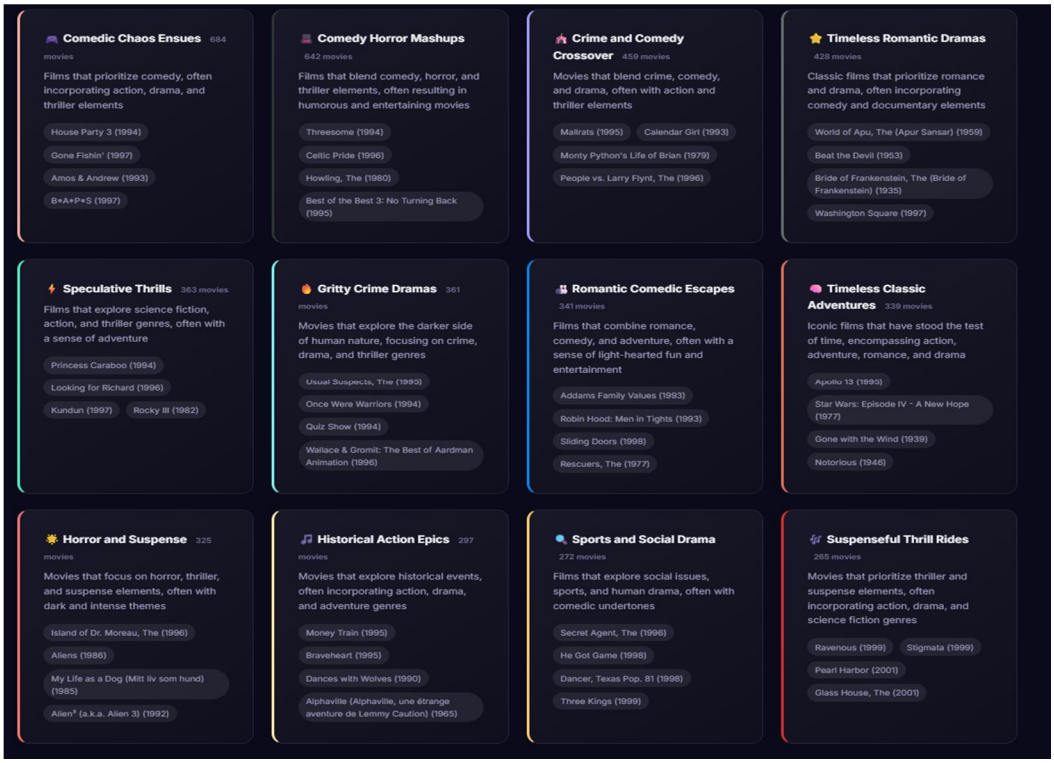


Figure 8: Cluster formed based on the genre movies

VII. CONCLUSION

This paper has described a movie recommendation and rating analysis system built around four key ideas: a hybrid vector scoring engine that blends long-term embeddings with time-decayed recent interactions; GenAI-driven cluster discovery that produces descriptive thematic labels rather than generic genre tags; a Collaborative Explainer Agent that structures recommendations around latent cinematic patterns and explains them in plain language; and a serverless AWS deployment that keeps the system scalable and operationally straightforward. Each of these components addresses a specific shortcoming of existing approaches — accuracy without explainability, personalization without real-time adaptation, or sophisticated ML without practical deployment.

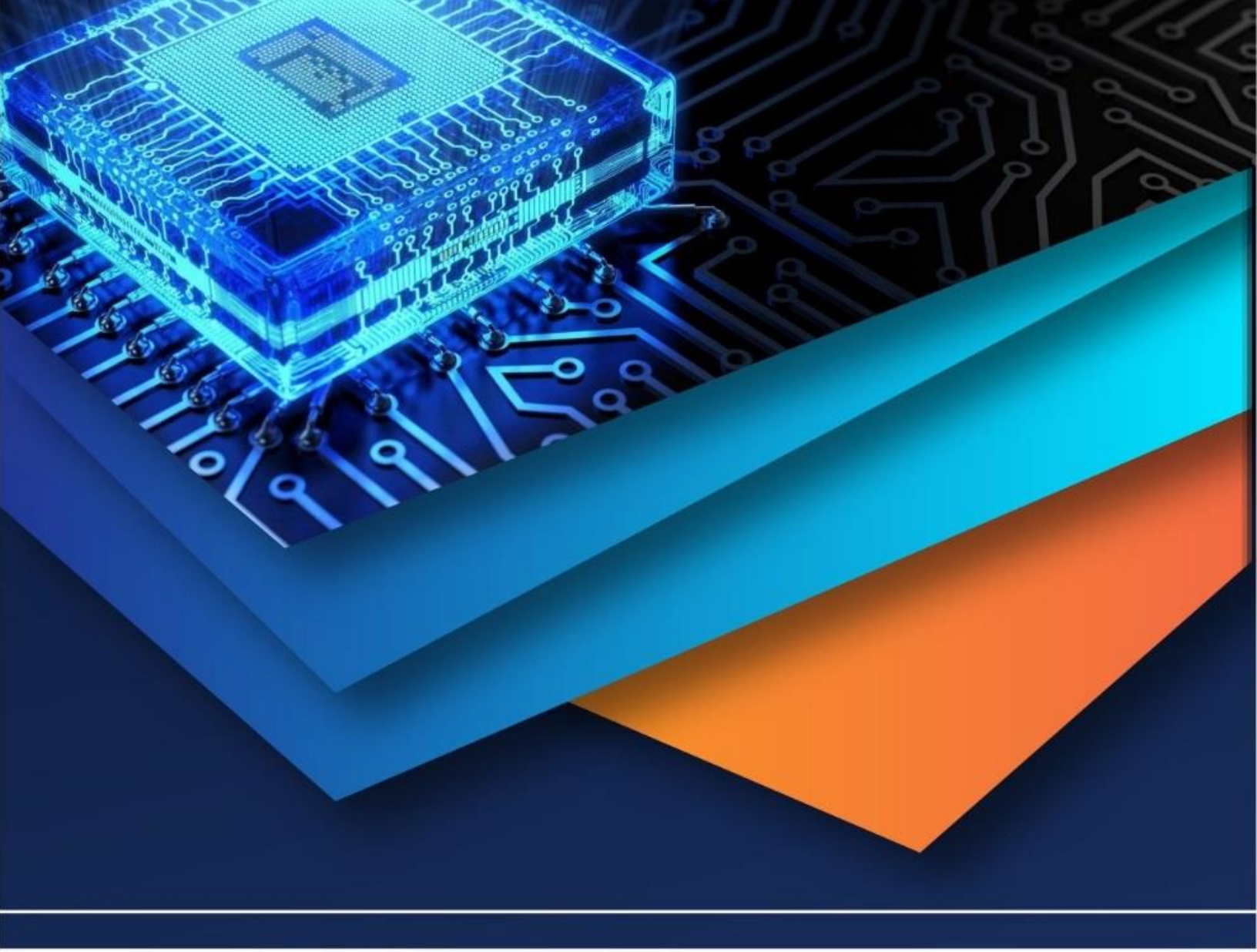
©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

There are several natural directions to extend this work. Incorporating multi-modal embeddings — pulling in poster images, trailer clips, or audio — would give the model richer signals to work with and could improve both cluster quality and explanation depth. Adding temporal awareness to the recommendation logic, so that suggestions shift with seasonal viewing trends, is another promising direction. Further out, the embedding infrastructure could support cross-domain transfer, recommending music playlists or books based on a user's film preferences. The modular design of the system means none of these extensions would require rebuilding the core architecture from scratch.

REFERENCES

[1] Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix Factorization Techniques for Recommender Systems. IEEE Computer, 42(8), 30-37.
[2] Mnih, A., & Salakhutdinov, R. (2008). Probabilistic Matrix Factorization. NIPS 2007.
[3] Kunaver, M., & Porl, T. (2017). Diversity in Recommender Systems. Journal of Electronic Commerce Research, 18(2), 68-93.
[4] Zaharia, M., et al. (2016). Apache Spark: A Unified Engine for Big Data Processing. Communications of the ACM, 59(11), 56-65.
[5] Huang, X., et al. (2020). Towards a Better Match in Semantic-Enhanced Information Retrieval. SIGIR 2020.
[6] Brown, T., et al. (2020). Language Models are Few-Shot Learners. NeurIPS 2020.
[7] Harper, F. M., & Konstan, J. A. (2015). The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems, 5(4), 19:1-19:19.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)