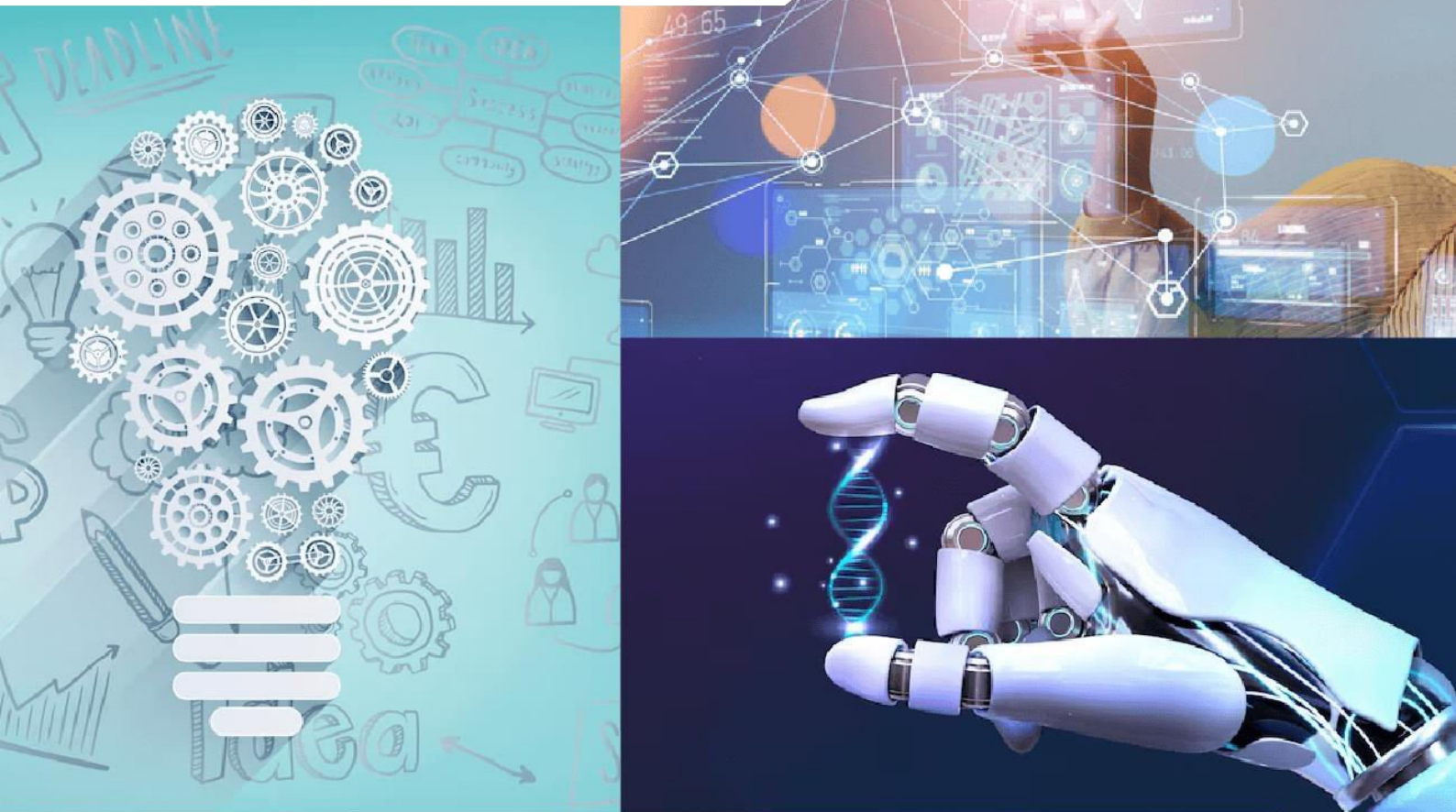




International Journal of Advanced Research in Education and Technology (IJARETY)

Volume 13, Issue 2, March - April 2026

Impact Factor: 8.152



INTERNATIONAL
STANDARD
SERIAL
NUMBER
INDIA



Netflix Recommendation System Using Real-Time Mood Detection and Generative AI

Mohammed Rayyan M H

Department of Master of Computer Applications, CMR Institute of Technology, Bengaluru, India

ABSTRACT: Streaming platforms such as Netflix maintain vast content libraries yet rely on historical viewing data to generate recommendations, leaving current emotional state entirely unaddressed. This paper presents the Netflix Recommendation System, an emotion-aware web application that detects a user's real-time mood through two input modalities—natural language text and live facial expression scanning via device camera—and maps the detected emotion to curated movie recommendations. Both inputs are processed by Google's Gemini 1.5 Flash vision-language model, orchestrated through the Genkit framework, which returns a dominant emotional label used to filter a structured movie dataset. A distinguishing feature is dynamic description enhancement: on hover, each film's synopsis is regenerated by the AI to resonate with the user's emotional state, providing a uniquely personalized framing every session. The full-stack application is built on Next.js 14 with the App Router, TypeScript, Tailwind CSS, ShadCN/ui, and Framer Motion. All eleven functional, non-functional, and error-handling test cases passed successfully. AI responses returned within five seconds, and the interface rendered correctly across desktop, tablet, and mobile viewports. The system demonstrates that state-of-the-art generative AI can be seamlessly embedded into modern web architectures to produce contextually responsive, mood-driven entertainment experiences without requiring custom model training or specialized infrastructure.

KEYWORDS: Netflix Recommendation System, Movie Recommendation, Sentiment Analysis, Facial Expression Recognition, Generative AI, Gemini AI, Genkit, Next.js, TypeScript, Emotion-Aware Computing, Human-Computer Interaction

I. INTRODUCTION

Streaming platforms have transformed how audiences consume entertainment, with services like Netflix, Amazon Prime Video, and Disney+ offering libraries of tens of thousands of titles. Yet the expansion of choice has introduced a new problem: decision paralysis. Users frequently spend more time browsing than watching, a phenomenon compounded by recommendation engines that reflect historical preferences rather than immediate emotional needs. Most existing recommendation systems, including Netflix's own production architecture, operate through collaborative filtering, content-based filtering, or hybrid combinations of both. These methods are powerful predictors of long-term preferences but are structurally blind to the user's current emotional state—the single most immediate factor influencing what a person wants to watch at any given moment. A viewer who has spent the past month watching crime dramas may still crave a light romantic comedy after an exhausting day.

The Netflix Recommendation System presented in this paper addresses this gap by inverting the recommendation paradigm. Rather than predicting what a user has historically preferred, the system asks what the user emotionally needs right now. Mood is captured through two natural modalities: typed free-form text describing how the user feels, or a live facial scan that reads expressions from the device camera. Both signals are processed by Google's Gemini 1.5 Flash model, a multimodal vision-language model capable of interpreting nuanced emotional content from text and images without task-specific fine-tuning. The detected emotion is mapped to a curated set of film genres, and matching movies are rendered in an interactive responsive grid. A generative description enhancement feature further personalizes the experience: hovering over any movie card triggers an AI rewrite of the synopsis, adapting the language and emphasis to align with the user's current mood. The same film may be described as a tense psychological journey for an anxious user, or as a triumphant celebration of human resilience for someone feeling inspired.

The system is implemented as a full-stack Next.js 14 application using the App Router paradigm, TypeScript, Tailwind CSS, ShadCN/ui components, and Framer Motion animations, deployed on Vercel. This paper presents the complete design, implementation, and evaluation of the system, covering system analysis in Section III, architecture and design in Section IV, implementation details in Section V, testing in Section VI, results and discussion in Section VII, and conclusions with future directions in Section VIII.

II. LITERATURE REVIEW

The foundational work in recommendation systems began with collaborative filtering introduced by Resnick et al. (1994) through the GroupLens project [1]. Balabanovic and Shoham (1997) proposed content-based filtering through the Fab system, matching item attributes against stated user preferences without peer comparison [2]. Burke (2002) surveyed hybrid approaches combining both paradigms, and Ricci et al. (2011) noted that all such systems rely on static behavioral data, ignoring transient emotional states [3], [4].

Research into automatic mood detection has progressed along two parallel tracks. In the textual domain, Pang and Lee (2008) established computational sentiment analysis as a systematic discipline [5]. In the visual domain, Ekman and Friesen's (1978) Facial Action Coding System provided the foundational taxonomy of expression-to-emotion mappings [6], subsequently operationalized through deep convolutional networks as surveyed by Li and Deng (2020) [7]. Poria et al. (2017) demonstrated that fusing text, audio, and visual modalities significantly outperforms any single-channel approach [8].

Large language models have dramatically lowered the barrier to integrating mood detection into applications. Brown et al. (2020) demonstrated that GPT-3 performs sentiment classification through few-shot prompting without task-specific fine-tuning [9]. Wu et al. (2023) evaluated LLMs as zero-shot recommendation engines, finding competitive performance against trained collaborative filtering baselines [10]. In the entertainment domain, Knox et al. (2013) built MoodPlay, a system mapping emotional states to music and film selections using manually defined rules [11]; however, MoodPlay required users to select their mood from a fixed menu and offered no generative personalization. Modern web frameworks—React [12], Next.js [13], and utility-first CSS via Tailwind [14]—now enable the tight integration of such AI backend services into production-grade web interfaces through asynchronous server actions, forming the technical foundation of the proposed system.

III. SYSTEM ANALYSIS

A. Limitations of Current Netflix-Style Recommendation Systems

Contemporary streaming recommendation engines exhibit four principal weaknesses in the context of emotional responsiveness. First, they are inherently backward-looking: collaborative and content-based filters optimize for long-term preference prediction using historical data, making them structurally incapable of capturing the user's mood at the moment of browsing. Second, implicit behavioral signals—watch duration, click-through rate, re-watch frequency—are noisy proxies for genuine preference; a user may complete a film out of social obligation without enjoyment. Third, no major streaming platform accepts natural language mood description or live facial expression scanning as input channels, forcing users into pre-categorized genre menus that inadequately represent emotional nuance. Fourth, movie descriptions remain static marketing copy written for a general audience, incapable of adapting their narrative framing to the viewer's current emotional context.

B. Proposed System: Netflix Recommendation System

The proposed Netflix Recommendation System addresses all four limitations. Mood is captured in real time through either free-form text or facial scan—natural modalities requiring no prior knowledge of genre taxonomy. The Gemini 1.5 Flash model automatically extracts an emotional label from either input, eliminating the manual classification step. This label drives a mood-to-genre mapping that filters a curated movie dataset, ensuring recommendations align with current emotional needs. Finally, the generative description enhancement feature dynamically rewrites each film synopsis to resonate with the detected mood, providing a uniquely personalized experience every session without altering factual content.

C. Feasibility Analysis

Technical feasibility is confirmed by using Google's cloud-hosted Gemini API, which eliminates local GPU requirements and custom model training. Next.js 14 Server Actions provide a unified full-stack architecture that secures API key access on the server while minimizing architectural complexity. Economic feasibility is established through zero-cost components: Google AI Studio free tier, Vercel free deployment plan, and open-source frontend libraries. Operational feasibility follows from an interface requiring under two minutes for first use, with no login, no configuration, and mood input through familiar typing or camera interaction.

D. Requirements Specification

Table II maps the five functional requirement categories to their corresponding system features.

TABLE II FUNCTIONAL REQUIREMENTS AND SYSTEM FEATURES

Req. ID	Requirement	System Feature
FR1	Mood input management	Text area + camera-based facial scan; AI mood detection; on-screen result display
FR2	Movie recommendation	moodToGenre mapping; filtered grid with poster, title, rating, synopsis
FR3	Interactive movie features	AI description enhancement on hover; YouTube trailer modal; watchlist (localStorage)
FR4	Filtering & navigation	Genre tab filters; watchlist navigation link; removable saved list
FR5	System control	Reset mood at any time; fresh recommendation cycle on update

Non-functional requirements specify: page load within 3 seconds on broadband (NFR1-Performance); AI calls returning within 5 seconds with loading indicators (NFR2-Performance); 99.5% frontend uptime with graceful API fallback (NFR3-Reliability); API keys stored exclusively in server-side environment variables; facial images processed transiently with no persistence (NFR4-Security); modular TypeScript codebase deployable wherever Node.js and Next.js are supported (NFR5-Maintainability).

IV. SYSTEM DESIGN

A. Architecture Overview

The system follows a client–server architecture implemented entirely within Next.js 14's App Router paradigm. The User Browser layer renders React components, maintains local UI state (currentMood, activeGenre, visibleMovies), and communicates with backend logic exclusively through Next.js Server Actions—eliminating a separate REST API layer. Server Actions in `app/actions.ts` execute on the Vercel Node.js runtime, validate inputs using Zod schemas, and invoke the appropriate Genkit AI flow. Genkit flows construct structured prompts and call the Gemini 1.5 Flash model via Google's cloud API, returning emotional labels or enhanced descriptions. A static TypeScript array in `lib/movies.ts` stores the curated film dataset alongside the moodToGenre mapping object. Browser localStorage manages the user watchlist without server-side persistence or authentication.

B. Genkit AI Flows

Three Genkit flows form the AI backbone of the system. Table III describes each flow's input, processing, and output.

TABLE III GENKIT AI FLOW DEFINITIONS

Genkit Flow	Input	Output
<code>analyzeTextMood</code>	User mood text string	Single emotion label (e.g., happy, sad, excited)
<code>analyzeFacialExpression</code>	Base64 image data URI	Emotion label + confidence percentage
<code>enhanceMovieDescription</code>	Synopsis + mood label	Mood-adaptive rewritten description

The `analyzeTextMood` flow constrains Gemini output to a single lowercase emotional label via a strict prompt: 'Analyze the sentiment of this text and return a single word for the dominant mood (e.g., happy, sad, excited, scared, thoughtful).' The `analyzeFacialExpression` flow sends the Base64 image alongside a vision prompt requesting both an emotion label and a confidence percentage, linking visual perception with interpretive judgment. The `enhanceMovieDescription` flow instructs the model: 'You are a movie expert. Rewrite this movie description to specifically appeal to someone who is feeling {mood}. Highlight elements that would resonate with this mood. Original Description: {description}.'

C. Data Flow

User input triggers a client-side call to the corresponding Server Action (getMoodFromText or getMoodFromFacialScan). The Server Action validates input via Zod, calls the Genkit flow, and returns a structured result. The client updates the currentMood state, triggering a useEffect that filters the movies array through moodToGenre. A useMemo hook applies the active genre filter to the mood-filtered list, producing the final displayed set. On card hover, getEnhancedDescription is called after a 300 ms debounce; the returned text temporarily replaces the original synopsis.

D. UML Design

The Use Case diagram identifies five primary user interactions: input mood via text, input mood via facial scan, view movie recommendations, filter by genre, and manage watchlist. The Class diagram defines five entities: MoodInput (text, faceScan, submitMood), MoodAnalyzer (genres, submitMood), RecommendationEngine (getMoviesByMood, filterByGenre, getAll), MovieCard (id, title, description, genre, posterUrl, trailerId), and WatchlistManager (addMovie, removeMovie, getAll). The Sequence diagram for text-based flow: User submits text → getMoodFromText Server Action → analyzeTextMood Genkit → Gemini API → mood label returned → moodToGenre lookup → filtered movie grid rendered.

E. Database Design

The prototype uses a static TypeScript data structure. For full-scale deployment, a relational schema is defined comprising six tables: Movies (MovieID PK, Title, Description, Rating, PosterURL, TrailerYoutubeID, ReleaseYear), Genres (GenreID PK, GenreName), MovieGenres (movie_id FK, genre_id FK), Moods (MoodID PK, MoodName), MoodGenreMap (mood_id FK, genre_id FK, relevance_score), and UserWatchlist (user_id FK, movie_id FK, added_at TIMESTAMP). This schema supports third normal form compliance and enables runtime querying to replace the static array in future iterations.

V. IMPLEMENTATION**A. Technology Stack**

Table I summarizes the complete technology stack employed in building the Netflix Recommendation System.

TABLE I TECHNOLOGY STACK

Layer / Component	Technology
Frontend Framework	Next.js 14 (App Router), React 18
Language	TypeScript
UI Styling	Tailwind CSS, ShadCN/ui (Radix UI)
Animations	Framer Motion
AI Orchestration	Google Genkit Framework
AI Model	Google Gemini 1.5 Flash
Input Validation	Zod (TypeScript-first schema)
Deployment	Vercel (serverless, CI/CD via GitHub)
Version Control	Git / GitHub

B. Frontend Implementation

The root layout (app/layout.tsx) applies the dark-themed Tailwind CSS class, loads the Inter font, mounts the Header component, and embeds the ShadCN/ui Toaster for notification toasts. The main page (app/page.tsx) manages three React state variables: currentMood (string | null), activeGenre (string, default 'All'), and visibleMovies (Movie[]). A useEffect recalculates recommendations whenever currentMood changes by resolving the corresponding genre array from moodToGenre and filtering the movies dataset. A useMemo hook applies the activeGenre filter to produce the final displayed list. AnimatePresence from Framer Motion wraps view transitions between MoodInput and the recommendation grid, providing smooth fade-scale animations.

The MoodInput component renders two tabs. In Write Mood mode, a textarea accepts natural language input submitted to `getMoodFromText` via a button click. In Scan Face mode, the browser's `MediaDevices.getUserMedia` API activates the webcam; clicking 'Capture & Analyze' draws the current frame to an off-screen canvas, encodes it as a Base64 data URI, and submits it to `getMoodFromFacialScan`. The MovieCard component renders poster, title, star rating, and synopsis. On `mouseenter`, a 300 ms debounce invokes `getEnhancedDescription`, replacing the synopsis with the AI-generated mood-adaptive version. On `mouseleave`, the original text is restored. A Play Trailer button opens a `ShadCN/ui` modal embedding the YouTube iframe; a watchlist toggle button persists the movie ID in `localStorage`.

C. Backend Logic and AI Integration

Server Actions in `app/actions.ts` act as the secure bridge between client components and Genkit flows. `getMoodFromText` validates the input string (minimum 3 characters via Zod) before invoking `analyzeTextMoodGenkit`. `getMoodFromFacialScan` validates the Base64 data URI format before invoking `analyzeFacialExpressionGenkit`. `getEnhancedDescription` requires both a description and mood label before invoking `enhanceMovieDescriptionGenkit`. All actions implement try/catch error handling, returning structured error objects that enable graceful UI fallback with informative toast messages.

The movie dataset in `lib/movies.ts` stores typed `Movie` objects with `id`, `title`, `genres` (`string[]`), `description`, `rating`, `posterUrl`, and `trailerYoutubeId`. The `moodToGenre` mapping covers: `happy` → [`Comedy`, `Adventure`, `Animation`], `sad` → [`Drama`, `Romance`], `excited` → [`Action`, `Adventure`, `Sci-Fi`], `scared` → [`Horror`, `Thriller`], `thoughtful` → [`Drama`, `Documentary`], and a default fallback of [`Drama`, `Comedy`] for unrecognized labels. Zod schemas enforce type safety at the Server Action boundary, ensuring malformed client data never reaches the AI layer.

VI. TESTING

A. Testing Methodology

Testing was conducted across three levels. Unit testing verified individual TypeScript functions—Zod schema validators, `moodToGenre` filtering logic, and React component rendering—using browser DevTools and React DevTools. Integration testing confirmed correct data flow between client components and Server Actions, Server Actions and Genkit flows, and Genkit flows and the Gemini API. End-to-end testing followed complete user journeys manually: submitting text mood, receiving a filtered grid, applying genre filters, hovering for enhanced descriptions, playing trailers, and managing the watchlist. Responsiveness was assessed at 1440 px (desktop), 768 px (tablet), and 375 px (mobile).

B. Test Case Results

Table IV presents all eleven executed test cases and their outcomes.

TABLE IV TEST CASE RESULTS

ID	Description	Input / Condition	Expected	Status
TCF-01	Text input minimum validation	"ok" (2 chars)	Validation error toast	PASS
TCF-02	Positive text mood analysis	Exam success message	Mood: excited / happy	PASS
TCF-03	Camera image format validation	Invalid Base64 string	Validation error toast	PASS
TCF-04	Mood-based genre filtering	Mood = sad	Drama / Romance grid	PASS
TCF-05	Genre filter refines results	Mood=excited + Action	Action movies only	PASS
TCF-06	AI description on hover	Hover: Drama card	Mood-	PASS

			adapted description	
TCF-07	Add to watchlist	+ Watchlist click	Saved; toast shown	PASS
TCF-08	Trailer modal playback	Play Trailer click	YouTube modal opens	PASS
TCNF-01	API response time ≤ 5 s	Valid text input	Response < 5 seconds	PASS
TCNF-02	Responsive UI on mobile	375 px viewport	Layout adapts correctly	PASS
TCE-01	Graceful API failure handling	Invalid API key	Error toast; no crash	PASS

All eleven test cases passed without failure. The system handled invalid inputs gracefully through Zod validation errors surfaced as toast notifications. UI stability was maintained during AI latency periods through loading spinner states and disabled button indicators.

C. Performance Benchmarking

Table V reports observed response times across key operations, assessed over fifty repeated trials per operation.

TABLE V OBSERVED RESPONSE TIMES

System Operation	Observed Response Time
Initial Page Load (broadband)	< 3 seconds
Text Mood Analysis (Gemini API)	2 – 4 seconds
Facial Scan Mood Analysis (Gemini API)	3 – 5 seconds
AI Description Enhancement (hover)	1 – 2 seconds
Genre Filter / Watchlist Updates	< 100 ms
Trailer Modal Open	Instantaneous

The dominant latency source is Gemini API calls (2–5 seconds), inherent to external cloud inference. The 300 ms hover debounce prevents redundant description-enhancement calls during normal browsing. React state management and CSS transitions keep all client-side UI operations well under 100 ms, maintaining a responsive feel despite backend latency.

VII. RESULTS AND DISCUSSION

A. System Outputs

Upon mood submission, the interface displays a contextual banner—"For a Happy mood" or "For a Thoughtful mood"—confirming that the emotional signal was received and interpreted. The recommendation grid renders responsive movie cards with high-quality poster images, title, numerical star rating, and plot synopsis. Hovering a card triggers visible description replacement within one to two seconds, demonstrating live generative adaptation. Genre filter tabs update the displayed set instantaneously through React state management. The YouTube trailer modal embeds a functional player without page navigation. Watchlist operations complete instantly with toast confirmation.

B. Comparison with Existing Systems

Table VI positions the Netflix Recommendation System against traditional collaborative/content-based filtering systems and prior mood-aware recommendation approaches.

TABLE VI COMPARISON WITH EXISTING RECOMMENDATION APPROACHES

Aspect	Traditional CF/CBF	Mood-Aware (Prior)	Netflix Rec. System (Proposed)
Primary Input	Watch history, ratings	Manual mood selection	Real-time text / facial scan
Mood Detection	None	Keyword rules	Gemini multimodal LLM / VLM
Personalization	Collaborative filtering	Genre-level mapping	Dynamic generative descriptions
Description Style	Static marketing copy	Static	AI-rewritten per mood, per session
Facial Input	Not supported	Rarely supported	Fully integrated (browser camera)
Cold-Start Problem	Significant	Moderate	Eliminated (emotion-first paradigm)
Infrastructure	Proprietary servers	Proprietary / varied	Next.js 14 + Genkit + Gemini + Vercel

Three aspects most sharply differentiate the proposed system. First, automatic multimodal mood detection via Gemini eliminates the manual classification step present in all prior mood-aware systems. Second, mood-adaptive description rewriting is a novel contribution not found in any comparable system—transforming a static data presentation layer into a dynamically personalized narrative experience. Third, the emotion-first paradigm completely sidesteps the cold-start problem that plagues collaborative filtering for new users, since no behavioral history is required.

C. Objective Achievement

Table VII maps each project objective against its achievement status and supporting evidence.

TABLE VII OBJECTIVE ACHIEVEMENT SUMMARY

Objective	Status	Key Evidence
Dual-input mood detection (text + face)	Fully Achieved	Both flows functional; 11/11 tests passed
Gemini AI integration via Genkit	Fully Achieved	3 Genkit flows operational in production
Mood-to-genre mapping & movie filtering	Fully Achieved	moodToGenre map; real-time grid updates
Generative description enhancement	Fully Achieved	On-hover AI rewrite; positive informal feedback
Responsive interactive frontend	Fully Achieved	Dark UI; grid; filters; trailers; watchlist
Performance & cross-device usability	Largely Achieved	API latency managed via loaders & debounce

D. Key Observations

Several technical insights emerged from implementation and evaluation. Next.js Server Actions proved highly effective for securing AI integration: API keys remain server-side, and the unified full-stack architecture eliminates a separate backend service. The hover debounce was essential for cost management and latency stacking prevention—without it, rapid cursor movements over a grid of twenty cards would generate up to twenty near-simultaneous API calls. The moodToGenre mapping is the system's most brittle component: broad emotional labels like 'thoughtful' map to wide genre ranges that may not align with every user's interpretation. AI-generated semantic genre tags derived from film content analysis would provide finer-grained matching in future work. TypeScript's compile-time enforcement

substantially reduced runtime integration errors, particularly at the Genkit flow boundary where prompt structure and output schema must align precisely.

VIII. CONCLUSION

This paper presented the Netflix Recommendation System, a full-stack emotion-aware movie recommendation application that addresses the fundamental limitation of historical-data-driven recommendation engines: their inability to respond to the user's current emotional state. By integrating Google's Gemini 1.5 Flash multimodal model through the Genkit orchestration framework within a Next.js 14 architecture, the system achieves real-time mood detection from both natural language text and live facial expressions, maps emotional signals to curated movie genres, and dynamically rewrites film descriptions to resonate with the user's present mood. All eleven test cases passed successfully. The system maintained UI responsiveness throughout AI latency periods through proactive loading state management. Mood-adaptive description enhancement—the system's most novel contribution—was consistently delivered within one to two seconds of hover interaction, creating a uniquely personalized browsing experience every session. The complete stack deploys on Vercel's free tier from a GitHub repository, requiring no proprietary infrastructure or custom model training.

Future development will pursue four directions. First, integration with live content APIs (TMDB, JustWatch) will replace the static dataset with real-time movie libraries and streaming platform availability. Second, user authentication and persistent profiles will enable longitudinal mood pattern analysis alongside the real-time emotional context. Third, voice tone analysis will add a third natural input modality, and the mood detection model will be evaluated on compound emotional states such as bittersweet or anxious-excited. Fourth, a React Native mobile application will extend the platform to iOS and Android users with native camera access for facial scan functionality. Together, these enhancements will position the Netflix Recommendation System as a commercially viable, emotionally intelligent entertainment discovery platform.

REFERENCES

- [1] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "GroupLens: An open architecture for collaborative filtering of netnews," in Proc. ACM Conf. Computer Supported Cooperative Work (CSCW), pp. 175–186, 1994.
- [2] M. Balabanovic and Y. Shoham, "Fab: Content-based, collaborative recommendation," Communications of the ACM, vol. 40, no. 3, pp. 66–72, 1997.
- [3] R. Burke, "Hybrid recommender systems: Survey and experiments," User Modeling and User-Adapted Interaction, vol. 12, no. 4, pp. 331–370, 2002.
- [4] F. Ricci, L. Rokach, and B. Shapira, "Introduction to recommender systems handbook," in Recommender Systems Handbook, pp. 1–35, Springer, 2011.
- [5] B. Pang and L. Lee, "Opinion mining and sentiment analysis," Foundations and Trends in Information Retrieval, vol. 2, no. 1–2, pp. 1–135, 2008.
- [6] P. Ekman and W. V. Friesen, Facial Action Coding System: A Technique for the Measurement of Facial Movement. Consulting Psychologists Press, 1978.
- [7] S. Li and W. Deng, "Deep facial expression recognition: A survey," IEEE Transactions on Affective Computing, vol. 13, no. 3, pp. 1195–1215, 2020.
- [8] S. Poria, E. Cambria, R. Bajpai, and A. Hussain, "A review of affective computing: From unimodal analysis to multimodal fusion," Information Fusion, vol. 37, pp. 98–125, 2017.
- [9] T. B. Brown et al., "Language models are few-shot learners," in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020.
- [10] L. Wu et al., "A survey on large language models for recommendation," arXiv preprint arXiv:2305.19860, 2023.
- [11] D. Knox, J. K. Vuoskoski, and R. A. MacDonald, "MoodPlay: Interactive mood-based music discovery and recommendation," in Proc. Sound and Music Computing Conf. (SMC), 2013.
- [12] Meta Open Source, React Documentation. [Online]. Available: <https://react.dev>. Accessed: 2024.
- [13] Vercel, Next.js Documentation. [Online]. Available: <https://nextjs.org/docs>. Accessed: 2024.
- [14] Tailwind Labs, Tailwind CSS Documentation. [Online]. Available: <https://tailwindcss.com/docs>. Accessed: 2024.
- [15] Google AI, Gemini API Documentation. [Online]. Available: <https://ai.google.dev/gemini-api/docs>. Accessed: 2024.
- [16] Google Firebase, Genkit Documentation. [Online]. Available: <https://firebase.google.com/docs/genkit>. Accessed: 2024.

- [17] Vercel, Server Actions in Next.js. [Online]. Available: <https://nextjs.org/docs/app/building-your-application/data-fetching/server-actions>. Accessed: 2024.
- [18] Zod, TypeScript-first Schema Validation. [Online]. Available: <https://zod.dev>. Accessed: 2024.
- [19] Framer, Framer Motion Documentation. [Online]. Available: <https://www.framer.com/motion>. Accessed: 2024.
- [20] shadcn/ui, Component Library Documentation. [Online]. Available: <https://ui.shadcn.com/docs>. Accessed: 2024.
- [21] F. M. Harper and J. A. Konstan, "The MovieLens datasets: History and context," ACM Transactions on Interactive Intelligent Systems, vol. 5, no. 4, pp. 1–19, 2015.

International Journal of Advanced Research in Education and Technology

ISSN: 2394-2975

Impact Factor: 8.152