Hacklytics Project Report

Group Members - Ajay Suresh, Vivek Prakash, Aaryan Kadam

February 22, 2026

Abstract

We present Vibe Engine, a multimodal music recommendation system designed to retrieve songs based on emotional intent rather than prior listening behavior. User text or images are converted into structured semantic queries using Gemini and embedded with MiniLM for large-scale lyric retrieval. Results are ranked using a blended score combining semantic similarity, popularity, and recency, with diversity constraints. The system supports image-based queries and Spotify enrichment for an intuitive end-user experience.

1 Motivation & System Overview

Modern music recommendation systems optimize for historical engagement rather than a listener's current emotional context. Users lack an effective way to express and retrieve music that matches how they feel in the moment - there is no practical way to search for a "vibe" such as emotional recovery, nostalgia, or late-night focus.

We built Vibe Engine, a multimodal music recommendation system that retrieves songs based on emotional intent rather than listening history. Users can describe a vibe in natural language or upload an image; a Gemini-powered enrichment layer converts this input into structured semantic signals, which are embedded and matched against a large lyric corpus. Recommendations are ranked using a blended score combining semantic similarity, popularity, and recency, and presented through a clean web interface with Spotify links and album art.

2 Ideation & Development Process

2.1 Dataset & Preprocessing

We used the Spotify 5M Songs Lyrics dataset from Kaggle and selected the top 500,000 tracks by view count to balance cultural relevance with computational feasibility. This subset captures widely recognizable music while enabling scalable semantic retrieval.

The raw data was processed using Apache Spark to handle scale efficiently. We selected key fields (title, artist, lyrics, views, year), removed nulls and duplicates, normalized text, stripped structural tags from lyrics, truncated extreme outliers, and exported the cleaned data into partitioned Parquet files for efficient downstream loading and embedding.

2.2 Lyrics Embedding (Stage 2)

We embedded 500,000 songs using the MiniLM sentence transformer, chosen for strong semantic performance under limited compute. Embeddings were generated in chunks on GPU and stored as a dense vector matrix for fast similarity search.

2.3 Recommendation Engine (Stage 3)

Our pipeline operates in four steps:

- **Prompt Enrichment:** Gemini rewrites the user input into a retrieval-optimized query and extracts moods, themes, and keywords.
- **Embedding:** The enriched query is embedded and matched against lyric embeddings via cosine similarity.
- **Blending Score:** Semantic similarity is combined with normalized popularity and recency using user-controlled weights.
- **Diversity Filtering:** Results are filtered by minimum similarity and capped per artist to ensure variety.

2.4 Product & User Experience

Vibe Engine is deployed as a simple web application with a FastAPI backend and a React frontend. The backend exposes two endpoints for recommendations: one for text prompts and one for image uploads. Users can describe a vibe or upload an image, adjust the weighting of popularity and recency, and choose the number of results to return.
Each recommendation is presented with album art, song metadata, a Spotify link, and a brief explanation of why the song matches the user's vibe. Spotify enrichment is handled as a best-effort enhancement using a two-stage lookup strategy with fuzzy validation, ensuring that incorrect links are avoided. The core recommendation pipeline remains fully functional even when Spotify data is unavailable.

3 Results

Initial demos show that the system retrieves emotionally aligned and diverse recommendations across abstract prompts (e.g., "late-night introspection") and image inputs. Final quantitative and qualitative evaluations will be added after the demo run.

4 Challenges & How We Addressed Them

- **Compute Constraints:** We used MiniLM and FP16 chunked processing to embed 500K songs under limited GPU memory.
- **LLM Rate Limits:** Gemini enrichment occasionally hit quota limits; the system degrades gracefully by falling back to raw queries.
- **Multimodal Tradeoff:** Rather than re-embedding the full corpus with CLIP, we used an LLM-as-bridge (image $\rightarrow$ text $\rightarrow$ embedding), which was a pragmatic hackathon tradeoff.

5 What's Next

Future work includes replacing the image-to-text bridge with native CLIP embeddings for true cross-modal retrieval, adding user feedback loops to personalize ranking weights, and incorporating listening history to complement vibe-based retrieval.