Vibelens - Image-Based Music Recommendation System

Submitted in partial fulfillment of the
requirements for the degree

of

Bachelor of Science in Computer Science

by

Nguyen Xuan Truong	**V202300998**
Nguyen Tien Nhan	**V202301006**
Le Mai Thanh Son	**V202301128**
Nguyen Dai An	**V202300997**
Nguyen Son Giang	**V202301187**

Under the supervision of

Prof. Pham Huy Hieu
Prof. Kok-Seng Wong

COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

VINUNIVERSITY

June, 2025

[†] These authors contributed equally to this work.
[*] Authors to whom correspondence should be addressed.

DECLARATION OF AUTHORSHIP

Project Title	Vibelens - Image-Based Music Recommendation System
Authors	*Nguyen Xuan Truong* , *Nguyen Tien Nhan*, *Le Mai Thanh Son*, *Nguyen Dai An*, and *Nguyen Son Giang*
Supervisor	Prof. Kok-Seng Wong and Prof. Pham Huy Hieu

We, *Nguyen Xuan Truong* , *Nguyen Tien Nhan*, *Le Mai Thanh Son*, *Nguyen Dai An*, and *Nguyen Son Giang*, hereby declare that this capstone project titled *Vibelens - Image-Based Music Recommendation System* is entirely our own work, unless otherwise referenced or acknowledged. The content of this project is the result of our own research and efforts, and we have complied with all ethical guidelines and academic standards.

We are aware of the consequences of academic dishonesty and understand that any violation of ethical standards in this project may lead to disciplinary actions as defined by VinUniversity's policies.

Nguyen Xuan Truong
V202300998

Nguyen Tien Nhan
V202301006

Le Mai Thanh Son
V202301128

Nguyen Dai An
V202300997

Nguyen Son Giang
V202301187

Hanoi, June, 2025

College of Engineering and Computer Science
VinUniversity
Vinhomes Ocean Park, Gia Lam District, Hanoi

CERTIFICATE

This is to certify that the entitled **Vibelens - Image-Based Music Recommendation System**, submitted by **Nguyen Xuan Truong** (Student ID: V202300998), **Nguyen Tien Nhan** (Student ID: V202301006), **Le Mai Thanh Son** (Student ID: V202301128), **Nguyen Dai An** (Student ID: V202300997), and **Nguyen Son Giang** (Student ID: V202301187) are undergraduate students of the **College of Engineering and Computer Science** has been examined. Upon recommendation by the examination committee, we hereby accord our approval of it as the presented work and submitted report fulfill the requirements for its acceptance in partial fulfillment for the degree of *Bachelor of Science in Computer Science*.

The Committee

Course Instructor
Prof. Pham Huy Hieu

Project Supervisor
Prof. Kok-Seng Wong

Hanoi, June, 2025

ACKNOWLEDGEMENTS

I would like to extend my sincere thanks to our project supervisor, Prof. Kok-Seng Wong, and our course instructor, Prof. Pham Huy Hieu, for their invaluable guidance, constructive feedback, and unwavering support throughout the development of this project.

I would like to thank VinUniversity and the COMP3080 – Course-related Project 1 course team for providing us with the platform and resources to bring our idea to life.

I was lucky to meet many talented peers, mentors, and teaching assistants who offered help, encouragement, and technical suggestions along the way, especially during the implementation and evaluation phases of our system.

Finally, I would like to thank my family and friends for their encouragement, patience, and continuous support that kept us motivated and grounded throughout this journey.

Hanoi, June, 2025

Nguyen Xuan Truong, *Nguyen Tien Nhan*, *Le Mai Thanh Son*, *Nguyen Dai An*, and *Nguyen Son Giang*

To my girl ...
For being my constant source of strength, love, and inspiration throughout this journey.

– Nguyen Xuan Truong

To my girl ...
Your presence brings light to my darkest hours - thank you for walking beside me, always.

– Nguyen Tien Nhan

To my close friends group "Food Safety"
For the unforgettable memories, the unwavering support, and the laughter that made this path brighter and kept me going.

– Le Mai Thanh Son

To my girl ...
Your belief in me never wavered. This milestone is as much yours as it is mine.

– Nguyen Dai An

To my girl ...
For your endless encouragement and quiet faith in me - it made all the difference.

– Nguyen Son Giang

ABSTRACT

In an era where personalized content enhances user engagement, Vibelens introduces a novel approach to music recommendation through image-based emotional and contextual analysis. This AI-powered web application seamlessly connects visual experiences with matching songs by integrating Computer Vision and Natural Language Processing (NLP). When a user uploads an image, the system uses advanced image captioning models to generate hidden descriptive text, which is then semantically compared to a lyrics database using vector similarity techniques. The most contextually and emotionally relevant music is recommended based on this comparison, offering users a highly personalized auditory experience.

To ensure usability and functionality, the system underwent rigorous testing including unit, integration, and user acceptance testing. A user satisfaction survey involving 50 participants revealed overwhelmingly positive feedback, with over 90% rating the experience 4 or 5 out of 5. Key platform features such as image upload, responsiveness, and recommendation quality were particularly well-received.

This project demonstrates the potential of combining multimodal AI technologies for creative applications and opens avenues for further research in emotion-aware computing and cross-modal content retrieval.

Keywords: Artificial Intelligence, Computer Vision, Image Captioning, Music Recommendation System, Natural Language Processing (NLP), Semantic Search

ABBREVIATIONS

AI	Artificial Intelligence
ANN	Approximate Nearest Neighbor
APIs	Application Programming Interfaces
DB	Database
FAISS	Facebook AI Similarity Search
GPT	Generative Pre-trained Transformer
GPU	Graphics Processing Unit
HNSW	Hierarchical Navigable Small Worlds
NLP	Natural Language Processing
SMART	Specific, Measurable, Achievable, Relevant, and Time-bound
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience
ViT	Vision Transformer

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1 Project Background

1.1.1 Literature Review

Recent advancements in artificial intelligence have spurred growing interest in cross-modal recommendation systems that link images to other media types, such as music. Traditional music recommendation approaches have primarily relied on collaborative filtering, metadata analysis, or acoustic feature extraction (Schedl et al., 2018) [1]. While effective in many contexts, these methods are not designed to interpret visual content or capture the emotional intent of an image. At the same time, research in computer vision and natural language processing has led to the development of vision-language models (VLMs), which are capable of generating descriptive captions from images (Ghosh et al., 2024) [2]. These models enable a bridge between visual and textual domains, laying the foundation for image-to-lyrics or image-to-audio alignment. In parallel, vector-based semantic search technologies, including Pinecone and FAISS, have become integral to scalable retrieval systems, allowing high-dimensional comparisons of embeddings derived from sentence transformers (Jie et al., 2023) [3]. A few pioneering studies (e.g., Avramidis et al., 2022 [4]) have explored how visual cues may enhance music recommendation, though practical implementations remain rare and limited in scope. Our project builds upon these foundations to offer an integrated, scalable solution that semantically connects image content to emotionally and contextually relevant songs.

1.1.2 Relevance and Importance

The problem addressed by Vibelens is highly relevant in the context of today's digital and multimedia-driven culture. As visual storytelling becomes central to how users express

themselves on platforms like Instagram, TikTok, and Facebook, there is growing demand for tools that can enhance images with emotionally resonant music. However, the current methods for selecting background music are largely manual, subjective, and inconsistent. This gap is especially pronounced among content creators, social media users, and digital marketers, who would benefit from a system that automatically suggests emotionally aligned songs based on image content.

Our proposed solution offers not only a novel user experience but also practical value in terms of automation, personalization, and creativity enhancement. By integrating computer vision, natural language processing, and semantic search in a unified pipeline, Vibelens can serve as a backend service for social platforms, music streaming apps, or content editing tools. The potential impact is significant: the system can streamline content creation workflows, improve audience engagement through emotionally tailored media, and open new directions for multimodal AI in entertainment, marketing, and digital storytelling.

1.1.3 Current State and Limitations

Despite the rapid advancement of music recommendation systems, most existing solutions rely on collaborative filtering, audio feature extraction, or user metadata to make recommendations. These approaches are commonly seen in platforms like Spotify, YouTube Music, or Apple Music, where suggestions are tailored based on listening history or acoustic similarity. However, such systems do not support content-based recommendations driven by visual input, and therefore cannot align songs with the emotional or semantic meaning of an image.

A few research prototypes and commercial tools have begun exploring multimodal recommendations, but they typically focus on either audio-visual synchronization for video or emotion-based tagging without semantic grounding. Furthermore, no widely adopted system provides image-to-music segment matching with real-time usability and scalable architecture.

Key limitations in the current landscape include:

- Lack of semantic interpretation of visual content in music matching.
- Inability to retrieve specific songs aligned with image meaning.
- Overdependence on subjective tagging or manual curation.

- Lack of personalization when bridging across modalities (image $\rightarrow$ audio).

These shortcomings highlight the need for a scalable, automated, and intelligent solution like Vibelens, which leverages recent advances in vision-language modeling and vector search to address the current gaps.

1.1.4 Conclusion

Given the limitations of current systems and the growing demand for intelligent, emotionally aware multimedia tools, there is a clear opportunity to develop a solution that bridges visual content with music in a meaningful way. Vibelens is designed to address this gap by providing a fully automated, real-time image-to-music recommendation system that captures the emotional and semantic essence of an image and matches it with contextually relevant songs. Building on recent advancements in computer vision, NLP, and semantic vector search, the project aims to deliver a scalable and user-friendly platform that enhances creative expression and content personalization. The following sections outline the technical definition and specific objectives of the Vibelens system.

1.2 Project Definition

1.2.1 Problem Statement

Users on social media and content creation platforms often struggle to find songs that match the emotional and contextual meaning of the images they share, resulting in a manual and subjective selection process which makes the selection process subjective and time-consuming. Currently, there is no intelligent system that can automatically analyze an image, understand its semantic and emotional content, and recommend a set of songs that align with its mood and semantics. This project aims to addresses that gap by developing an AI-powered solution that uses computer vision, natural language processing, and semantic similarity search to provide users with emotionally relevant song recommendations based on uploaded images.

1.2.2 Context and Scope

This project focuses on the development of an AI-based recommendation system that suggests emotionally and semantically relevant songs based solely on a user-uploaded image. The system encompasses key components including image captioning using ViT-GPT2,

sentence embedding using DistilUSE, and semantic similarity search via the Pinecone vector database. The scope includes building a scalable backend infrastructure using Flask, Celery, Redis, and Docker Swarm; implementing a user-friendly frontend with Next.js; and collecting a curated dataset of lyrics and MP3s to support meaningful retrieval.

The system is designed to work in real-time, delivering 5–10 song recommendations per image without requiring user history or manual tagging. However, the project does not cover audio-based feature extraction, real-time audio playback optimization, multilingual lyric interpretation, or emotional classification of music beyond semantic embedding. Additionally, it does not recommend specific song segments, but instead retrieves entire songs based on semantic alignment with the image's generated caption.

By clearly defining these boundaries, the project remains focused on delivering a robust and functional prototype that demonstrates the feasibility of image-to-music recommendation through cross-modal AI techniques.

1.2.3 Significance and Implications

Solving this problem is important because it enables a new form of content personalization that connects the visual and auditory senses in an intelligent, automated way. As digital media consumption grows, users are increasingly seeking tools that can enhance emotional storytelling through music and images. Without a system like Vibelens, users must rely on manual, time-consuming methods to find suitable songs for their visuals - often resulting in mismatched content and reduced emotional impact.

By addressing this gap, Vibelens introduces an innovative solution that can benefit not only casual users and content creators but also social media platforms, music streaming services, and digital marketing tools. It enhances creative workflows, improves user engagement, and sets the foundation for future multimodal AI applications. Failing to explore and develop this cross-modal capability would mean missing a valuable opportunity to push forward personalized media experiences in an era where emotional connection and content relevance are key to audience retention and satisfaction.

1.2.4 Quantification

As of April 2025, there are approximately 5.64 billion internet users and 5.31 billion active social media user identities worldwide, accounting for over 68% of the global population (DataReportal, 2025) [5]. A significant portion of this population regularly engages in image-based and video-based content creation, especially on platforms like Instagram, TikTok, and Facebook. In parallel, the global music streaming market is projected to grow steadily, with revenues expected to exceed USD 30 billion by 2027, indicating strong and sustained user demand for personalized audio content (Exploding Topics, 2024) [6].

Furthermore, Adobe (2025) research highlights the strong impact of music on content performance: nearly 29% of music fans are more likely to purchase a product when they hear a song they like paired with it [7], emphasizing the importance of emotional and contextual alignment between audio and visuals. This supports the need for systems like Vibelens that can automate the song selection process by understanding image semantics and recommending emotionally relevant music, especially for creators and marketers seeking to enhance engagement.

1.3 Project Objectives

The primary objective of this project is to develop an intelligent image-based music recommendation system that seamlessly integrates computer vision, natural language processing, and semantic vector search to recommend emotionally relevant songs based on uploaded images. The project is structured around the following specific, measurable, achievable, relevant, and time-bound (SMART) objectives:

- **Objective 1: Build an End-to-End AI-Powered Recommendation Pipeline**
  - Design and implement a complete system pipeline consisting of image captioning (ViT-GPT2), text embedding (DistilUSE), and vector-based semantic search (Pinecone).
  - Ensure that the system can process an uploaded image and return 5–10 song recommendations within 10 seconds.
  - Achieve functional integration of all modules using Flask (backend), Celery (task queue), and Next.js (frontend).

- **Objective 2: Develop a Scalable Dataset Collection and Storage System**
  - Use Scrapy, Celery, and Redis to crawl and store over 23,000 Vietnamese song lyrics and corresponding MP3s from online sources.
  - Store metadata in PostgreSQL and raw files in MinIO for efficient retrieval and scalability.
  - Set up automated monitoring via Grafana to track crawler status and storage metrics.
- **Objective 3: Create an Intuitive User Interface for Real-Time Image Uploads and Results**
  - Develop a responsive web interface using Next.js that allows users to upload images and receive personalized song recommendations with audio previews.
  - Conduct usability testing and refine the UI based on user feedback to ensure accessibility and smooth interaction.
- **Objective 4: Evaluate System Accuracy and Performance**
  - Measure semantic similarity between image captions and song lyrics using cosine similarity.
  - Target a top-5 recommendation relevance score above 80% based on human evaluation.
  - Maintain system latency under 10 seconds per user query in a deployed environment.

1.4 Project Specifications

To fulfill the project's objectives, the Vibelens system is designed with the following technical specifications and operational requirements:

1.4.1 Technical Requirements

- **Image Captioning:** Use the ViT-GPT2 model to convert uploaded images into semantically rich textual descriptions.
- **Text Embedding:** Apply Sentence Transformers (DistilUSE) to embed both image captions and song lyrics into a common high-dimensional semantic space.

- **Semantic Search:** Utilize Pinecone vector database for fast, scalable similarity search based on cosine distance between embeddings.

1.4.2 System Architecture and Performance

- **Microservices Architecture:** All modules (crawling, preprocessing, captioning, search, frontend) are containerized and deployed using Docker Swarm.
- **Latency:** Ensure a system response time of under 10 seconds from image upload to song recommendation delivery.
- **Scalability:** Backend is built with Flask and orchestrated via Celery and Redis to handle asynchronous task management at scale.
- **Monitoring:** Integrated with Grafana and Prometheus for performance tracking and system observability.

1.4.3 Frontend and User Interaction

- **Framework:** Built with Next.js for a responsive and user-friendly interface.
- **Features:**
  - Drag-and-drop image upload.
  - Recommended songs displayed with embedded audio preview.
- **Accessibility:** Designed for cross-platform use on desktop and mobile browsers.

1.4.4 Data Collection and Storage

- **Crawler:** Built with Scrapy + Celery to gather Vietnamese song lyrics and metadata from websites like Hopamchuan.
- **Audio Downloading:** MP3s downloaded from YouTube-to-MP3 converters and stored in MinIO (S3-compatible storage).
- **Metadata Storage:** PostgreSQL database for storing song titles, artists, lyrics, and URLs.

1.4.5 Functionalities

- Upload an image and generate a hidden caption.
- Retrieve and display 5–10 semantically relevant songs.
- Preview recommended songs directly in the browser.
- Ensure smooth end-to-end experience from image to music.

1.4.6 Compatibility

- Compatible with all major modern browsers (Chrome, Safari, Firefox).
- Backend services deployable on any Linux-based VM (tested on DigitalOcean).
- Supports integration with third-party APIs and cloud databases.

1.4.7 Security Measures

- Input validation and file type checking to prevent malicious uploads.
- Docker-based service isolation to reduce cross-module vulnerability.
- Secured API endpoints with token-based access (optional extension).

Chapter 2

PROJECT MANAGEMENT

2.1 Project Plan

The development of the Vibelens system was organized into a 10-week timeline with clear milestones, deliverables, and role-based task assignments. The plan followed an iterative and modular development approach to ensure scalability, maintainability, and continuous integration of system components. The project plans are outlined below:

2.1.1 Responsibilities

Team Member	Role	Responsibilities
Nguyen Xuan Truong	DevOps Engineer	Deployment pipeline, Docker Swarm, CI/CD, DigitalOcean hosting, logging & monitoring, and design frontend
Nguyen Tien Nhan	Data Engineer	Lyrics crawling, audio downloading, preprocessing, Scrapy-Celery pipeline
Le Mai Thanh Son	Backend Developer	API design, Flask integration, Redis/Celery/Kafka orchestration
Nguyen Dai An	ML/NLP Specialist	Image captioning (ViT-GPT2), embedding (DistilUSE), semantic matching logic
Nguyen Son Giang	Algorithm Specialist	Vector similarity tuning, Pinecone configuration, heuristic testing

Table 2.1: Team Structure and Responsibilities

2.1.2 Milestones

- Completed data crawling pipeline with lyrics and MP3s (Week 3).

- Integrated ViT-GPT2 captioning and sentence embedding (Week 5–6).
- Deployed backend services with Docker Swarm on cloud infrastructure (Week 7).
- Launched working frontend interface with real-time recommendations (Week 8).
- Successfully demonstrated full system functionality from image input to song output (Week 10).

2.1.3 Project's Timeline & Tasks

Week	Milestones / Activities
Week 1	Project planning, literature review, GitHub repository setup, and role assignment
Week 2	Initial web scraper (using requests + BeautifulSoup); setup of basic crawling logic
Week 3	Transition to Scrapy-based asynchronous crawling with Celery and Redis
Week 4	Lyrics collection and MP3 file downloading; audio segmentation into verse/chorus
Week 5	Image captioning module integration using ViT-GPT2; tested on collected image samples
Week 6	Sentence embedding of captions and lyrics using DistilUSE; setup of Pinecone vector DB
Week 7	Backend infrastructure development with Flask, Redis, Celery, Kafka, and PostgreSQL
Week 8	Frontend development using Next.js; initial UI/UX prototype for image upload and song output
Week 9	System integration and internal testing across components (captioning → embedding → search)
Week 10	Final debugging, system optimization, report writing, demo preparation, and deployment

Table 2.2: Week-by-Week Timeline and Tasks

2.2 Contribution of Team Members

The successful implementation of the Vibelens project was the result of strong collaboration and clearly defined responsibilities among all team members. Each individual contributed

based on their technical strengths while also supporting shared tasks such as presentation and report preparation. Below is a breakdown of each member's key contributions:

- **Nguyen Xuan Truong (DevOps Engineer):** Truong led the setup of the entire deployment pipeline, configuring Docker Swarm, CI/CD automation with GitHub Actions, and hosting the backend infrastructure on DigitalOcean. He also managed logging, monitoring, system scalability, and designing the frontend. In addition to his technical role, Truong actively participated in drafting the final report and presentation slides.
- **Nguyen Tien Nhan (Data Engineer):** Nhan developed the distributed web crawler using Scrapy, Celery, and Redis to collect lyrics and audio files. He also implemented preprocessing logic and ensured the integrity of the MP3–lyrics alignment. His work laid the foundation for data ingestion and indexing. Nhan also contributed to the slides and supported the discussion during the final presentation.
- **Le Mai Thanh Son (Backend Developer):** Son built and integrated the RESTful APIs for image upload, caption generation, and music recommendation. He also ensured smooth communication between frontend and backend modules. Son also took the lead in writing and polishing the final report, helped manage the presentation materials, and was primarily responsible for designing and conducting the user satisfaction survey to evaluate the system's usability and relevance.
- **Nguyen Dai An (ML/NLP Specialist):** An was responsible for integrating the ViT-GPT2 image captioning model and converting outputs into vector embeddings using DistilUSE. He contributed significantly to the semantic matching pipeline and helped tune system performance to ensure accurate recommendations. An also supported testing and debugging across all components and participated in the slide preparation and presentation.
- **Nguyen Son Giang (Algorithm Specialist):** Giang designed and optimized the vector search mechanism using Pinecone and cosine similarity metrics. He focused on tuning the similarity threshold to maximize relevance in the returned song list. Giang played a central role in writing technical sections of the final report and worked closely with Son and Truong to refine documentation and presentation materials.

All team members were actively involved in preparing the final presentation and contributed to creating the demonstration slides. This shared effort reflects the collaborative

spirit of the project and highlights the interdisciplinary skill sets required to deliver a fully functional and user-centered AI system.

2.3 Challenges and Decision Making

2.3.1 Key Challenges

- **Infrastructure Constraints:** Due to budget limitations, the team could not afford dedicated GPU-based cloud servers. As a result, all model inference (e.g., ViT-GPT2 for image captioning) had to be optimized for CPU performance or offloaded to free-tier APIs. This required several rounds of testing and simplification of model configurations to maintain acceptable latency.
- **Multi-Service Integration Complexity:** Integrating the backend components - Flask API, Redis, Celery workers, Kafka streaming, Pinecone, and PostgreSQL - within a Docker Swarm environment introduced many version conflicts and inter-service communication bugs. These issues were especially prominent during deployment to DigitalOcean, where logging and debugging were more limited compared to local setups.
- **Data Quality and Alignment Issues:** Scraped lyrics often included irrelevant tags, broken lines, or duplicate content, and many MP3s did not align well with lyrics. This required additional preprocessing heuristics, such as rule-based cleaning scripts and fallback mechanisms when chorus detection failed.
- **Frontend-Backend Asynchrony:** During early integration, inconsistencies between frontend requests and backend response structures caused API failures. Careful endpoint documentation and schema validation were introduced to resolve these mismatches.

2.3.2 Decision-Making Process

To address these challenges, the team adopted a flexible and collaborative decision-making model:

- **Technical decisions** (e.g., switching to Pinecone for vector search instead of a local FAISS implementation) were made based on benchmark tests and discussed during weekly sync meetings.

- **Task reallocation** was handled dynamically. For instance, when deployment took longer than planned, Truong temporarily shifted efforts toward backend debugging while other members took the lead in drafting the report.
- **Feature prioritization** was guided by impact and feasibility. Optional features like sentiment-based ranking or real-time audio trimming were dropped in favor of stabilizing the core pipeline.

Decisions were usually made through group consensus, often facilitated via group chat discussions. This agile and transparent approach ensured that every challenge became an opportunity for deeper collaboration and a better-designed system.

Chapter 3

SYSTEM DESCRIPTION

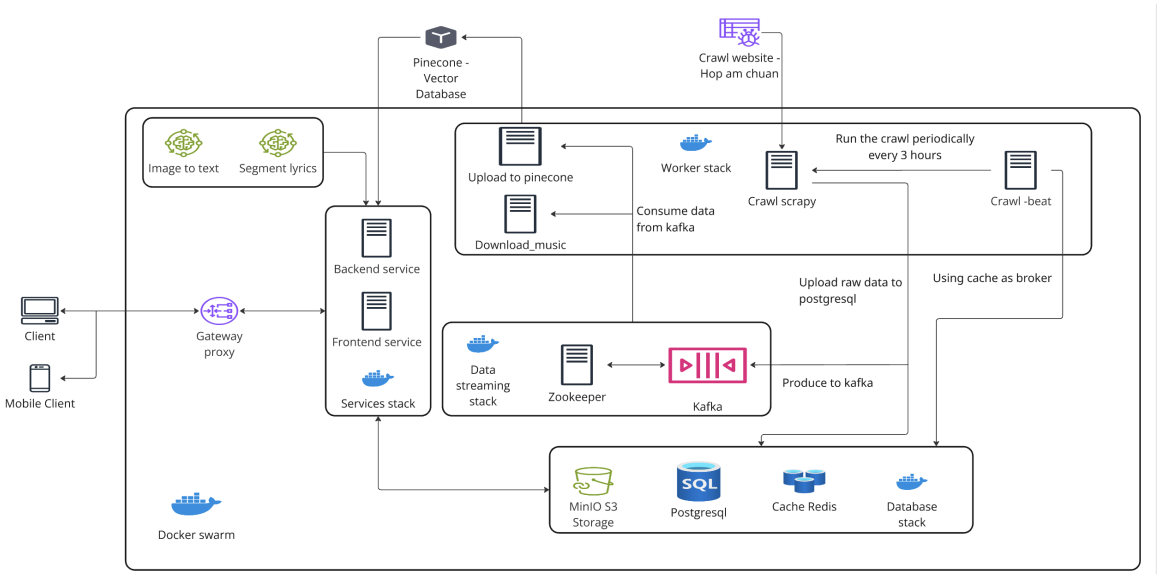

Figure 3.1: Distributed Microservice Architecture of Vibelens with Docker Swarm Orchestration

3.1 Block Diagram of the System

Vibelens is an image-based music recommendation system that utilizes machine learning and vector similarity to recommend music content based on user-uploaded images. The system is deployed in a microservices architecture managed via **Docker Swarm** and **Portainer**, and the major blocks are organized into five key stacks: Client Gateway, Service Stack, Worker Stack, Streaming Stack, and Database Stack.

- **Client and Gateway Proxy:** Both web and mobile clients interact with the system

through a unified entrypoint handled by **NGINX Proxy Manager**. It provides routing, SSL, domain management, and forwards requests to internal microservices.

- **Service Stack:**
  - The **Backend Service** (built with Flask) performs key inference tasks: transforming images to text using a computer vision model, and segmenting lyrics for semantic processing. These vectors are pushed to Pinecone for similarity search. It also handles integration with MinIO for object storage and PostgreSQL for metadata.
  - The **Frontend Service** is built with Next.js, compiled into a static site and served by NGINX. It provides a responsive interface for users to upload images, view recommendations, and play music.
- **Worker Stack:**
  - **Crawl Scrapy**: Executes every 3 hours (scheduled by **Crawl-beat**) to scrape music metadata from target websites (e.g., Hop Am Chuan). It uses Redis as a broker to manage job distribution.
  - **Upload-to-Pinecone Worker**: Consumes Kafka messages, processes lyrics into vectors, and uploads them to Pinecone.
  - **Download Music Worker**: Downloads music based on links obtained from crawling and stores it in MinIO.
- **Data Streaming Stack:**
  - **Apache Kafka** decouples crawling and processing stages, acting as a buffer and stream handler.
  - **Zookeeper** coordinates Kafka nodes, ensuring broker health and topic partitioning.
- **Database Stack:**
  - **PostgreSQL** stores structured metadata such as song titles, tags, crawl timestamps, and search logs.
  - **Redis** is used as both a high-performance cache for query acceleration and a broker for Scrapy.

- **MinIO**, a lightweight S3-compatible service, stores all user images and downloaded audio.

3.2 Design of Each Block and Select the Best Alternative

3.2.1 Client and Gateway Proxy

- **NGINX Proxy Manager** was chosen for managing all ingress traffic, offering a web UI, automatic SSL provisioning (via Let's Encrypt), and native Docker integration. Alternatives like Traefik were evaluated, but NGINX Proxy Manager provided a simpler setup and ease of use.

3.2.2 Backend Service: Flask

- **Flask** was selected for its simplicity, minimal dependencies, and compatibility with CPU-based PyTorch inference workloads.
- Compared to Django (too heavy) and FastAPI (more suitable for async and real-time), Flask enabled faster development and direct integration of the inference models.
- The image-to-text and lyrics-segmentation models are run inside this service and are exposed via REST endpoints.

3.2.3 Frontend Service: Next.js

- **Next.js** enables static site generation and server-side rendering, improving SEO and load speed.
- It was chosen over Angular and Vue.js due to its rich plugin ecosystem, React compatibility, and developer experience.

3.2.4 Worker Stack Design

- **Scrapy** was chosen for its mature ecosystem and support for asynchronous crawling. Redis was used as the broker for Crawl-beat tasks due to its speed and simple deployment.
- BeautifulSoup and Selenium were rejected for lacking either async I/O or requiring full browser environments.

- Kafka workers (Python consumers) were designed to support batch consumption, checkpointing, and stateless retry logic.

3.2.5 Crawling Interval Justification (3 Hours)

The Scrapy crawler is scheduled to run every **3 hours** using Crawl-beat. This interval was selected based on a combination of ethical scraping practices, system throughput capacity, and change frequency of the target content. The reasons are as follows:

- **Server Courtesy and Ethical Scraping:** Target sites such as Hop Am Chuan are community-driven and not intended to serve high-frequency automated requests. Crawling too often may place undue stress on their infrastructure or violate their usage policies. A 3-hour interval ensures that the system remains respectful and minimizes the risk of negative impact.

- **Avoiding Bot Detection:** Repeated and rapid requests can trigger bot detection mechanisms such as CAPTCHAs, IP bans, or throttling. By spacing out the crawling interval to every 3 hours and using rotating headers and crawl delays, we reduce the chance of being flagged or blocked by the server.

- **Content Update Rate:** Music metadata (e.g., chords, lyrics, song titles) does not change rapidly over short time frames. A 3-hour schedule aligns well with the typical content update frequency of such platforms, ensuring data freshness without redundant crawling.

- **Resource Management:** Crawling involves concurrent tasks such as network I/O, parsing, and dispatching data to Kafka and PostgreSQL. Frequent execution could strain system resources, particularly when handling multiple sites or larger datasets. A longer interval allows time for the system to process previously fetched data before initiating new jobs.

- **Downstream Load Balancing:** Workers such as *Upload-to-Pinecone* and *Download Music* consume Kafka topics generated by the crawler. Maintaining a 3-hour gap prevents message queue backlogs and supports stable downstream processing throughput.

This interval represents a balance between timely data acquisition, ethical behavior, and infrastructure efficiency. It can be fine-tuned in the future based on load testing, user demand, or site-specific characteristics.

3.2.6 Data Streaming Stack

- **Kafka** was selected over Celery and RabbitMQ for its streaming-first design, built-in replayability, and better support for decoupling producer-consumer logic.
- **Zookeeper** provides leader election and topic partitioning management for Kafka. It also assists in failover and log compaction.

3.2.7 Database Stack

- **PostgreSQL** was chosen over MySQL due to superior indexing, full-text search capabilities, and JSONB support.
- **Redis** is used for two purposes: broker in the crawler, and cache layer for fast reads and frequent lookups.
- **MinIO** supports S3-compatible APIs and offers fast, private object storage for images and MP3s.

3.2.8 Container Orchestration: Docker Swarm + Portainer

- **Docker Swarm** provides lightweight orchestration, service discovery, and rolling updates.
- **Portainer** adds a visual management layer, simplifying monitoring and scaling tasks.
- This combination was preferred over Kubernetes, which would have introduced more operational complexity.

3.3 Testing of Each Block

3.3.1 Backend Testing

- **Pytest** was used to test all functions including:
  - Image-to-text processing
  - Lyrics segmentation
  - Pinecone and MinIO interactions
- Tests covered both success and failure cases, e.g., missing files, unsupported image formats, and Pinecone indexing errors.

3.3.2 Frontend Testing

- Manual testing using Chrome/Firefox was used to verify compatibility.
- Lighthouse evaluated accessibility, performance, SEO, and PWA readiness.
- No Cypress, Jest, or end-to-end automation was used.

3.3.3 Worker Stack Testing

- **Scrapy spiders** were tested against a local mirror of target websites to validate scraping logic and item pipelines.
- **Kafka consumers** were tested with mock messages and verified using debug logs and checkpoints.
- Edge cases like missing fields or invalid links were handled with retry logic and dead-letter queues.

3.3.4 Data Streaming Testing

- Kafka topics were monitored using Kafka UI dashboards.
- Test producers sent sample payloads to simulate real-time data flow.
- **Zookeeper** cluster status and leader elections were inspected using JMX tools and internal Kafka logs.

3.3.5 Database and Storage Testing

- **PostgreSQL** was tested using SQL queries and migration scripts. Data validation included join correctness and foreign key constraints.
- **Redis** was tested using 'redis-cli' to inspect keys, expiration, and list queues.
- **MinIO** file uploads and downloads were tested via Postman and 'boto3' to simulate pre-signed URL access and content validation.

3.3.6 Performance and Load Testing

- **Locust** simulated concurrent users uploading images and requesting music recommendations.
- Metrics gathered include request throughput, latency distribution, and memory/CPU usage.
- **Postman** collections were executed in CI to validate API stability.

3.4 Algorithms and Data Modeling Approaches in a Music Recommendation System

This section presents the dual-database architecture and algorithmic methods adopted in the design of a content-based music recommendation system. The system aims to (1) provide efficient and intelligent semantic search over lyrics and metadata, and (2) maintain accurate, structured data management for crawled music content. To achieve these objectives, we integrate two complementary databases - **PostgreSQL** for structured data storage and **Pinecone** for vector-based search - and tailor a distinct algorithm to each.

3.4.1 Semantic Vector Search Using Pinecone and HNSW

To support intelligent recommendations based on the meaning of lyrics, song titles, or genres, each song is represented as a high-dimensional embedding vector generated by a pre-trained language model. These vectors are indexed and stored in Pinecone, a vector database designed for high-performance similarity search.

To perform approximate nearest neighbor (ANN) search efficiently in high-dimensional space, we employ the *Hierarchical Navigable Small Worlds* (HNSW) algorithm. HNSW organizes data into a layered graph structure in which each level contains a sparse subset of the entire dataset. The top layers allow large jumps between distant points to quickly narrow down the search region, while the lower layers focus on local refinement. The search process begins at a fixed entry point in the uppermost layer and traverses downward, progressively approaching the most relevant neighbors based on vector similarity (typically cosine distance).

This hierarchical graph structure ensures low-latency search performance even as the dataset scales, making it well-suited for real-time recommendations. In this project, HNSW enables the system to retrieve semantically similar songs based on user input, such as a lyric snippet or keyword query.

The overall semantic search pipeline is as follows: upon receiving a user query (e.g., a fragment of lyrics or descriptive keywords), the system first converts the input into a dense vector embedding using a pre-trained language model. This query vector is then passed to Pinecone, which compares it with existing song vectors using cosine similarity as the similarity metric. The system retrieves the top $K = 10$ most similar vectors and ranks them in descending order of similarity. These top-ranked vector IDs are then used to fetch the full metadata - such as title, artist, lyrics, and URL - from the PostgreSQL database, enabling accurate and context-aware song recommendations.

Figure 3.2 provides a simplified illustration of the HNSW indexing mechanism. It shows how a query vector navigates through different graph layers to locate semantically relevant vectors.

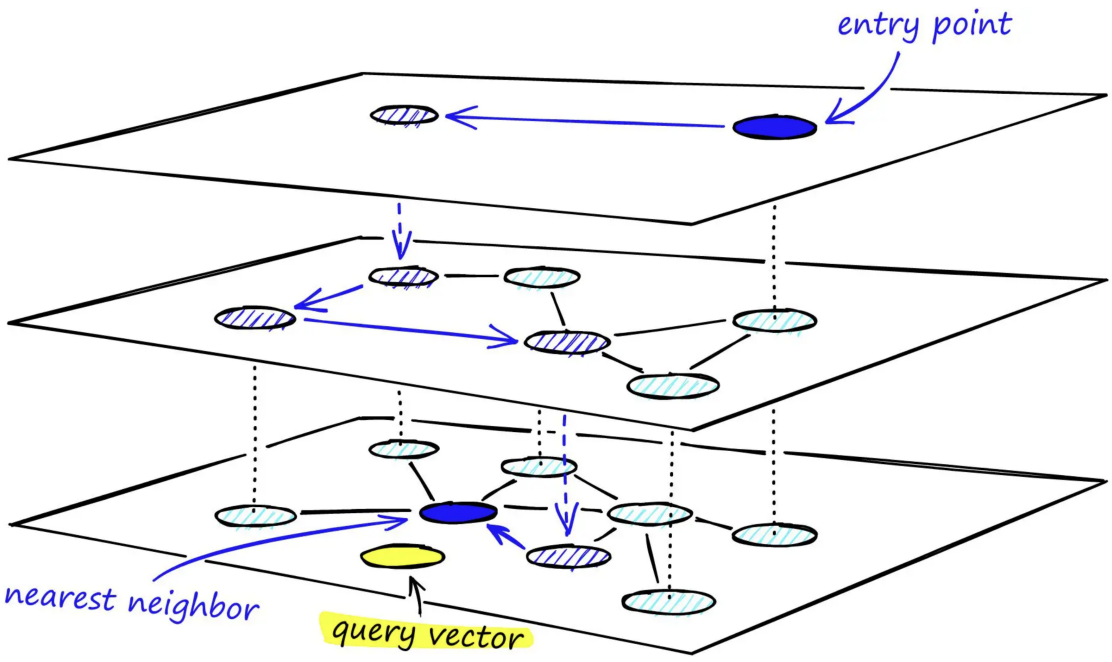

Figure 3.2: HNSW indexing structure used in Pinecone for vector similarity search

3.4.2 Structured Storage and Deduplication with PostgreSQL and B-Tree Indexing

While Pinecone supports similarity-based retrieval, PostgreSQL serves as the backbone for managing structured and reliable data. It stores the raw metadata associated with each song - including the title, artist, lyrics, source URL, and other attributes - as originally crawled from the web.

To ensure data integrity and prevent duplication, each song's URL is passed through a deterministic hash function to produce a unique identifier. These identifiers are indexed using B-tree structures native to PostgreSQL. Upon insertion, the system checks whether a record with the same hashed ID already exists; if it does, the new entry is discarded. This strategy ensures efficient and consistent deduplication while allowing for exact-match queries using standard SQL syntax such as `SELECT`, `WHERE`, and `JOIN`.

PostgreSQL thus provides the infrastructure for high-precision metadata management, facilitating tasks such as crawl monitoring, data cleaning, and audit logging.

Figure 3.3 illustrates how hashed IDs are stored and searched using a B-tree indexing structure. It also highlights the deduplication mechanism based on URL hashing.

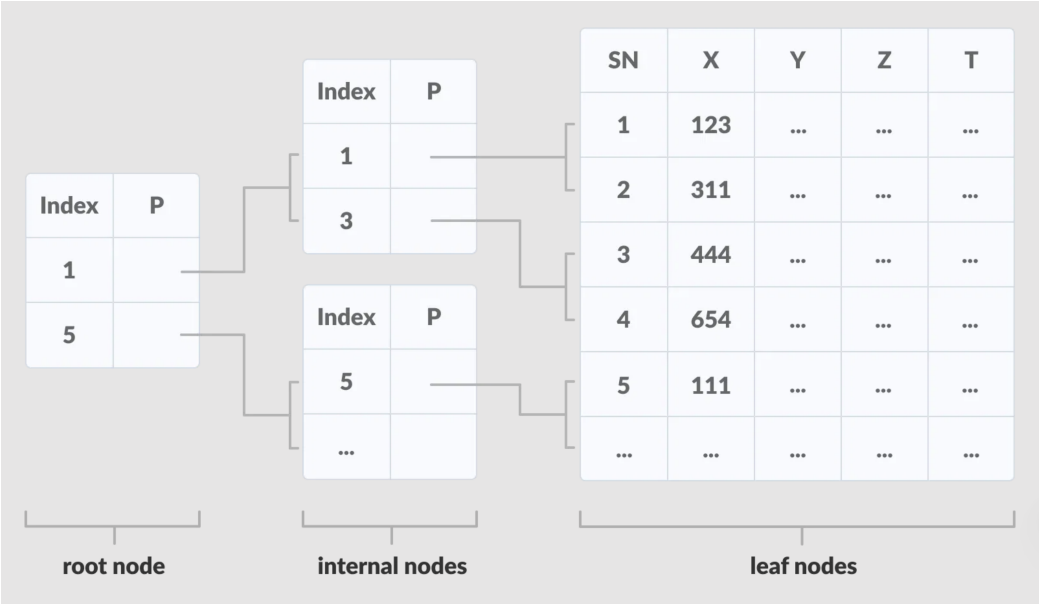

Figure 3.3: Hashing and B-tree indexing in PostgreSQL for deduplication and fast lookup

3.4.3 Comparison Between PostgreSQL and Pinecone

Table 3.1 summarizes the distinct strengths and limitations of PostgreSQL and Pinecone within the system, illustrating the rationale for combining them to meet complementary requirements.

Objective	**PostgreSQL**	**Pinecone (Vector DB)**
Storage of structured data (e.g., title, artist, URL, lyrics)	Suitable for full structured storage	Not designed for complex metadata; only stores vectors and simple fields
Exact data lookup (e.g., URL deduplication)	Supports precise querying with SQL logic	Not optimized for exact matching
Deduplication	Easily implemented via hash-based constraints and B-tree indexing	No native support for logical constraints
Semantic content search	Not applicable	Specifically designed for vector similarity search
AI-based music recommendation using embeddings	Cannot compute nearest vectors	Optimized for K-nearest-neighbor and cosine similarity
Query performance	Optimized for transactional and relational queries	Optimized for high-dimensional vector retrieval
Role in system design	Manages raw metadata and structured crawl results	Powers embedding-based recommendation and intelligent retrieval

Table 3.1: Functional Comparison Between PostgreSQL and Pinecone

3.4.4 System Design Implications

The dual-database strategy reflects a separation of concerns. PostgreSQL ensures high-fidelity storage and reliable query operations over crawled data, while Pinecone provides low-latency retrieval of semantically similar content. By aligning each database with a specific set of tasks and pairing it with an appropriate algorithm (HNSW for semantic search and B-tree for indexing hashed records), the system balances performance, scalability, and functional clarity.

This hybrid approach allows the recommendation engine to operate intelligently without compromising on data integrity, offering both precision in storage and flexibility in retrieval.

3.5 System Implementation

The Vibelens system was implemented through a series of carefully orchestrated development phases that translated the conceptual design into a fully operational and scalable application. The implementation process focused on modularity, efficiency, and real-world deployment considerations to ensure robustness and extensibility.

3.5.1 Backend and API Layer

The backend was developed using Flask, providing RESTful APIs for handling image uploads, caption generation, semantic vector search, and music recommendation. Celery and Redis were used for asynchronous task processing, particularly for resource-heavy operations like embedding and crawling. Kafka was integrated as a message broker to enable scalable data flow between services, and PostgreSQL was used to store song metadata, captions, and image logs.

3.5.2 Frontend Development

The frontend interface was built using Next.js, offering a responsive and intuitive user experience. It allows users to upload an image and view song recommendations with hidden captions and an embedded audio player. The interface was designed to be mobile-friendly and optimized for real-time interaction with backend services via REST APIs.

3.5.3 Image Captioning and Embedding

To extract semantic meaning from uploaded images, the team integrated ViT-GPT2 for image captioning. The generated captions were then passed through DistilUSE (Universal Sentence Encoder) to create high-dimensional vector embeddings. These embeddings represent the semantic meaning of the image and are used for comparison against pre-embedded lyrics.

3.5.4 Vector Database and Semantic Search

The lyric corpus was embedded in advance using the same sentence transformer and stored in Pinecone, a cloud-based vector similarity search engine. When a caption is embedded at runtime, a cosine similarity search is performed against the vector database to return the top 5–10 most relevant songs.

3.5.5 Crawling and Data Storage

A distributed web crawler built with Scrapy + Celery + Redis was used to collect lyrics from Vietnamese music websites and corresponding MP3 files from YouTube. All metadata was stored in PostgreSQL, while audio files were managed using MinIO, an S3-compatible object storage system. A real-time monitoring system was added using Grafana, allowing the team to track crawler performance and resource usage.

3.5.6 Deployment and Scalability

The entire system was containerized using Docker, with services orchestrated via Docker Swarm for easy horizontal scaling. The deployed environment on DigitalOcean ensured that all services ran reliably with isolated containers, while GitHub Actions was used for CI/CD automation to streamline updates and maintain consistency across deployments.

3.5.7 Real-World Considerations

Due to infrastructure limitations, model inference was optimized for CPU usage where possible, and non-essential features (e.g., sentiment analysis of lyrics) were deferred to prioritize core functionality. The system was tested across browsers and devices to ensure compatibility and resilience in real-world usage scenarios.

This modular, cloud-ready architecture ensures that Vibelens is scalable, maintainable, and extensible - ready for potential future enhancements such as multilingual support, emotion classification, or integration with social media platforms.

Chapter 4

SYSTEM TESTING AND ANALYSIS

This chapter outlines the testing strategies, tools, and procedures used to evaluate the functionality, reliability, and performance of the Vibelens system. Given the modular nature of the system, a mix of unit testing and integration testing was performed to ensure technical robustness and seamless interaction between components.

4.1 Testing Methodology

To validate the correctness and stability of Vibelens, we followed a structured testing methodology:

- **Unit Testing**: Each software module - including the image captioning pipeline, embedding model, and retrieval functions - was tested individually using `pytest`. These tests ensured that each component functioned correctly in isolation.
- **Integration Testing**: The full system workflow was tested to ensure smooth interaction between services. From image input to semantic embedding and song recommendation retrieval, all modules were tested together using manual tests and automated tools like Postman and Locust. This helped verify data integrity, request-response flow, and system compatibility.

4.2 Testing Results and Analysis

The table below summarizes the testing results for each key module of the Vibelens system:

Test Type	Tools Used	Key Metrics / Results
Unit Testing	`pytest`, `unittest`	All core modules passed their individual test cases
Integration Testing	Manual, Postman, Locust	System operated correctly with consistent output under 100+ requests
Performance Benchmark	Locust	Average end-to-end latency: 10s, no crashes

Table 4.1: Summary of testing methods and key technical results.

4.3 User Satisfaction Survey & Analysis

To evaluate the effectiveness, usability, and overall user satisfaction of the Vibelens system, we conducted a comprehensive user survey targeting 50 respondents. These individuals were given full access to the platform and asked to evaluate their experience across four key areas:

- Image Upload & Usability
- Music Recommendation Quality
- User Interface & Experience
- Overall Satisfaction & Feedback

This section presents an analysis of the collected responses, providing valuable insights into user perceptions and identifying potential areas for improvement.

4.3.1 Image Upload & Usability

4.3.1.1 Ease of Image Upload Process

To assess the usability of the image upload feature on the Vibelens platform, we asked 50 users the following question: **"How easy was it to upload an image on the website?"**. Users rated their experience on a linear scale from 1 to 5, where 1 represented "Very difficult" and 5 represented "Very easy".

Results: The feedback was overwhelmingly positive as shown in Figure 4.1:

- 66% of users (33 out of 50) rated the upload process as "very easy" (5/5).

- 26% gave it a 4/5, indicating it was mostly seamless with minor room for improvement.
- Only 8% of users rated it 3 or lower, suggesting minimal usability concerns.

This indicates that the majority of users found the image upload feature intuitive and efficient. The user interface for uploading was clearly designed with simplicity in mind, and it successfully handled user interactions smoothly across different devices.

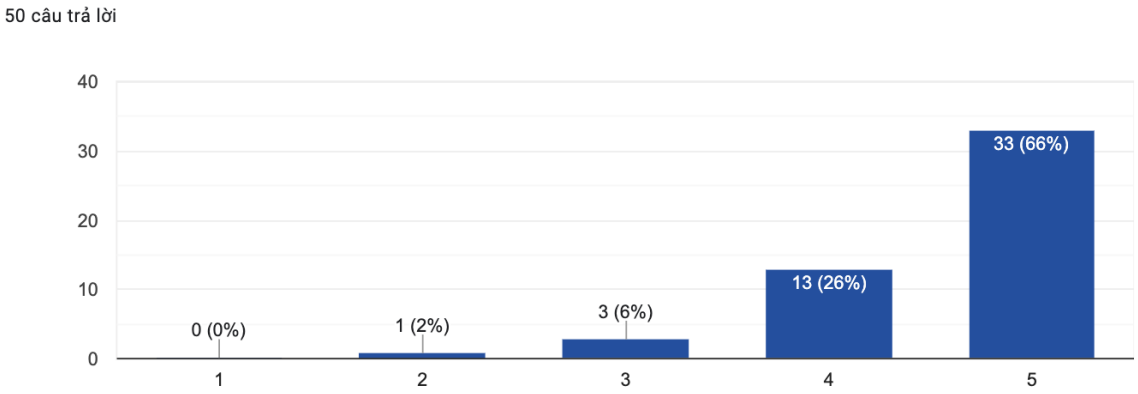

Figure 4.1: How easy was it to upload an image on the website?

4.3.1.2 Platform Responsiveness During Upload

To further validate overall platform responsiveness, we also asked: **"Was the website responsive and smooth during your experience?"**. Users rated their experience on a linear scale from 1 to 5, where 1 represented "Very laggy or unresponsive" and 5 represented "Very smooth".

Results:

- 56% (28 users) rated the responsiveness 4/5.
- 30% gave it a perfect 5/5, while only 14% rated it 3 or below.

These results confirm that users experienced a responsive and smooth platform during image upload, further strengthening the case for strong usability in the core features of Vibelens as shown in Figure 4.2.

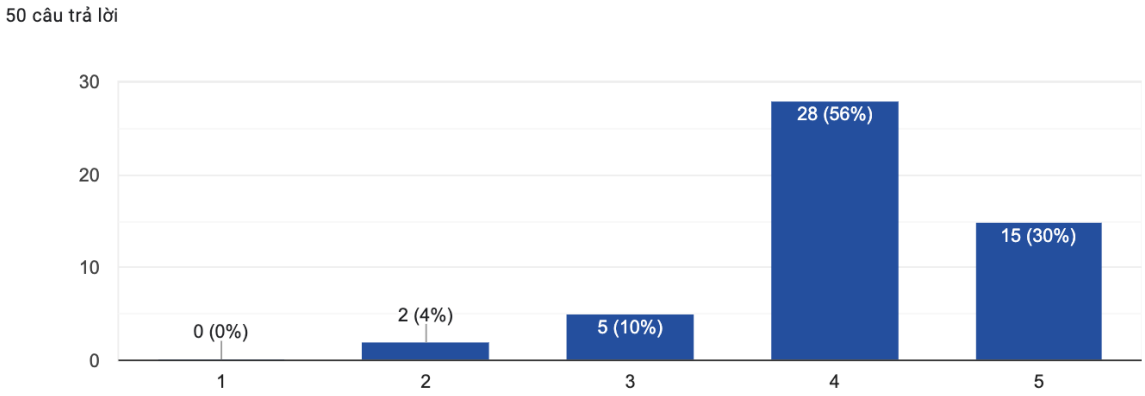

Figure 4.2: Was the website responsive and smooth during your experience?

4.3.2 Music Recommendation Quality

To assess the core functionality of Vibelens - recommending emotionally and contextually appropriate songs based on uploaded images - we gathered responses from 50 users through a structured survey. This subsection presents the user feedback regarding the accuracy, relevance, and sufficiency of the music recommendation engine.

4.3.2.1 Relevance to Image Content

Participants were asked: **"How relevant were the recommended songs to the content of your uploaded image?"** Users rated their experience on a linear scale from 1 to 5, where 1 represented "Completely irrelevant" and 5 represented "Very relevant".

- 66% of respondents rated the content-based relevance as 5/5, indicating strong alignment between the songs and the visual themes of their images.
- Another 26% selected 4/5, while only a small minority (8%) rated it lower.

This overwhelmingly positive response suggests that Vibelens performs well in extracting key visual elements from images and successfully mapping them to appropriate lyrical or thematic content in songs as shown in Figure 4.3.

4.3.2.2 Relevance to Image Emotion

The next question addressed the emotional dimension: **"How relevant were the recommended songs to the emotion of your uploaded image?"** Users rated their experience on

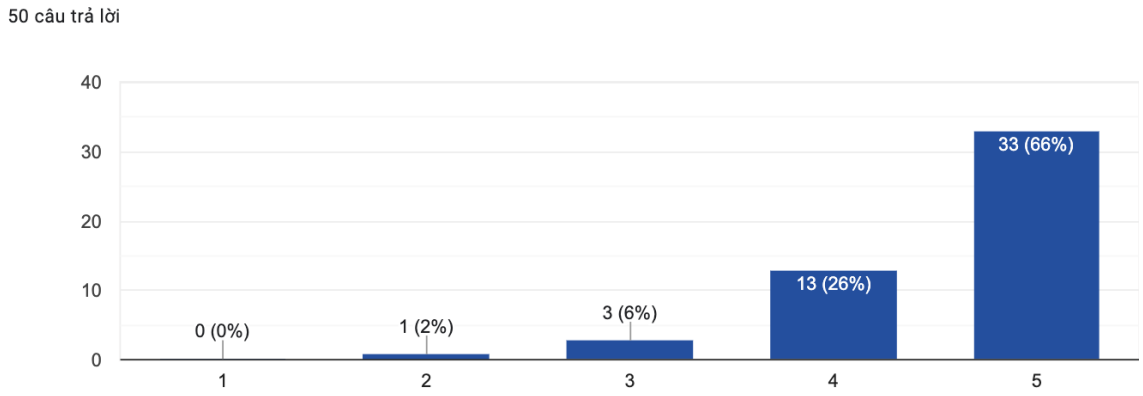

Figure 4.3: How relevant were the recommended songs to THE CONTENT of your uploaded image?

a linear scale from 1 to 5, where 1 represented "Completely irrelevant" and 5 represented "Very relevant".

- A majority of users (52%) selected 3/5, suggesting that while the system could identify general emotions, the emotional alignment was perceived as moderate.
- 24% rated this aspect 4/5, and only 14% gave the highest rating of 5/5.

Compared to content relevance, emotional matching showed room for improvement. This feedback highlights the challenge of mapping complex human emotional perception to computational models and may inform future upgrades to Vibelens' emotion recognition module as shown in Figure 4.4.

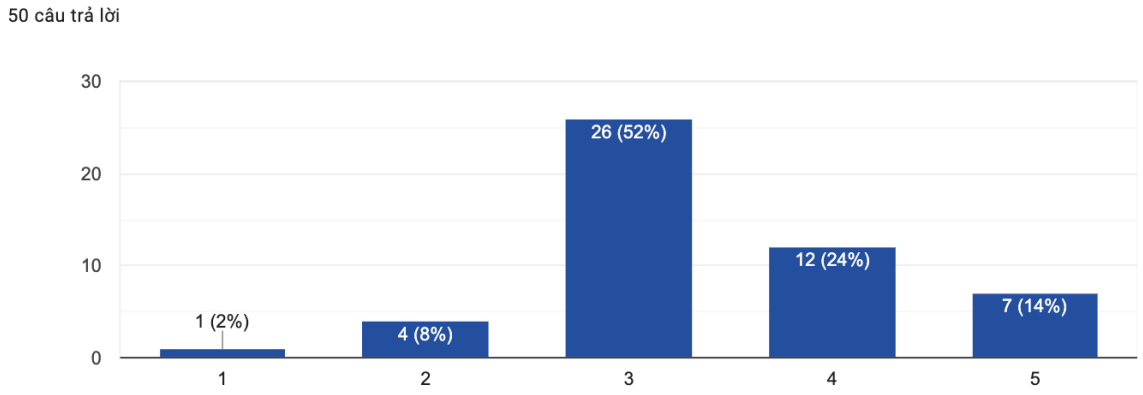

Figure 4.4: How relevant were the recommended songs to THE EMOTION of your uploaded image?

4.3.2.3 Satisfaction with Song Quantity

Finally, users were asked: **"Were you satisfied with the number of songs recommended?"** Users rated their experience on a linear scale from 1 to 5, where 1 represented "I didn't receive any" and 5 represented "Very satisfied".

- 56% of respondents rated their satisfaction at 4/5, while 30% gave a perfect 5/5.
- A small fraction of users (14%) rated this metric between 2–3, citing a desire for either more variety or tailored curation.

Overall, users were generally satisfied with the volume of recommendations, suggesting the current output size strikes a reasonable balance between conciseness and diversity as shown in Figure 4.5.

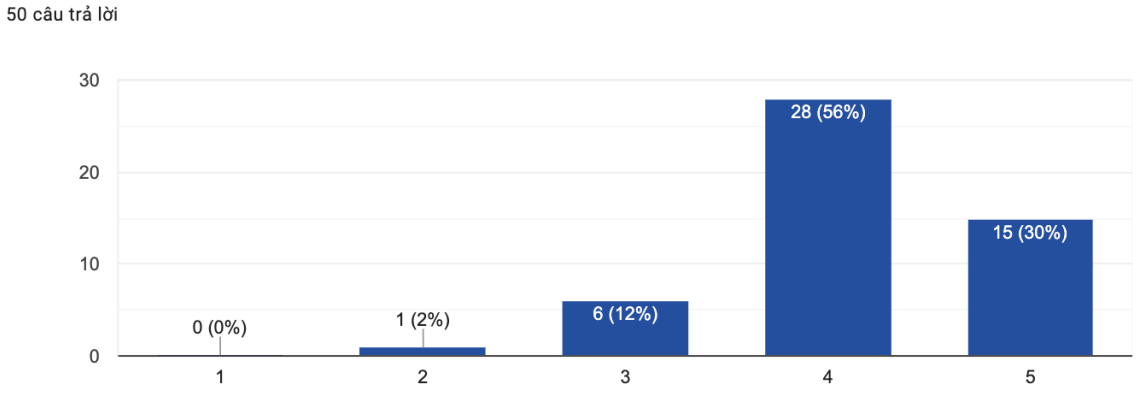

Figure 4.5: Were you satisfied with the number of songs recommended?

4.3.3 User Interface & Experience

This subsection evaluates the usability, visual design, and responsiveness of the Vibelens platform, based on survey responses from 50 users. The goal was to assess how intuitive and enjoyable the interface was during the music recommendation process, including both aesthetic appeal and operational smoothness. Insights from this evaluation offer valuable feedback for refining the user journey.

4.3.3.1 Overall User Interface Design

Users were asked to rate the overall design and user interface of the website on a scale from 1 (very poor) to 5 (excellent). As illustrated in the Figure 4.6, a significant majority of users expressed high satisfaction:

- 70% of users rated the UI as 5 - Excellent.
- 28% rated it as 4 - Good.
- Only 1 user (2%) rated it as 3, and none rated it 1 or 2.

This resulted in a high average rating of 4.68/5, indicating that the interface was both visually appealing and easy to navigate.

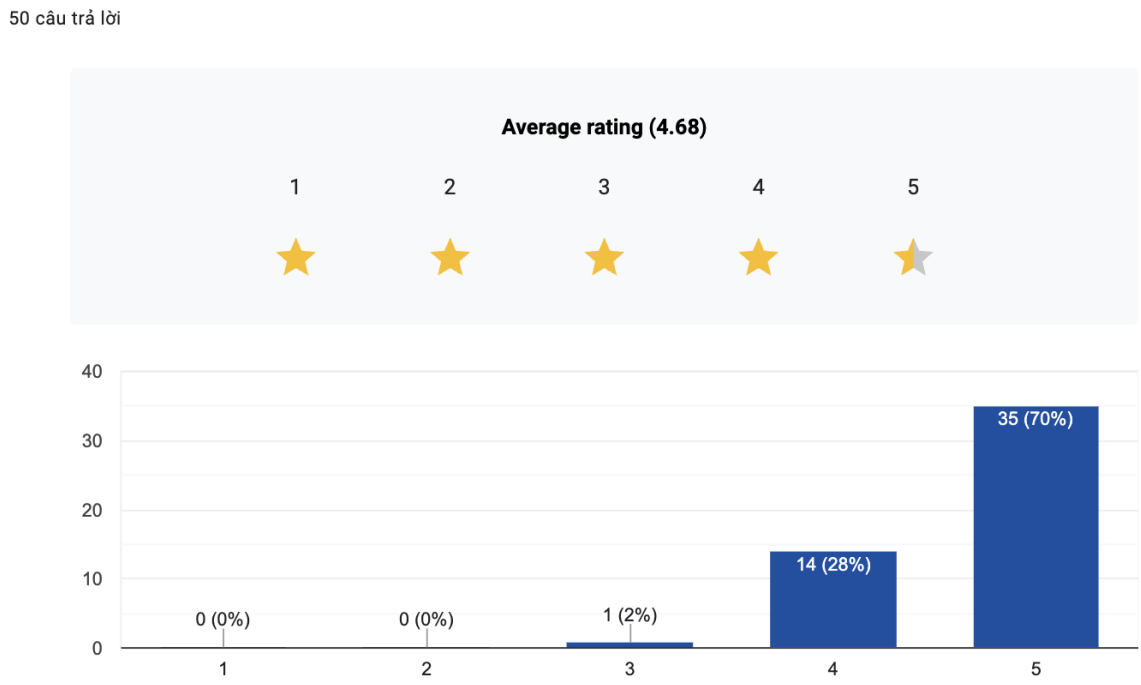

Figure 4.6: How would you rate the overall design and user interface of the website?

4.3.3.2 Responsiveness and Wait Time

To measure operational performance, users were asked: **"Was the wait time reasonable before the songs appeared?"** Users rated their experience on a linear scale from 1 to 5, where 1 represented "It didn't work" and 5 represented "Very fast".

The responses were generally positive:

- 50% rated the wait time as 4, and
- 32% gave it a 5, affirming the experience was fast and acceptable.
- Only 3 users (6%) expressed lower satisfaction (ratings of 2 or 3).

These results confirm that the backend system responded efficiently, with a majority of users encountering minimal delay in recommendation delivery as shown in Figure 4.7.

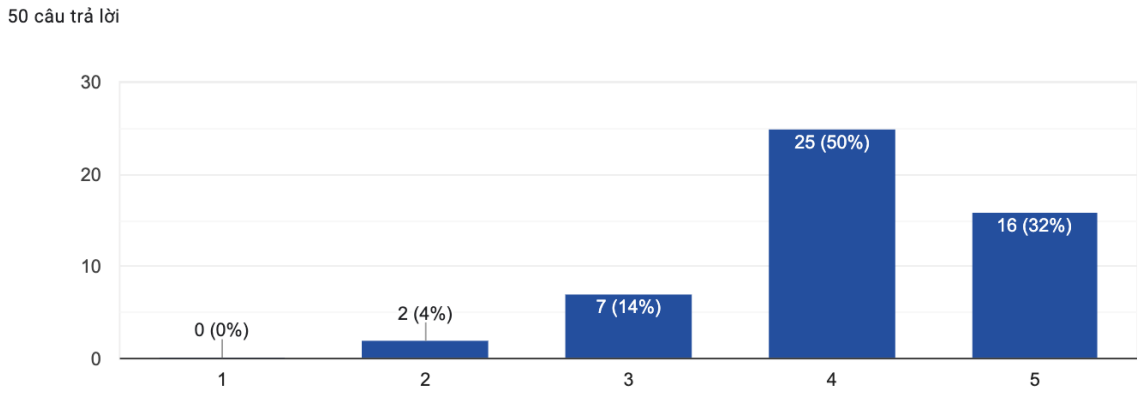

Figure 4.7: Was the wait time reasonable before the songs appeared?

4.3.4 Overall Satisfaction & Feedback

This subsection consolidates user sentiment on their overall experience with Vibelens. It aims to capture how satisfied users were with the platform as a whole and their likelihood of continued or referred usage. These insights provide crucial indicators of product-market fit and long-term engagement potential.

4.3.4.1 Overall Experience Satisfaction

When asked **"Overall, how satisfied are you with your experience using Vibelens?"**, the results were largely favorable. Users rated their experience on a linear scale from 1 to 5, where 1 represented "Very dissatisfied" and 5 represented "Very satisfied".

- 56% of users gave a rating of 5 – Very satisfied,
- 34% rated it 4 – Satisfied, and
- Only 1 user (2%) and 4 users (8%) rated it 2 and 3, respectively.

No users selected 1 (very dissatisfied), which suggests a solid baseline experience. The satisfaction trend highlights the effectiveness and appeal of the platform from a holistic perspective as shown in Figure 4.8.

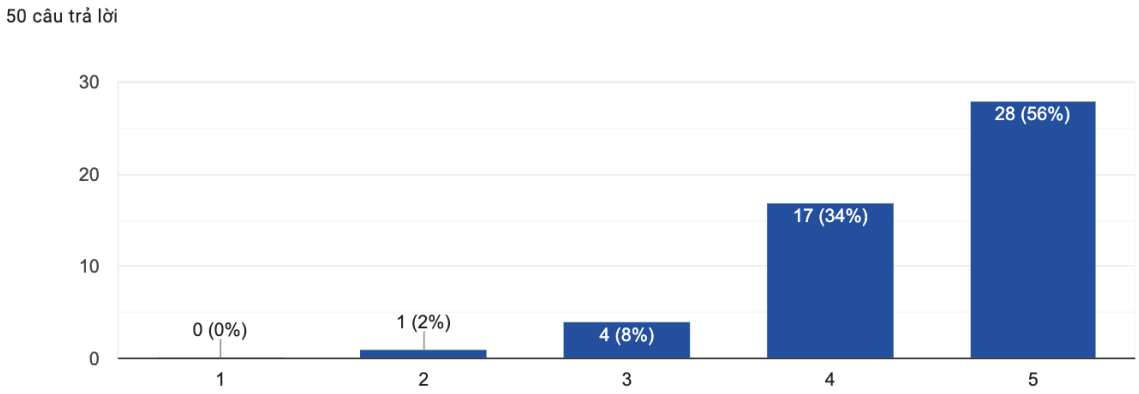

Figure 4.8: Overall, how satisfied are you with your experience using Vibelens?

4.3.4.2 Willingness to Recommend or Reuse

To gauge future engagement, users answered: **"Would you recommend Vibelens to a friend or use it again?"** The responses revealed:

- 42% answered Yes,
- 50% answered Maybe, and
- Only 8% responded No.

While direct enthusiasm (Yes) is moderate, the high proportion of "Maybe" responses indicates openness and potential for re-engagement - especially if certain features are improved or tailored further. The low outright rejection rate (8%) affirms that Vibelens leaves a generally positive impression among first-time users as shown in Figure 4.9.

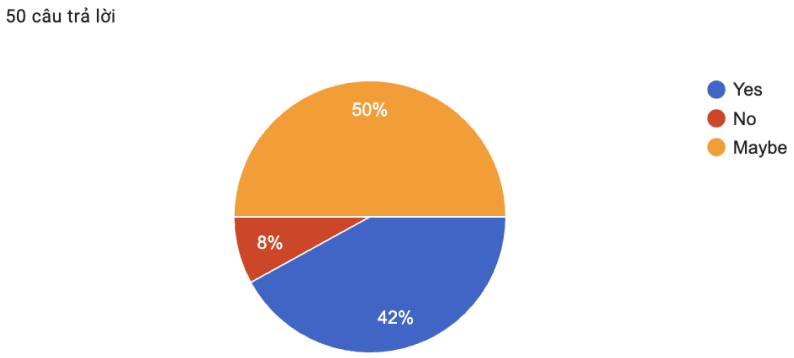

Figure 4.9: Would you recommend Vibelens to a friend or use it again?

Chapter 5

CONCLUSION AND RECOMMENDATION

5.1 Conclusion

This project set out to address the challenge of creating a personalized and emotionally resonant music recommendation system through image input. The proposed solution, **Vibelens**, combines computer vision, natural language processing, and semantic search to bridge visual experiences with musical expression.

Throughout development and deployment, we achieved our key objectives:

- We successfully built a working web prototype that allows users to upload images and receive contextually relevant music recommendations in real-time.
- Usability, responsiveness, and design were central to our implementation, as reflected in our user feedback:
  - **66% of users rated the image upload process as very easy**, and
  - **86% of users rated the interface responsiveness as 4 or 5 out of 5**, affirming the success of our frontend design.
- In terms of user experience and visual appeal, the platform received an **average rating of 4.68/5**, indicating that users found the interface both intuitive and aesthetically satisfying.
- Users were also generally satisfied with the system's performance, with **90% rating their overall experience 4 or higher**, and **42% expressing that they would recommend the tool to others**.

By reflecting on these outcomes, it is evident that Vibelens has made a meaningful contribution to the exploration of **emotion-aware, image-based recommendation systems**. The project not only demonstrates the feasibility of cross-modal AI applications in entertainment but also sets the foundation for broader use in marketing, therapy, or immersive art experiences. The positive reception from users suggests strong potential for real-world utility and continued development.

5.2 Future Recommendation

Building upon the promising outcomes of Vibelens, we propose several directions for future development and refinement to enhance the system's capabilities and broaden its applicability:

- **Expand Music Database Coverage:** Currently, the recommendation engine operates on a limited selection of music tracks. Increasing the diversity and volume of the database - including multilingual, instrumental, or genre-specific tracks - could provide richer and more tailored suggestions for users with diverse tastes and cultural backgrounds.
- **Improve Semantic Matching Algorithms:** While the current NLP and vision-language embedding pipeline performed well, there is room for refinement. Future iterations could integrate advanced models (e.g., CLIP-based multimodal transformers or emotion-aware captioning models) to strengthen the emotional and contextual relevance of music-image pairings.
- **Incorporate Real-Time Feedback Loops:** Introducing a feedback mechanism where users can upvote or downvote song suggestions will allow for adaptive learning over time, improving personalization and algorithmic accuracy.
- **Optimize System Responsiveness and Mobile Performance:** Despite favorable ratings in responsiveness (average 4.3/5), survey data indicated a small segment of users (14%) experienced minor lags or usability hiccups. Future work should explore server-side optimizations and mobile-first interface design to address these issues.
- **Add Community and Social Features:** To increase user engagement and stickiness, Vibelens could incorporate features such as playlist sharing, commenting on image-music pairs, or exploring others' public uploads, fostering a community of expressive multimedia storytelling.

- **Conduct Longitudinal Usability Studies:** While the current feedback reflects short-term impressions, long-term studies could help validate sustained usability and explore how user preferences evolve over repeated usage.

These recommendations provide a roadmap not only for technical refinement but also for expanding the platform's social and emotional value. Future researchers and developers can build upon this foundation to turn Vibelens into a more powerful and emotionally intelligent multimedia experience.

REFERENCES

[1] Markus Schedl, Hamed Zamani, Ching-Wei Chen, Yashar Deldjoo, and Mehdi Elahi. Current challenges and visions in music recommender systems research. *International Journal of Multimedia Information Retrieval*, 7(2):95–116, Apr 2018.

[2] Akash Ghosh, Arkadeep Acharya, Sriparna Saha, Vinija Jain, and Aman Chadha. Exploring the frontier of vision-language models: A survey of current methodologies and future directions, 2024.

[3] James Jie, Jianguo Wang, Guoliang Li, and James Pan. *Survey of Vector Database Management Systems*.

[4] Kleanthis Avramidis, Shanti Stewart, and Shrikanth Narayanan. On the role of visual context in enriching music representations, 2022.

[5] DataReportal. Digital around the world, Apr 2025.

[6] Fabio Duarte. Music streaming services stats (2025), Apr 2025.

[7] Adobe Express. Music's impact on social engagement | adobe, 2025.