AI-Powered Literature & Music Analyzer — Major Project Blueprint

> **Goal:** Build a production-grade web platform that analyzes poems/lyrics and music audio to extract structure, mood, and style; then connects both modalities for insights and recommendations. Designed to showcase full-stack, ML, and systems skills attractive to recruiters.

---

1) Executive Summary

- **What it does:**
- Upload text (poem, lyrics, prose) → analyze meter, rhyme, themes, tone/emotion, readability, entities, and originality/similarity.
- Upload audio (song clip) → analyze tempo/BPM, key, chord/scale hints, spectral features, energy/valence (mood), genre/tags, and instrument likelihood.
- **Cross-modal:** Recommend music for a poem (and vice-versa) by aligning text and audio embeddings; generate poetic summaries of songs; build mood-based playlists.
- **Why it stands out:** Combines **Next.js app**, **Python ML microservices**, **vector search (pgvector)**, **streaming analysis**, **background workers**, **cloud storage**, **observability**, and **clean software architecture**.
- **MVP in ~8–10 weeks; Full project in 14–16 weeks**, with stretch goals for generation.

---

2) User Personas & Top Use Cases

- **Student/Researcher:** Upload poems to study meter and rhyme; compare poets; export annotated analysis.
- **Musician/Producer:** Upload demo clip; get tempo/key/mood; find poems/lyrics matching intended feel.
- **Literature Enthusiast:** Explore themes and emotions; discover songs that match a poem's vibe.
- **Recruiter/Engineer:** Evaluate candidate's ability to ship a scalable, ML-powered product.

---

3) Core Features (with Depth)

A. Text (Poetry/Lyrics) Analyzer

1) **Language Detection:** fastText/langdetect → route to pipeline (English MVP; extend to Hindi/Telugu later). 2) **Normalization & Tokenization:** spaCy pipeline; punctuation/stopword handling. 3) **Syllable Counting & Meter Heuristics (English):** - Use CMU Pronouncing Dictionary + g2p fallback for OOV. - Count syllables per line; estimate stress pattern; heuristics for iambic/trochaic/anapestic. 4) **Rhyme Scheme Detection:** - Extract final stressed vowel + coda phonemes; - Hash rhyme endings; assign scheme labels (A,

B, C…); handle imperfect/slant rhymes via phoneme distance. 5) **Theme/Topic Modeling:** BERTopic or top-k zero-shot labels; show top terms. 6) **Emotion/Sentiment:** Fine-tuned transformer on GoEmotions (joy, sadness, anger, etc.); aggregate per stanza. 7) **Readability & Style:** Flesch-Kincaid, type-token ratio, PoS distribution, named entities. 8) **Similarity & Retrieval:** Sentence embeddings (e.g., SBERT) → **pgvector** for nearest-neighbors; find similar poems/lyrics. 9) **Outputs & Visuals:** - Rhyme graph (nodes=lines, edges=rhyme strength), meter heatmap, stanza emotion chart.

B. Music (Audio) Analyzer

1) **Preprocessing:** Resample mono 16k/22.05k; trim silence; normalization. 2) **Low-Level Features:** MFCCs, chroma, spectral centroid/rolloff, zero-crossing rate; tempo (BPM), onset strength. 3) **Key/Scale Estimation:** Chroma templates + heuristic voting; (stretch: chord progression hints). 4) **Mood/Emotion:** Pretrained audio-tagging embeddings (e.g., VGGish/YAMNet) + classifier for energy/valence. 5) **Genre/Instrument Tags:** Multi-label classifier using embeddings; thresholded tags with confidence. 6) **Audio Embeddings for Retrieval:** Create fixed-length embeddings; store in **pgvector**; nearest-neighbors. 7) **Outputs & Visuals:** Waveform, mel-spectrogram, tempo timeline, energy curve.

C. Cross-Modal (Text ↔ Audio)

1) **Shared Space Alignment:** Use pretrained **CLAP-like** audio-text model or train a lightweight projection head to map SBERT (text) and audio embeddings into a shared space. 2) **Recommendations:** - Given a poem → retrieve top-k audio clips with matching mood/themes. - Given a song → retrieve top-k poems/lyrics with matching vibe. 3) **Generative Summaries (Stretch):** - Poetic summary of a song's mood and key images (LLM-assisted, with guardrails). - Lyric prompts based on poem themes.

---

4) System Architecture

Pattern: Modular microservices with simple boundaries.

- **Web App (Next.js + Tailwind + shadcn/ui)**
- Authentication (Clerk/Auth.js), role-based permissions.
- Upload UI (text or audio); progress bar; real-time status via WebSocket/SSE.

- Dashboards with charts (Recharts) and interactive visualizations.

- **API Gateway (Node.js/Express or Next.js Route Handlers)**

- Handles auth, input validation, rate limiting, request fan-out to ML services.

- **ML Service (Python/FastAPI)**

- Endpoints: `/analyze/text`, `/analyze/audio`, `/embed/text`, `/embed/audio`.

- Uses spaCy, transformers, librosa, torchaudio; caches results in Redis.

- **Workers & Queue (Celery/RQ + Redis/RabbitMQ)**

- Long-running audio tasks executed asynchronously; status tracked via job IDs.

- **Datastores**

- **PostgreSQL** for relational data.
- **pgvector** extension for embeddings (text+audio).
- **Object Storage** (S3/MinIO) for audio files and generated plots.

- **Redis** for caching & job status.

- **Observability**

- Logging (structured JSON), metrics (Prometheus), traces (OpenTelemetry), dashboards (Grafana).

- **CI/CD & Infra**

- Docker for all services; GitHub Actions → build/test; deploy to Render/Fly.io/EC2.
- IaC (Terraform) for cloud resources (stretch).

Security/Privacy: - File type/size limits; MIME sniffing; AV scan (ClamAV) for uploads; signed URLs for S3. - PII scrubbing from text; content moderation for uploads.

---

5) Data Model (Relational + Vector)

Tables (PostgreSQL): - `users(id, name, email, role, created_at)` - `texts(id, user_id, title, language, raw_text, created_at)` - `audio(id, user_id, title, s3_url, duration_sec, samplerate, created_at)` - `analyses(id, target_type [text|audio], target_id, status, started_at, completed_at, summary_json)` - `text_metrics(text_id, syllables_json, meter_json, rhyme_json, emotions_json, topics_json, readability_json)` - `audio_metrics(audio_id, tempo_bpm, key_guess, energy_curve_json, tags_json, embedding_id)` - `embeddings(id, modality[text|audio], vector float8[], dim int, created_at)` (pgvector) - `jobs(id, kind, payload_json, status, progress, created_at, updated_at)` - `shares(id, target_type, target_id, shared_with, scope[read|comment], created_at)`

Indexes: - GIN/JSONB indexes on `*_json` fields where needed; `ivfflat` or `hnsw` index on `embeddings.vector`.

---

6) API Design (Sample)

Auth: JWT/session; rate limit per user.

Text: - `POST /api/texts` → `{ title, text }` → `text_id` - `POST /api/analyze/text/:text_id` → returns `job_id` - `GET /api/jobs/:job_id` → `{ status, progress }` - `GET /api/texts/:id/analysis` → full metrics and visuals URLs - `POST /api/search/text` → `{ query }` → similar texts via embeddings

Audio: - `POST /api/audio` (multipart) → `audio_id` - `POST /api/analyze/audio/:audio_id` → `job_id` - `GET /api/audio/:id/analysis` - `POST /api/search/audio` → `{ clip_id|audio_embed }` → similar tracks

Cross-modal: - `POST /api/recommendations/from-text` → `{ text_id }` → top-k audio - `POST /api/recommendations/from-audio` → `{ audio_id }` → top-k texts

OpenAPI auto-docs for FastAPI + typed clients.

---

7) Algorithms & Implementation Details

A. Rhyme Scheme Detection (English)

1. For each line, take last content word; get phonemes via CMUdict; if OOV → g2p fallback.
2. Extract from last stressed vowel onward.
3. Compute rhyme key (e.g., tuple of phonemes); compare using edit distance or phoneme similarity.
4. Greedy grouping to assign letters A/B/C…; allow slant rhyme via threshold.

B. Meter Estimation

1. Syllable counts per line (CMUdict syllables; fallback heuristic using vowel groups).
2. Estimate stress pattern using CMU primary/secondary stress; allow substitutions (pyrrhic/spondee) via penalties.
3. Score candidates (iambic, trochaic, anapestic, dactylic) → pick best fit.

C. Topic & Emotion

- **Embeddings:** `sentence-transformers` (e.g., all-MiniLM-L6-v2) cached.
- **Topics:** BERTopic with class-based TF-IDF; visualize term importance.
- **Emotion:** Fine-tune BERT on GoEmotions; aggregate per line/stanza; produce confidence with softmax temperature.

D. Audio Analysis

- **Features:** librosa for mel-spectrograms, chroma, tempogram; onset detection for BPM.
- **Embeddings:** Pretrained audio models (e.g., VGGish/YAMNet) → 1024/embedding; mean-pool over time windows.
- **Classifiers:** Shallow MLP/LogReg on embeddings for mood/tags; calibrate with Platt scaling.
- **Key/Scale:** Template correlation across chroma vectors; majority vote over frames.

E. Cross-Modal Alignment

- Option 1 (MVP): Normalize both embeddings; cosine similarity for retrieval.
- Option 2 (Stretch): Learn 2-layer projection heads on small paired (lyrics+audio) dataset; train contrastively.

---

8) Datasets & Ethics

- **Text:** Public-domain poetry (Project Gutenberg), lyrics **without** storing copyrighted content (only derived embeddings/metrics). Add terms prohibiting full copyrighted text uploads unless user owns rights.
- **Audio:** Use permissive datasets (FMA-small, GTZAN with caveats, MagnaTagATune). For user uploads, limit to short clips (<30s) for analysis.
- **Ethics/Compliance:** Respect copyright, provide takedown, avoid biased labels; allow “do not store” option.

---

9) Frontend UX

- Clean upload flows; analysis progress (SSE/WebSocket).
- Visuals: spectrogram, tempogram, rhyme graph, meter heatmap, emotion radar; export as PNG/PDF.
- Explore: search by mood (happy, melancholic), tempo range, key, themes.
- Shareable analysis pages with privacy controls.

---

10) Engineering Plan & Timeline (16 Weeks)

Week 1–2: - Repo setup (monorepo with `apps/web`, `services/ml`, `infra`); Docker; Postgres+pgvector; MinIO; Redis. - Scaffolding: Next.js app, FastAPI service, CI (lint/test/build), basic auth.

Week 3–4 (MVP Text): - Implement text pipeline (lang detect, syllables, rhyme, emotion, topics); store metrics; simple charts.

Week 5–6 (MVP Audio): - Audio upload, feature extraction, BPM/key; embeddings + simple mood classifier; store metrics.

Week 7–8 (Cross-Modal v1): - Embedding retrieval across modalities; recommendations from text→audio and audio→text.

Week 9–10 (UX & Sharing): - Polished dashboards; shareable results; vector search UI; pagination & filters.

Week 11–12 (Scale & Observability): - Caching, background jobs, S3 signed URLs; Prometheus+Grafana; structured logging; error budgets.

Week 13–14 (Polish & Docs): - OpenAPI docs; tutorial notebooks; test coverage; accessibility; security review.

Week 15–16 (Stretch): - Generative summaries; chord hints; multilingual text pipeline (Hindi/Telugu syllables & rhyme via Indic NLP).

---

11) Tech Stack (Opinionated Picks)

- **Frontend:** Next.js 14, React 18, Tailwind, shadcn/ui, Recharts, React Query, Auth.js/Clerk.
- **Backend/API:** Next.js route handlers or Node/Express gateway; FastAPI (Python) for ML.
- **ML/NLP/Audio:** spaCy, transformers, sentence-transformers, BERTopic, librosa, torchaudio, VGGish/YAMNet.
- **Data:** PostgreSQL + pgvector, Redis, MinIO/S3.
- **Infra:** Docker, GitHub Actions, Render/Fly.io (simple), AWS (scalable), Terraform (stretch).
- **Testing:** pytest, unittest, Jest/Playwright; synthetic fixtures.

---

12) Evaluation & Metrics

- **Text:**
- Rhyme detection F1 on curated set; meter accuracy vs. annotated examples.
- Emotion: macro-F1 on validation split.
- **Audio:**
- BPM MAE vs. hand-labeled; key accuracy top-1/top-2; tag mAP.
- **Cross-modal:** Recall@k for retrieval; MRR.
- **System:** P95 latency for short texts (<1s), small audio (<10s) < 3s (cached); uptime SLO 99%.

---

13) Risks & Mitigations

- **Copyright issues** → clip length limits; user attestations; storage opt-out.
- **Model bias/accuracy** → calibration, human-in-loop overrides.
- **Complexity creep** → MVP first; freeze scope after week 6; stretch later.
- **Performance** → background jobs, caching, vector indexes.

---

14) Deliverables (for Academic & Hiring)

- Live demo (URL), screencast, and test accounts.
- Public repo with README, architecture diagram, benchmarks, Swagger docs.
- Short whitepaper/report explaining algorithms with examples.
- Case studies: 5 poems + 5 audio clips with detailed annotations.

---

15) Sample Implementation Snippets (Pseudo/Realistic)

FastAPI – Text Analyze

```
@app.post("/analyze/text/{text_id}")
async def analyze_text(text_id: str):
    text = db.get_text(text_id)
    lang = detect_lang(text)
    syllables = count_syllables(text)
    meter = estimate_meter(text)
    rhymes = detect_rhyme_scheme(text)
    emo = emotion_model.predict(text)
    topics = topic_model.infer(text)
    emb = sbert.encode([text])
    emb_id = store_embedding(emb, modality="text")
    db.save_text_metrics(text_id, syllables, meter, rhymes, emo, topics, emb_id)
    return {"ok": True}
```

Audio – BPM & Embedding

```
sig, sr = librosa.load(path, sr=22050, mono=True)
tempo, beats = librosa.beat.beat_track(y=sig, sr=sr)
mel = librosa.feature.melspectrogram(y=sig, sr=sr)
emb = audio_model.embed(mel)  # e.g., YAMNet/VGGish
store_embedding(emb, modality="audio")
```

Cross-Modal Retrieval (Cosine)

```
-- pgvector similarity
SELECT id, 1 - (embeddings.vector <=> $1) AS score
FROM embeddings
WHERE modality = 'audio'
ORDER BY embeddings.vector <=> $1
LIMIT 10;
```

---

16) Repo Structure (Monorepo)

```
root/
  apps/
    web/ (Next.js)
```

```
services/
  ml/ (FastAPI, models, notebooks)
packages/
  shared/ (types, client SDK)
infra/
  docker-compose.yml, terraform/
docs/
  architecture.md, api.md
```

17) Stretch Ideas

- **Chord progression & key change detection** using Viterbi over chord states.
- **Multilingual poetry analysis** (Indic scripts): syllabifier via Indic NLP; rhyme using transliteration + phoneme heuristics.
- **Poetry generation conditioned on audio mood** (careful scoping).
- **Mobile capture**: record 15s audio, instant analysis.

18) What to Build First (Concrete Checklist)

1. Dockerized Postgres + pgvector + MinIO + Redis.
2. Next.js upload UI for text + audio; job progress via SSE.
3. FastAPI `/analyze/text` with rhyme/meter/emotion (MVP).
4. FastAPI `/analyze/audio` with BPM/key + embeddings (MVP).
5. Vector search endpoints + cross-modal top-k.
6. Visualizations + shareable result pages.

19) Why This Wins in Interviews

- Shows **end-to-end ownership**: data, models, APIs, UI, infra.
- Demonstrates **in-demand tech**: embeddings, vector DB, async jobs, streaming, cloud.
- Delivers a **unique, polished product** that is easy to demo and discuss.

Verdict: This is absolutely “major-project scale.” Start with the MVP path above; lock scope to ship in ~10 weeks, then add cross-modal alignment and generation as stretch goals.