♪ MUSIC AGENT

Music Analysis Through Natural Language

A music analysis agent that lets users explore and manage their music library, powered by LLMs

HERWEDE NOAH, AHMED-YAHIA LAÏFA - M2 DS2E 2026

Project overview

Goal

Enable users to explore, analyze, and manage a music library using natural language queries.

Technical approach

A Large Language Model (LLM) connected to Python tools for audio analysis and music data management.

Core principle

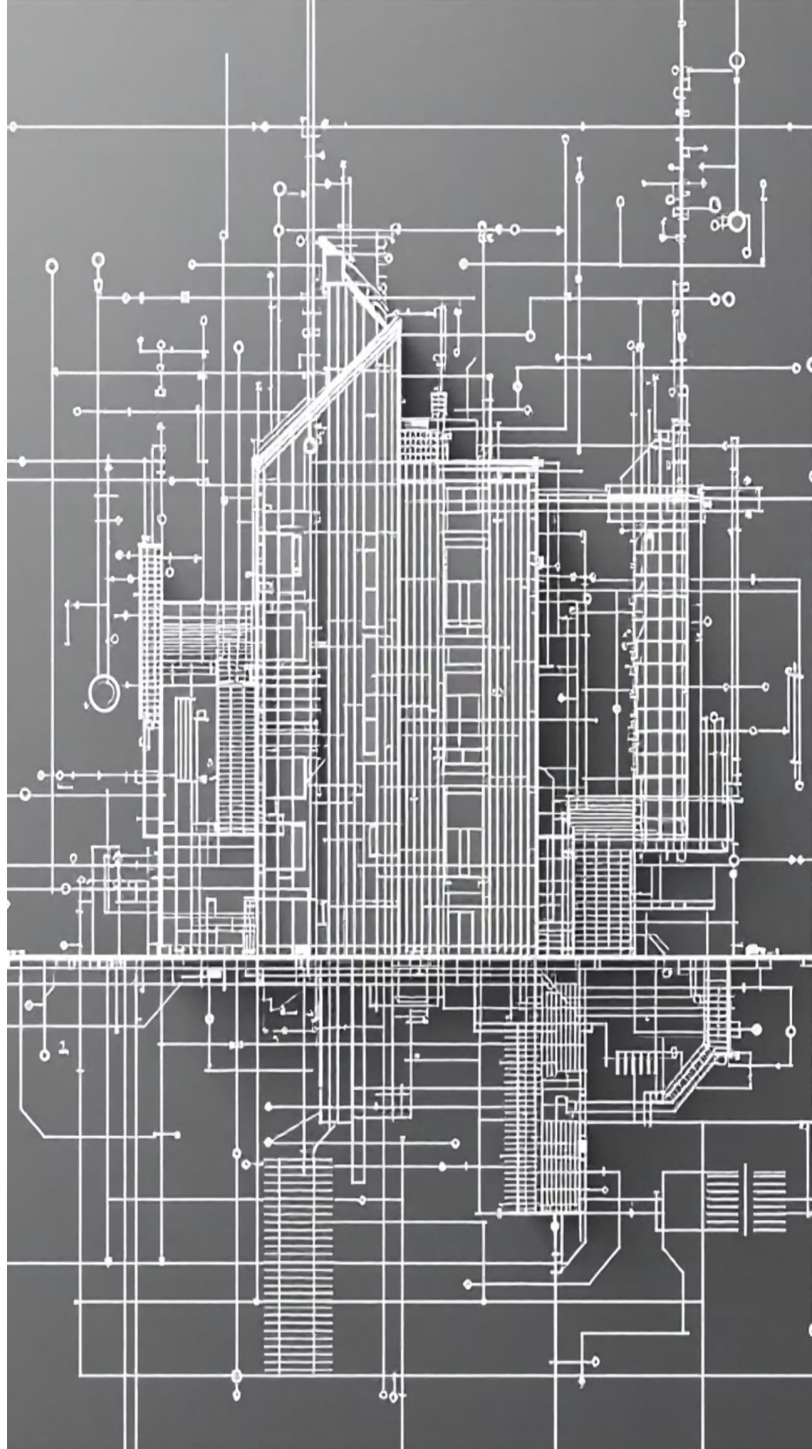
LLM plans actions

Python tools execute computations.

Reliability

All results come from real data processing

No hallucinated outputs.



Project Scale & Complexity

5K

Lines of Code

Python codebase powering the entire system

15+

LLM Tools

Distinct tool definitions exposed to language models

4

LLM Providers

Ollama, Groq, OpenAI, Anthropic supported

5

Database Tables

Core data structures for tracks and playlists

Core Capabilities

Library Analysis

Global music library analysis with similarity search based on audio embeddings and visual HTML report generation with plots.

Audio Processing

Audio feature extraction, format conversion, and direct saving of uploaded audio into the local database.

External Integration

Deezer trends, YouTube download, and Shazam identification integrated seamlessly into the workflow.

Playlist Management

Automatic playlist generation with unified reporting tools for track-level or library-level insights.

ARCHITECTURE

Three-Layer System Design

User Interfaces

Flask-based web interface and command-line interface for natural language queries, file uploads, LLM switching, and clickable links to reports.

Agent Core

Decision-making layer that receives queries, interacts with LLMs, and determines tool execution. Enforces tools-first workflow for data tasks.

Python Tools

Execution layer performing all concrete computations: audio analysis, similarity search, report generation, conversion, visualization, and external API access.

> **Design Philosophy:** Strict separation of responsibilities prevents hallucinations, all results come from real computations, not LLM-generated text.

Data Flow Architecture

1

User Input

Natural language query with optional audio file

2

Agent Processing

Forwards to LLM with system prompt and tool descriptions

3

LLM Response

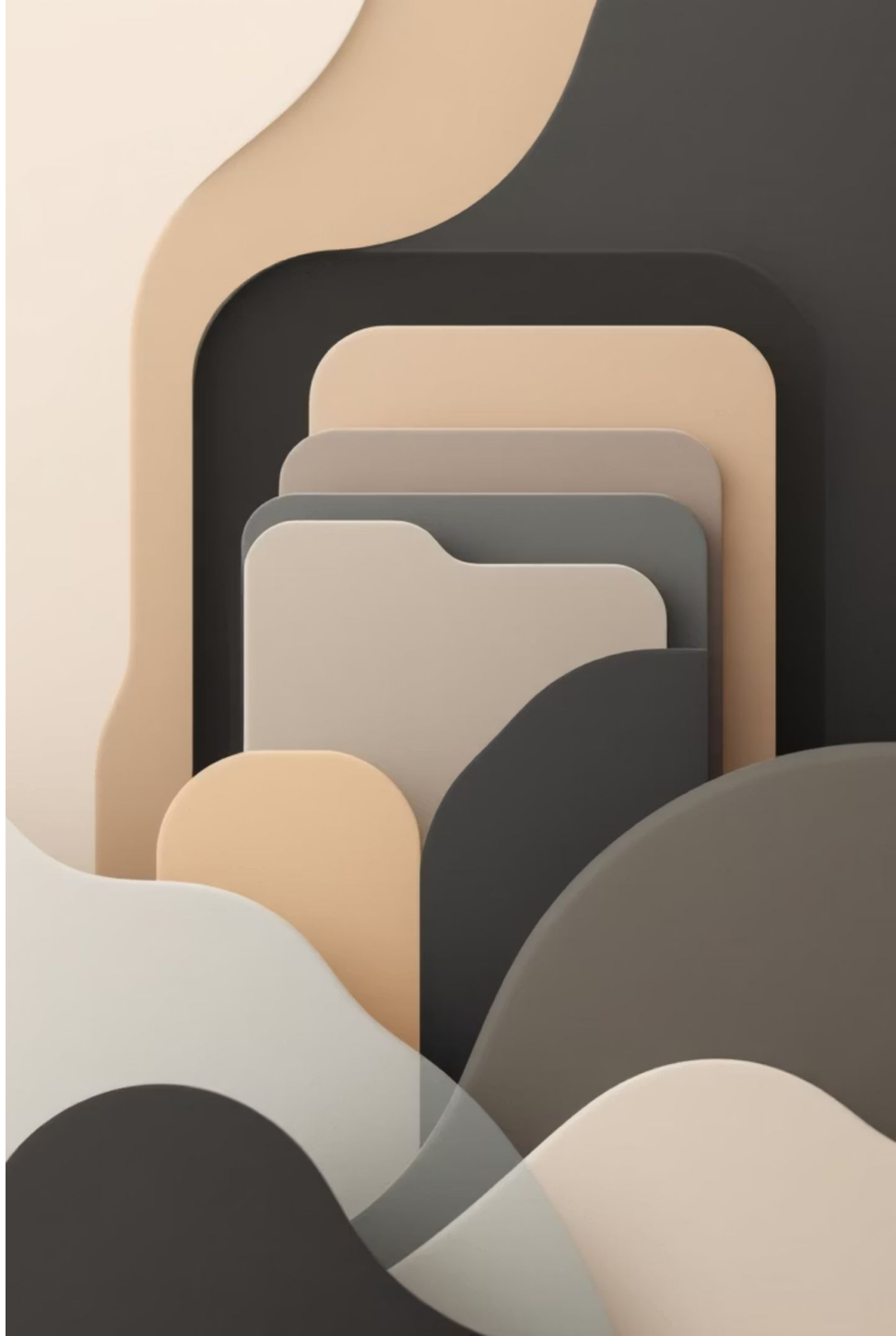
Returns structured JSON with tool calls and parameters

4

Tool Execution

Agent dispatches tools, formats results, returns to LLM

The LLM never introduces new data or computed values. It only orchestrates tool selection and adds optional comments with link highlights.



Local Data Storage with SQLite

Database Tables

tracks

Title, artist, file path, genres

embeddings

Audio feature vectors, tempo, energy, descriptors

playlists

Generated playlists with track ordering

external_tracks

Cached results from Deezer, Shazam

File System Organization

- Raw audio data and uploaded files
- Converted audio files
- Generated reports and plots
- Temporary processing files

Clear separation simplifies maintenance, debugging, and enables direct linking in the web interface.

Comprehensive Tool Catalog

Analysis & Search

Library analysis and similarity search based on audio embeddings

Reporting & Visualization

Report generation and data visualization with plots

Audio Utilities

Format conversion and genre detection

External APIs

Deezer trends, YouTube download, Shazam identification

File Management

Upload handling and database saving

Tool Chaining for Complex Queries

Single queries can trigger multiple steps: library analysis → report generation, or conversion → database saving. Essential for handling sophisticated natural language requests.

Focus on Audio Converter

Purpose

Standardize audio files before processing

Core Operations

- Audio format conversion
- Container and stream normalization
- Deterministic audio preprocessing

Integration in the system

- Implemented as an autonomous Python tool
- Selected via LLM tool calling
- Output persisted to local storage

```python
def _normalize_format(fmt: str) -> str:
    fmt = fmt.strip().lower()
    if fmt.startswith("."):
        fmt = fmt[1:]
    if not fmt or not fmt.isalnum():
        raise ValueError("Invalid output format")
    return fmt


def _unique_path(path: Path) -> Path:
    if not path.exists():
        return path
    stem = path.stem
    suffix = path.suffix
    parent = path.parent
    idx = 1
    while True:
        candidate = parent / f"{stem}_{idx}{suffix}"
        if not candidate.exists():
            return candidate
        idx += 1


def convert_audio(
    filepath: str,
    output_format: str,
    output_dir: Optional[str] = None,
    overwrite: bool = False,
) -> Dict[str, Any]:
    """Convert a local audio file using ffmpeg."""
    input_path = Path(filepath)
    if not input_path.exists():
        raise FileNotFoundError(f"File not found: {filepath}")

    fmt = _normalize_format(output_format)
    out_dir = Path(output_dir) if output_dir else input_path.parent
    out_dir.mkdir(parents=True, exist_ok=True)

    output_path = out_dir / f"{input_path.stem}.{fmt}"
    if not overwrite:
        output_path = _unique_path(output_path)

    cmd = [
        "ffmpeg",
        "-loglevel",
        "error",
        "-y" if overwrite else "-n",
        "-i",
        str(input_path),
        "-vn",
        str(output_path),
    ]

    try:
        subprocess.run(cmd, check=True, capture_output=True, text=True)
    except FileNotFoundError as exc:
        raise RuntimeError("ffmpeg is not installed or not on PATH") from exc
    except subprocess.CalledProcessError as exc:
        detail = exc.stderr.strip() if exc.stderr else str(exc)
        raise RuntimeError(f"ffmpeg failed: {detail}") from exc

    return {
        "input_path": str(input_path),
        "output_path": str(output_path),
        "format": fmt,
        "overwrote": overwrite,
    }
```

LLM Integration & Function Calling

1 **Multi-Provider Support**

Automatic provider selection with dynamic switching via web interface

2 **System Prompt**

Defines model's role precisely

3 **JSON Schema**

Describes all available tools with parameters

4 **Structured Response**

LLM returns tool_calls JSON, not text

5 **Tool Execution**

Agent interprets and executes Python functions

Fallback Mechanisms

1. Enforce strict JSON response if structure invalid
2. If the LLM return errors or is indisponible the agent switch to a keyword-based heuristics as last resort

These mechanisms ensure robustness even when LLM responses deviate from expected format.

Modular Code Structure

Central Orchestration

`agent.py` contains all orchestration logic and tool dispatching mechanisms.

Tools Package

Separate modules for specialized functionality:

- Audio analysis
- Database access
- Report generation
- Visualization
- Format conversion
- External API integration

> *"This modular organization makes the codebase easy to navigate and helps quickly identify the responsibility of each component."*

Benefit: Each component can be tested, maintained, and extended independently without affecting other parts of the system.

Live Demo

Interactive demonstration of Music Agent capabilities

Key Design Decisions

1

Modular Architecture

Strict separation of agent from tools enables independent evolution of each component without system-wide refactoring.

2

LLM Role Limitation

Restricting the language model to planning, and descriptive roles ensures explainability, robustness, and eliminates hallucination risks.

3

Local-First Storage

Using SQLite and autonomous tools simplifies testing, maintenance, and enables future extensions with full data control.

> **Core Principle:** All statistics, recommendations, and results are derived from real computations on actual data, never from LLM-generated text.

Limitations & Future Directions

Current Limitations

- External tracks don't always provide usable audio data,
- Some code sections could be further decomposed
- Optimise LLM Tools calling

Future Improvements

01

Spotify Integration

Add additional API support

02

Test Coverage

Increase automated testing

03

Performance

Optimize for large libraries

Questions & Discussion

Thank you for your attention. We're ready to answer your questions about Music Agent.