

Song Search using CLAP

SUSE Hack Week 25



Goals

Search music using natural language descriptions

- Find songs by describing what you hear: "piano melody", "female vocalist"
- Compare songs to find similar tracks
- Experiment AI with Copilot Code Assistant

CLAP Overview

CLAP (Contrastive Language-Audio Pretraining) - Bridging audio and language

Core Concept:

CLAP can represent both songs and text descriptions as **512-dimensional vectors** in the same mathematical space. This enables searching for music using natural language queries.

How it works:

- A song becomes a 512-number vector capturing its audio characteristics
- A text query like "piano music" also becomes a 512-number vector
- Vectors that point in similar directions represent similar content
- We measure similarity to find songs matching the description

Model: music_audioset_epoch_15_esc_90.14.pt - Specialized for music (71% genre accuracy, trained on ~4M samples)



Implementation

Two Modes:

- **Single Analysis:** `single_analysis.py` - One song vs one query
- **Batch Analysis:** `multiple_analysis.py` - All songs vs all queries

Architecture:

- `clap_analysis.py` - Core library (shared)
- Automatic model download & caching
- Parallel processing across CPU cores
- CSV output with detailed metrics

Cosine Similarity Scores:

- Range: -1 (opposite) to $+1$ (identical)
- Thresholds: $> 0.3 = \text{HIGH}$, $> 0.15 = \text{MODERATE}$, $\leq 0.15 = \text{LOW}$

Output Example - Classical

Song: Aaron Dunn - Minuet (Classical Piano)

HIGH (> 0.3)	MODERATE (> 0.15)	LOW (\leq 0.15)
0.41 piano	0.29 chill	0.15 70s
0.40 classical instrumental song with piano	0.28 chillout	0.14 choir
0.38 classical music	0.27 sad	0.14 beautiful
0.37 instrumental	0.26 jazz	0.14 danceable
0.33 instrumental music	0.25 acoustic	0.12 pop music wit bass guitar
	0.23 acoustic guitar	0.12 funk
	0.22 pop	0.12 bass guitar
	0.21 folk	0.11 soul
	0.21 happy	0.11 rock

Observation: Piano and classical-related queries score highest. Chill/mellow descriptors moderate. Rock and modern era queries score low.

Tip: Detailed, specific queries combining multiple descriptors (genre + instrument + style) yield excellent results!

Output Example - Rock

Song: SUSE Band - Paint it Green (Rock)

HIGH (> 0.3)	MODERATE (> 0.15)	LOW (≤ 0.15)
0.42 indie rock	0.28 hard rock	0.14 heavy metal
0.41 alternative rock	0.28 folk	0.14 dance
0.40 classic rock	0.26 punk	0.13 soul
0.37 Progressive rock	0.25 80s	0.13 instrumental
0.37 indie pop	0.25 funk	0.13 acoustic guitar
0.36 rock	0.2590s	0.13 singing
0.33 female vocalist	0.2460s	0.12 bass guitar
0.32 rock music	0.24 metal	0.12 guitar
0.31 blues	0.23 Mellow	0.06 female voice

Observation: Rock subgenres (indie, alternative, classic) score highest. Generic instrument queries score low.

Query Wording Matters: Notice "female vocalist" (0.33, HIGH) vs "female voice" (0.06, LOW) - 5x difference! Use music industry vocabulary for better results.

Finding Similar Songs

Bonus Feature: Beyond text search, we can find which songs sound similar to each other.

Example: Finding songs similar to SUSE Band - Paint it Green (Rock)

Song	Similarity
HoliznaCC0 - Dreams Of Lilith (Rock)	0.72
Jenna Jay - Someone Real (Pop)	0.56
Zane Little - Always and Forever (Pop)	0.47
SUSE Band - SUSE Yes Please (Pop)	0.43
SUSE Band - Can't Stop The SUSE (Pop)	0.39
Aaron Dunn - Minuet (Classical Piano)	0.18

Use Cases:

- Find songs with similar instrumentation or mood
- Create automatic playlists based on audio similarity
- Discover patterns in your music collection

AI Support in Development - What Went Well

Project developed with GitHub Copilot (Claude Sonnet 4.5)

1. Research & Model Selection

- Compared CLAP models → selected music_audioset (71% GTZAN accuracy)
- Read CLAP paper, explained architecture accessibly

2. Code Analysis & Technical Discovery

- Examined checkpoint: model lacks fusion, max 10s segments
- Designed overlapping segment solution

3. Data Analysis & Documentation

- Found "vocalist" vs "voice" pattern (5x difference in results)
- Generated all technical docs with examples

1. Unrequested Code Changes

- Single-file CLI disappeared, score icons changed without asking
- **Problem:** Full file rewrites hide unintended changes
- **Lesson:** Always review diffs carefully

2. First Solution \neq Best Solution

- **Performance:** Song analyzed per query \rightarrow challenged \rightarrow **15x faster**
- **Threading:** Single-threaded \rightarrow asked "faster?" \rightarrow multiprocessing

Key Lesson: Working \neq optimal. Challenge AI with domain knowledge.

Best Practices for AI-Assisted Development

Do's:

- Ask AI to research and compare options (models, approaches)
- Request code analysis to understand existing implementations
- Share actual data/errors for pattern recognition
- Challenge solutions: "Can this be faster?" "Is there a better way?"
- Request documentation with examples from your project

Don'ts:

- Don't accept first working solution as optimal
- Don't let AI rewrite entire files without showing diffs
- Don't assume AI made only requested changes - review carefully
- Don't trust performance without testing alternatives

Bottom line: AI is a powerful assistant for research, analysis, and documentation. For code: it provides a working starting point, but optimization requires your expertise and critical thinking.



References

- **CLAP:** <https://github.com/LAION-AI/CLAP> - The main model being researched
- **Hugging Face:**
https://huggingface.co/docs/transformers/model_doc/clap
 - Pre-trained models for CLAP
- **Free Music Archive:** <https://freemusicarchive.org/> - Creative Commons songs for testing
- **SUSE Hack Week 25 Project:**
<https://hackweek.opensuse.org/25/projects/clap-machine-learning-to-search-song-starting-from-text>
 - Project page
- **Project Repository:**
<https://github.com/gcolangiulisuse/hw25-song-search> - GitHub repo of the project