Audio Fingerprinting

Jayant Manchanda - report

April 11, 2025

1 Introduction

Audio fingerprinting is the process of extracting a signature from an audio signal so that it can be efficiently matched against a large database of known signatures. This work presents an attempt at the Shazam-inspired algorithm for audio fingerprinting, focusing on robust frequency peak extraction and combinatorial hashing.

The fundamental principle behind Shazam-style fingerprinting is that the "peak constellation map" derived from the time-frequency representation of a track remains stable under various forms of audio degradation, including additive noise, compression artifacts, and slight speed fluctuations. Once these peaks are detected, a combinatorial hashing strategy transforms local peak pairs into concise integer tokens (hashes). Efficient indexing of these tokens allows real-time or near-real-time identification from an audio snippet [1].

2 Literature Review

The early audio fingerprinting methods established the basic ideas that are still evident in most variants. Chief among these is the Shazam approach [1] and most of the research followingward has been variations around this approach. An alternative, the Philips technique, applies different spectral feature extraction but shares Shazam's goal of optimizing real-time performance, accuracy, and resilience against acoustic distortions. Google's Waveprint system further refines these ideas, prioritizing scale and handling for enormous music catalogs.

Deep learning-based approaches [2] are being used for this fingerprinting purposes. Graph Neural Networks (GNNs) have also been explored [3]. However, classical combinatorial hashing remains competitive for real-time mobile deployment due to its simplicity and low computational cost.

In this project, I follow the core methodology of Shazam while making minor modifications to the hashing scheme and index-building process.

3 Method

The overall system comprises of two major components: (1) constructing a fingerprint database from reference tracks and (2) querying against this database with a short and noisy audio snippet.

3.1 Spectrogram Analysis and Peak Extraction

A standard short-time Fourier transform (STFT) method for time-frequency representation is used. The audio signal is windowed and transformed to the frequency domain, and only the magnitude is retained. This magnitude spectrum is converted to decibel scale (dB):

$$S_{\mathrm{dB}}(k, n) = 20 \log_{10} \big| X(k, n) \big|.$$

Although the original Shazam technique employs a density-based approach to peak detection, here fixed dB threshold is used to remove softer peaks. Peaks are identified by locating local maxima within a specified neighborhood and retaining only those above the threshold.

3.2 Constellation Map

Each retained peak is treated as a (time, frequency) coordinate in the spectrogram, forming a *constellation map*. These coordinates are then sorted and passed on to the hash generation function.

3.3 Hash Generation

Following the Shazam strategy, pairs are formed from each peak (the "anchor") with subsequent peaks within a specified time window. Each pair encodes:

- Anchor frequency
- Target frequency (the frequency of a subsequent peak)
- Time difference between the anchor and target peak

These pairs are constrained to a limited frequency range (30 Hz to 4000 Hz) and a maximum time window of 2.0 s. Each token is then quantized and packed into a 32-bit integer by allocating bits to the anchor frequency, target frequency, and time difference. As opposed to the paper's approach of using a 64 bit packed integer, a 32 bit packed integer is used for simplicity. These tokens form the basis of rapid audio matching in the next stage

3.4 Parameters

3.5 Parameter Choices

- **Sample Rate:** 22,050 Hz
- **STFT Window Size:** 4096 samples
- **Hop Size:** 512 samples
- **Peak Detection:** 20×20 neighborhood, −40 dB threshold
- **Time Delta Range:** 0.0 s to 2.0 s, quantized at 0.01 s increments

- **Frequency Range:** 30 Hz to 4000 Hz
- **Hash Token:** 32-bit integer (10 bits for frequency, 10 bits for time difference, etc.)

I chose a sample rate of 22,050 Hz to balance audio fidelity with computational efficiency; this differs from the lower rates (e.g., 8,000 Hz) sometimes used in Shazam's original mobile-focused design. The STFT uses a 4,096-sample window with a 512-sample hop, providing moderate frequency resolution while maintaining relatively fine time resolution. Peak detection relies on a 20×20 neighborhood search and a -40 dB amplitude threshold, ensuring only the most prominent spectral peaks are retained.

For hashing, I limit the time delta between anchor and target peaks to a 0.0 s–2.0 s window, further quantized in 0.01 s steps to capture small-scale timing relationships. I also constrain the analysis to the 30 Hz–4,000 Hz band, ignoring lower and higher frequencies that typically contribute less to music recognition. Each anchor–target pair is ultimately packed into a 32-bit integer: 10 bits each for frequency and time difference. While Shazam originally uses more bits to encode additional information (like track ID and offsets), this simpler 32-bit scheme keeps the hashing compact yet sufficiently robust.

3.6 System Design and High-Level Implementation

1. **Database Fingerprint Builder:** Generate peaks and combinatorial hash tokens for each reference track. Store these hashes in an inverted index (a Python dictionary mapping hash values to lists of (track ID, time offset)). Serialize the dictionary with `pickle`.
2. **Query Processing:** Extract peaks and hash tokens from the query snippet. For each query hash, look up matching entries in the inverted index.
3. **Scoring and Ranking:** For each matching entry, align the query tokens with the database tokens to accumulate scores. Rank the candidate tracks by score, returning the top matches.

PIPELINE:

1. Audio Input (Database or Query)
2. STFT $\rightarrow$ Magnitude $\rightarrow$ dB Conversion
3. Local Peak Detection $\rightarrow$ Constellation Map
4. Hash Generation $\rightarrow$ Combinatorial Hashes
5. (Database) Inverted Index Construction
6. (Query) Inverted Index Lookup
7. Scoring and Ranking $\rightarrow$ Top Matches

4 Evaluation

4.1 Overall Performance

The system was evaluated by checking whether the correct track appeared in the top-k predictions for each test query. Table 1 summarizes the results.

Genre	Top-1 Acc.	Top-3 Acc.
Pop	89%	96%
Classical	75%	78%
Overall	**82%**	**87%**

Table 1: Accuracy comparison by genre (Top-1 and Top-3).

Classical music proved more challenging due to its less defined percussive content, while pop tracks achieved higher accuracy.

4.2 Parameter Comparisons

Table 2 presents performance under different settings. The best result is obtained with a 20×20 neighborhood, −40 dB threshold, 4096 FFT, and 512-sample hop.

Nbh (n×n)	dB Thresh.	FFT Size	Hop	Top-1	Top-3
15×15	−30	2048	512	80%	85%
20×20	−40	2048	512	81%	86%
20×20	−40	2048	256	80%	85%
20×20	−40	4096	512	81%	86%

Table 2: Performance under different parameter configurations.

4.3 False Negatives and Overlapping Audio

In some cases, false negatives occurred because two audio tracks shared nearly identical content. Examples include:

- `pop.00022.wav` & `pop.00015.wav`
- `pop.00054.wav` & `pop.00060.wav`
- `pop.00045.wav` & `pop.00046.wav`

Treating these duplicates as "correct" would slightly improve the system's reported accuracy.

4.4 Peak Selection Approaches

Switching from `peak_local_max` (in `scikit-image`) to a custom `max_filter` approach significantly improved results:

Metric	Accuracy
Top-1 (Overall)	64%
Top-1 (Pop)	88%
Top-1 (Classical)	41%
Top-3 (Overall)	71%
Top-3 (Pop)	97%
Top-3 (Classical)	45%

Table 3: Accuracy with `peak_local_max` approach.

5 Weaknesses and Known Bugs

- **Pickle Database:** The entire database index must be loaded into memory. A more efficient, scalable solution (e.g., Elasticsearch or Redis) would suit real-world deployments.
- **Heuristic Parameters:** dB threshold, neighborhood size, and quantization intervals were chosen based on empirical trials. Adaptive or data-driven optimization could improve performance.
- **Frequency Range:** Restricting the range to 30–4000 Hz may exclude higher-frequency cues. Adjusting this range may improve accuracy for certain genres.

6 Future Improvements

General parameter tuning across multi genres could be the first direction towards improvement. Also the code could be refactored and made computationally more efficient. This could invovle using parallel programming techniques to make the process quicker and computationally efficient. The peak selection function could be further improved, or perhaps be written for the usage of audio fingerprinting.

References

[1] A. Wang. An industrial strength audio search algorithm. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR)*, 2003.

[2] C. Nikou. Survey on deep learning methods for audio fingerprinting. *Journal of Audio, Speech, and Music Processing*, 12(1):45–59, 2023.

[3] A. Bhattacharjee. GraFPrint: A GNN-Based Approach for Audio Identification. *Conference on Advanced Audio Processing*, 2025.

*Declaration of general use of LLM's for language and understanding