# Music Similarity Based on Multivariate Correlation

Master Project Report
April 2026

Ronald Domi, 23-744-600
Giuseppe Doda, 23-717-044

Supervisors:
Prof. Dr. Renato Pajarola
Xiao Tan, MSc
Research Assistant

Visualization and MultiMedia Lab
Department of Informatics
University of Zürich

University of Zurich[UZH]

VISUALIZATIONANDMULTIMEDIALAB

# Abstract

This project investigates whether classical statistical methods can capture perceptually meaningful similarity between music tracks. Neural embedding models dominate music information retrieval but require significant computational resources and were not designed for perceptual similarity tasks. We explore two lightweight alternatives: extracting spectral features, reducing dimensionality with PCA, and comparing songs using either Earth Mover's Distance (EMD) or Mutual Information (MI). Our best EMD configuration achieves 60.79% accuracy on the MagnaTagATune odd-one-out task, and our best MI configuration (KSG estimator) achieves 57.41%. While MERT-330M achieves slightly higher accuracy (60.98%), our methods are orders of magnitude faster (3.2s vs 89.9 min).

# Contents

# 1. Introduction

Music similarity is a fundamental task in music information retrieval (MIR) with applications in recommendation systems, playlist generation, and music discovery.

## 1.1. Current Landscape

The state of the art in music similarity is dominated by deep learning. Models like MERT [LYZ+23] and CLMR [SB21] learn embeddings from large-scale audio data and achieve strong performance on MIR benchmarks. However, deploying these models in practice presents challenges:

- **Computational cost**: Neural models require GPU acceleration and significant memory. MERT-330M cannot run efficiently on consumer hardware.
- **Latency**: Generating embeddings for a single song takes seconds, making real-time applications difficult.
- **Interpretability**: Embeddings are opaque—it is unclear which musical properties drive similarity judgments.
- **Deployment complexity**: Models require specific frameworks, dependencies, and often cloud infrastructure.

These constraints make neural approaches impractical for lightweight applications such as local music players, embedded systems, or privacy-conscious users who prefer on-device processing.

## 1.2. An Alternative: Statistical Methods

This work explores whether classical statistical methods can achieve comparable results without the overhead of neural models. Instead of learning representations from data, we extract interpretable spectral features and compare songs using well-understood distance metrics.

Our pipeline combines PCA for dimensionality reduction with two similarity measures: Earth Mover's Distance (EMD), which compares songs as probability distributions by measuring the minimum transport cost between them, and Mutual Information (MI), which quantifies the shared statistical structure between two songs' feature representations without assuming a specific form of dependence. We evaluate both approaches against neural baselines on a perceptual similarity task to determine whether these lightweight methods can compete with deep learning.

# 2. Related Work

## 2.1. Neural Music Embedding Models

Deep learning dominates music representation learning. MERT [LYZ+23] uses transformer architectures with self-supervised learning, producing 768 or 1024-dimensional embeddings from models with 95M–330M parameters. CLMR [SB21] applies contrastive learning to group similar music while separating dissimilar clips.

Importantly, these models were not trained for perceptual music similarity. MERT targets music understanding tasks such as tagging, genre classification, and instrument recognition. CLMR learns to distinguish augmented versions of the same clip from different clips—a proxy task that may not align with human similarity judgments. When used for similarity, we simply compute cosine distance between their embeddings, repurposing representations designed for other objectives.

Beyond this mismatch, their size and computational requirements limit deployment, and their embeddings are opaque—when two songs are deemed similar, it is unclear which musical properties drove that judgment. We use these models as baselines to evaluate whether lighter methods can achieve comparable accuracy on a task the models were never optimized for.

## 2.2. Feature Extraction

Before any similarity computation, audio must be converted to numerical features. The Essentia library [BWG+13] provides a comprehensive set of spectral descriptors commonly used in MIR research: MFCCs, spectral centroid, flux, and others. These features are interpretable, where each captures a known acoustic property, as opposed to learned embeddings.

The challenge is that Essentia produces many features per time frame (249 in our configuration), resulting in high-dimensional time series that are expensive to compare directly. This motivates dimensionality reduction.

## 2.3. Dimensionality Reduction for Audio

We propose using PCA to reduce multivariate audio time series to a small number of principal components. We suspect that global structure after dimensionality reduction often suffices to capture discriminative information.

Alternative methods like t-SNE and UMAP optimize for local neighborhood preservation using their own distance metrics. In a pipeline that already involves distance choices (EMD ground distance, signature comparison), adding another distance-based projection would complicate interpretation. PCA's linear, variance-based projection avoids this issue.

## 2.4. Earth Mover's Distance

Rubner et al. [RTG00] introduced EMD for image retrieval, representing images as weighted point sets (signatures) and measuring the minimum transport cost between them. Orlova et al. [OVK16] extended this to music similarity, using spectral features and k-means clustering to create song signatures.

EMD is appealing because it considers the geometry of the feature space—moving mass a short distance costs less than moving it far. This aligns with perceptual intuition: songs with similar but slightly shifted spectral content should be considered similar.

## 2.5. Mutual Information

Mutual information (MI) measures statistical dependence between random variables without assuming a specific relationship form. Kraskov et al. [KSG04] proposed the KSG estimator, a k-nearest neighbor approach for continuous multivariate data.

MI has been applied to various signal processing tasks but remains underexplored for music similarity. Unlike distance-based metrics, MI captures whether two songs share information structure regardless of how features are scaled or transformed.

# 3. Problem Statement

## 3.1. Odd-One-Out Task

We evaluate music similarity using the odd-one-out task from the MagnaTagATune dataset [LWM+09]. Given a triplet of audio clips $(A, B, C)$, human annotators voted on which clip is most different from the other two. The clip receiving the most votes is the ground truth "odd one out."

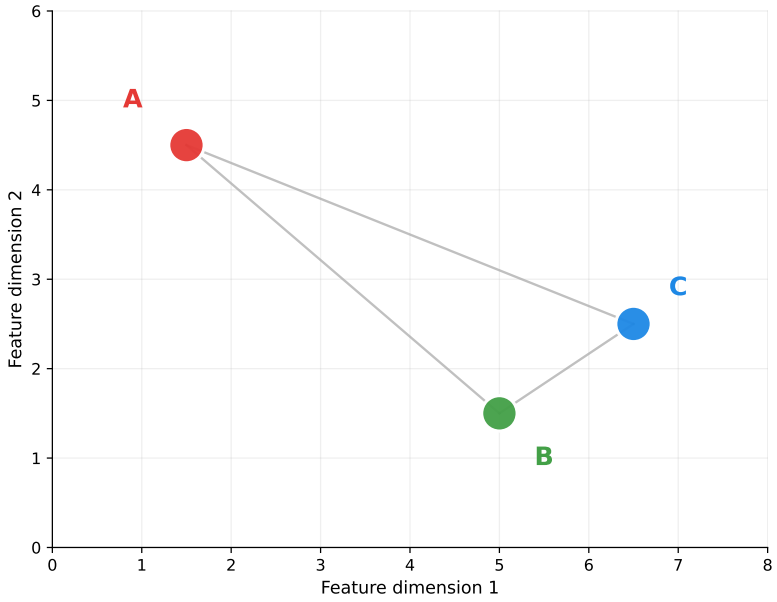



Figure 3.1.: Odd-one-out prediction: Song A has the highest average distance to B and C, making it the predicted odd one out.

**Prediction Method** To predict the odd one out, we compute distances between all pairs of songs. For song $A$, we calculate its average distance to the other two songs:

$$\bar{d}_A = \frac{d(A, B) + d(A, C)}{2} \tag{3.1}$$

We compute the same average distance for songs $B$ and $C$. The song with the **highest average distance** is predicted as the odd one out—it is the least similar to the other two songs.

The distance function $d(\cdot, \cdot)$ can be either Earth Mover's Distance (EMD) or Mutual Information (MI), which are the metrics we investigate in this work.

**Evaluation** We measure **accuracy**: the percentage of triplets where our prediction matches the human ground truth. The random baseline is 33.3%. Neural models achieve 50–55% on this task.

## 3.2. Dataset

We use the MagnaTagATune (MTAT) dataset:

- 26,000+ audio clips (30 seconds each)
- 5,000+ unique songs across diverse genres
- 533 triplet comparisons with human votes
- 190 binary tag annotations per clip

Each triplet comparison contains three clip IDs and vote counts indicating human judgments of which clip is most different. The dataset is 5,000 songs clipped multiple times to create 26,000 audio clips. The binary tags are annotated by humans in a form of descriptions, for example ['guitar', 'classical', 'soft'].

This dataset is used primarily as the working space of this project. The two main contributions of this dataset are the PCA transformation learning on 26k clips, and the odd-one-out ground truth for 1019 triplets.

It is against this odd-one-out ground truth that all the evaluations are run against.

# 4. Our Pipeline: Technical Solution

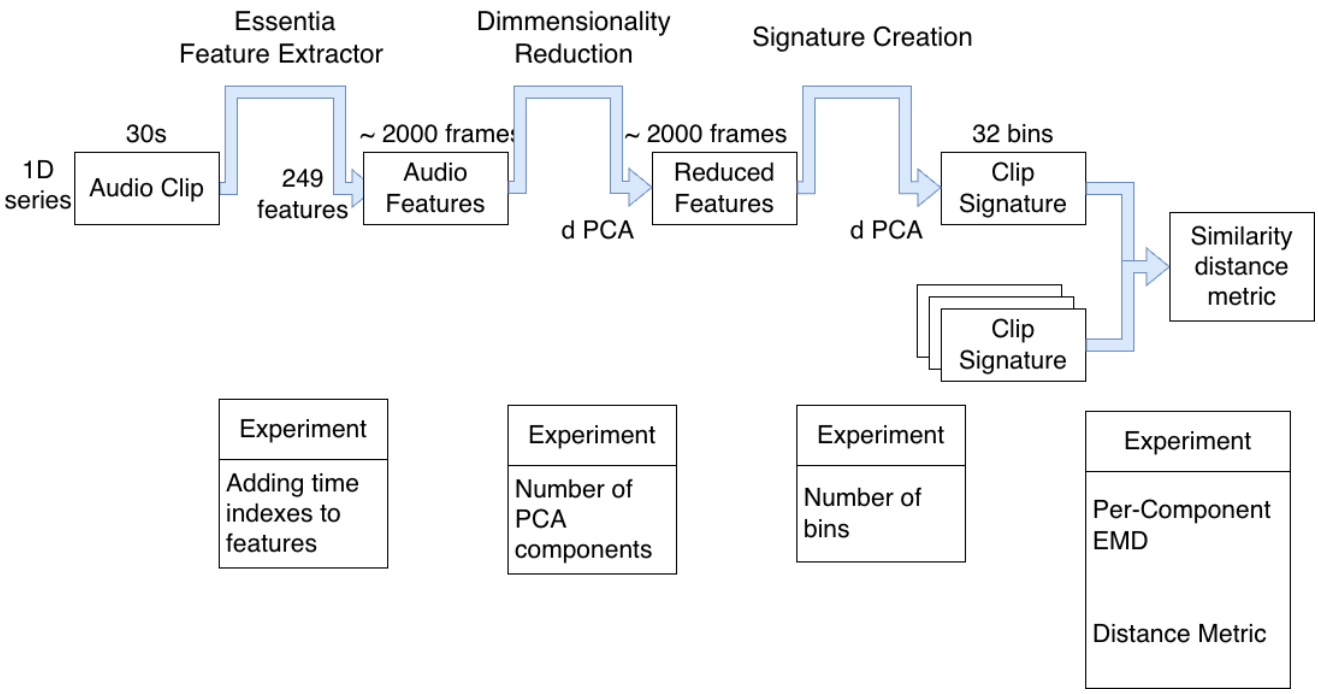



Figure 4.1.: Overview of the PCA-EMD pipeline for music similarity computation.

Our pipeline consists of three main stages: feature extraction, dimensionality reduction, and distance computation. The pipeline is shown in Figure 4.1.

## 4.1. Feature Extraction

We extract spectral features using the Essentia library [BWG+13] with the following configuration:

- Sample rate: 16,000 Hz (mono)
- Frame size: 2,048 samples
- Hop size: 533 samples (∼30 fps)
- Window: Hann

For each audio frame, we compute 249 spectral features organized into the following groups:

The features capture different aspects of audio: spectral shape, timbral features, pitch/harmony, and temporal features.

A 30-second clip yields approximately 2,000 feature frames, resulting in a matrix $\mathbf{X} \in \mathbb{R}^{249 \times 2000}$ per song.

Table 4.1.: Feature Groups and Dimensions

| Category | Features | Dims |
|---|---|---|
| *Spectral Shape* | | |
| | Bark bands (27 bands) | 27 |
| | Bark bands statistics (kurtosis, skewness, spread) | 3 |
| | Spectral centroid, crest, decrease, flux | 4 |
| | Spectral rolloff, flatness, complexity | 3 |
| | Spectral energy (total + 4 sub-bands) | 5 |
| | Spectral contrast + valleys | 14 |
| *Timbral* | | |
| | MFCCs (13 coefficients) | 13 |
| | Delta MFCCs | 13 |
| | Tristimulus | 3 |
| *Pitch & Harmony* | | |
| | Chromagram (12 pitch classes) | 12 |
| | HPCP (12 pitch classes) | 12 |
| | Pitch + confidence + salience | 3 |
| | Inharmonicity, odd-to-even ratio | 2 |
| *Temporal & Energy* | | |
| | Zero-crossing rate | 1 |
| | RMS energy, loudness, HFC | 3 |
| | Silence rates (20/30/60 dB) | 3 |
| | **Total** | **249** |

## 4.2. Dimensionality Reduction

Without dimensionality reduction, our pipeline would create signatures by running k-means directly on 249-dimensional feature vectors. K-means has complexity $O(n \cdot k \cdot d \cdot i)$ where $d$ is the dimensionality, so clustering in 249D versus 3D is roughly 80x slower—a noticeable but not prohibitive difference. Similarly, computing ground distances between centroids for EMD scales linearly with dimensionality.

The more significant concern is the curse of dimensionality: in high-dimensional spaces, distances between points become approximately equal, making it difficult to distinguish nearby from distant points. This affects both k-means clustering quality (centroids become less representative) and EMD's ability to capture meaningful differences between distributions.

Dimensionality reduction addresses both issues by projecting features into a compact subspace that preserves the most important variance. We treat each song as a multivariate time series and apply PCA to reduce the 249 features to just 2–10 principal components.

**Why PCA over t-SNE or UMAP?** Alternative methods like t-SNE and UMAP rely on their own internal distance or similarity metrics to construct the low-dimensional embedding. Our pipeline already involves distance choices at multiple levels: the ground distance for EMD (L1 or L2) and the distance metric for comparing signatures. Introducing another dimensionality reduction method with its own distance parameters would add a new experimental dimension, making the pipeline harder to interpret and results harder to attribute. PCA avoids this issue—it is a linear projection based purely on variance, introducing no additional distance assumptions.

**Training and Transformation** We fit PCA on all 26k available songs, then transform individual songs. This global fitting ensures consistent projection across all songs. Figure 4.2 shows how cumulative explained variance increases with the number of components.

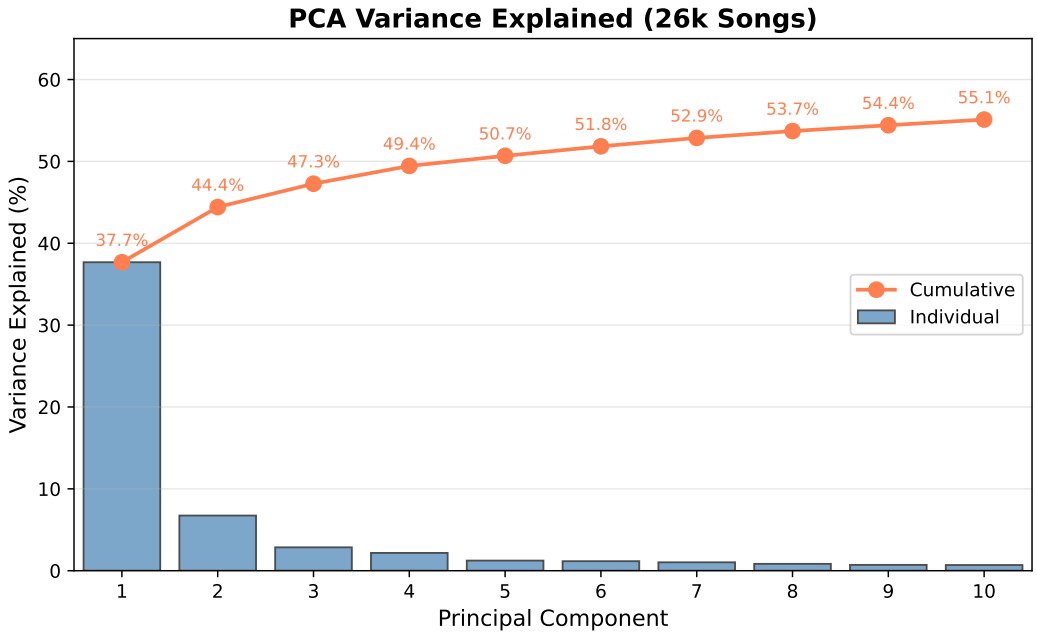



Figure 4.2.: Variance explained by PCA components fitted on 26k songs. The first component captures 37.7% of variance, with 10 components explaining 55.1% total.

**Normalization** After PCA transformation, components have different scales reflecting their explained variance—PC1 might range from $-100$ to 100, while PC2 ranges from $-10$ to 10. When computing EMD between signatures, this scale difference means that distances along high-variance components dominate the total cost, while differences

in lower-variance components contribute little. The hypothesis was that normalizing all components to unit variance would give each dimension equal influence in the distance calculation, potentially capturing information ignored before. We therefore experimented with two approaches: (1) normalizing all PCA components to unit variance, and (2) keeping the original variance structure.

**Time Frame Encoding** We also experimented with adding the frame index as an additional feature before PCA transformation, leading to 250 features per time frame. The hypothesis was that temporal order might help distinguish songs with similar spectral content but different structure. We tested both raw frame indices and normalized frame indices (scaled to match the variance of other features). This allows the similarity metrics to consider when in the song certain spectral patterns occur, not just what patterns occur.

## 4.3. Earth Mover's Distance

Earth Mover's Distance compares two probability distributions, expected to be in the form of a signature: a small set of representative points with associated weights that approximate the full distribution.

### 4.3.1. Signature Creation

A signature consists of two parts: (1) a set of *centroids* representing typical feature values in the song, and (2) *weights* indicating how much of the song each centroid represents. For our pipeline with d PCA components and 24 bins, each song's signature has shape:

- Centroids: $\mathbf{C} \in \mathbb{R}^{24 \times d}$ (24 representative points in dD space)
- Weights: $\mathbf{w} \in \mathbb{R}^{24}$ (proportion of frames per centroid, summing to 1)

There are several ways to create signatures from continuous data. *Adaptive binning* recursively splits the feature space by identifying the dimension with highest variance and partitioning at the median, creating axis-aligned regions that adapt to data density. *K-means clustering* [Llo82] groups similar frames together and uses cluster centers as centroids. As described in Section II of Lloyd's paper, each iteration computes distances between points and centroids, giving complexity $O(T \cdot n \cdot d \cdot i)$—linear in dimensionality.

#### Method Comparison: K-Means vs Adaptive Binning

K-means is the standard approach for EMD signature creation in the literature [RTG00, OVK16]. While adaptive binning is faster during initial fitting ($O(n \log k)$ vs. $O(n \cdot k \cdot d \cdot i)$ for K-means), a key optimization changes the comparison: *global K-means*.

With global K-means, we fit the clustering model once on the entire reference corpus at startup. At query time, we only need to *predict* cluster assignments for the new song's frames—a simple nearest-centroid lookup with complexity $O(T \cdot k \cdot d)$. This makes K-means significantly faster at inference:

Table 4.2.: Signature creation inference time (100 songs, 3 PCA components, 2000 frames, 32 bins). Source: `src/emd/binning_benchmark.ipynb`.

| Method | Per-Song Inference | Notes |
|---|---|---|
| Adaptive Binning | 1.45 ms | Must recompute splits per song |
| Global K-Means | 0.37 ms | Predict only (centroids pre-fitted) |

Because our application compares uploaded songs against a fixed reference library, the one-time fitting cost is amortized across all queries. We therefore use global K-means for signature creation.

### 4.3.2. Distance Computation

EMD computes the minimum cost to transform one distribution into another. Given two signatures with centroids and weights, EMD finds the optimal transport plan—how much mass to move from each source centroid to each target centroid—such that total cost is minimized. Cost is defined as mass transported multiplied by ground distance traveled.

This is formulated as a linear programming problem. The ground distance between centroids can be Euclidean (L2) or Manhattan (L1); we tested both. We solve using the HiGHS solver via SciPy, with complexity $O(n^3 \log n)$ where $n$ is the number of bins.

## 4.4. Mutual Information Estimators

Mutual information (MI) offers a complementary perspective compared to EMD: it quantifies the statistical dependence between two songs' feature representations without requiring signature creation. Given two songs with PCA-transformed feature matrices $\mathbf{X} \in \mathbb{R}^{T_1 \times d}$ and $\mathbf{Y} \in \mathbb{R}^{T_2 \times d}$ (where $T_i$ is the number of time frames and $d$ the number of PCA components), MI measures how much information the two representations share.

Formally, the mutual information between continuous random variables $X$ and $Y$ with joint density $f_{XY}$ and marginal densities $f_X$, $f_Y$ is defined as [KSG04]:

$$I(X,Y) = \int \int f_{XY}(x,y) \log \frac{f_{XY}(x,y)}{f_X(x)\, f_Y(y)}\, dx\, dy. \tag{4.1}$$

MI is always non-negative, with $I(X,Y) = 0$ if and only if $X$ and $Y$ are independent. Unlike linear correlation, MI captures arbitrary forms of statistical dependence. We implement four estimators, each making different trade-offs between computational cost, bias, and assumptions about the data. We denote the $k$-th column of $\mathbf{X}$ and $\mathbf{Y}$ as $X_k$ and $Y_k$ respectively, each a univariate time series representing a single PCA component. The true multivariate MI $I(\mathbf{X};\mathbf{Y})$ operates on all $d$ components jointly; the per-component MI $I(X_k;Y_k)$ captures only the dependence within a single matching component pair.

**Histogram Estimator** The histogram estimator serves as a baseline: the simplest way to estimate MI from finite samples [KSG04]. For each PCA component pair $(X_k, Y_k)$, the continuous value ranges are partitioned into $B$ equally spaced bins (defaulting to the Rice rule: $B = \lceil 2N^{1/3} \rceil$), and the joint and marginal probability distributions are estimated by counting points per bin. MI is then computed from the resulting contingency table. Because finite samples produce random fluctuations in bin counts that create spurious associations, we apply the Miller–Madow correction to subtract the first-order upward bias.

The histogram cannot operate in the full joint space: the number of bins grows exponentially as $B^{2d}$, so at $d = 3$ with $B = 20$, the joint histogram would require 64 million bins for only $\sim$2,000 data points, where nearly all bins are empty. Instead, we compute MI independently for each component pair and report the mean across all $d$ components. This average is not an estimate of the true multivariate $I(\mathbf{X};\mathbf{Y})$, as it ignores cross-component dependencies. It provides a floor: the accuracy achievable with the most basic approach, against which the joint estimators (KDE and KSG) can be compared.

**Kernel Density Estimation** Moon, Rajagopalan, and Lall [MRL95] proposed replacing histogram bins with kernel density estimators to obtain smooth, continuous density estimates. Instead of counting points in fixed hypercubes, each sample point contributes a Gaussian kernel to the density estimate, weighted by its distance to the evaluation point. This eliminates the histogram's sensitivity to bin placement and origin choice, and achieves a better mean square error convergence rate, particularly for small sample sizes.

The key insight is the use of the sample covariance matrix to shape the kernel: it accounts for linear dependencies among coordinates, orienting the Gaussian to align with the principal axes of variation and scaling the bandwidth proportionally along each axis. A single bandwidth parameter $h$ controls the overall smoothing range; following Moon et al., we use the optimal value derived by Silverman [Sil86] that minimizes the mean integrated square error under a Gaussian assumption.

Unlike the histogram estimator, KDE does not discretize into bins and its computational cost scales linearly with dimensionality ($O(n^2 \cdot d)$), so it can estimate densities directly in the full joint space. Given two songs with PCA-transformed matrices $\mathbf{X} \in \mathbb{R}^{T \times d}$ and $\mathbf{Y} \in \mathbb{R}^{T \times d}$, we fit three KDEs: a joint density $\hat{f}_{\mathbf{XY}}$ in the combined $2d$-dimensional space, and marginal densities $\hat{f}_{\mathbf{X}}$ and $\hat{f}_{\mathbf{Y}}$ in $d$ dimensions each. MI is then computed as:

$$\hat{I}_{\text{KDE}}(\mathbf{X}, \mathbf{Y}) = \frac{1}{N} \sum_{i=1}^{N} \log_2 \frac{\hat{f}_{\mathbf{XY}}(\mathbf{x}_i, \mathbf{y}_i)}{\hat{f}_{\mathbf{X}}(\mathbf{x}_i)\, \hat{f}_{\mathbf{Y}}(\mathbf{y}_i)}. \tag{4.2}$$

This estimates the true multivariate $I(\mathbf{X}; \mathbf{Y})$, capturing cross-component dependencies that the histogram's per-component average misses. At $d = 3$ (our optimal configuration), the joint space is 6-dimensional with $\sim$2,000 sample points.

**KSG Estimator** The KSG estimator [KSG04] takes a different approach based on $k$-nearest neighbor distances. While MI can theoretically be expressed via Shannon entropy as $I(X, Y) = H(X) + H(Y) - H(X, Y)$, the KSG algorithm explicitly avoids estimating these three terms independently, as doing so with a fixed $k$ would result in mismatched distance scales and non-canceling systematic errors. KSG first finds the distance $\varepsilon(i)$ to the $k$-th nearest neighbor of each point in the joint space, then counts the number of neighbors ($n_x$, $n_y$) in the marginal spaces whose distance is *strictly less than* $\varepsilon(i)/2$. By locking the distance scale across the joint and marginal spaces, the density approximation errors are constructed to approximately cancel in the final MI computation.

The key idea is that the neighborhoods adapt to the local data density: where data is more dense, $\varepsilon$ is smaller and the estimate resolves finer-grained structure.

Like KDE, KSG operates on the full $d$-dimensional vectors $\mathbf{X}$ and $\mathbf{Y}$ jointly, constructing nearest-neighbor trees in the combined $2d$-dimensional space. Because KSG uses adaptive nearest-neighbor distances rather than a fixed kernel, it does not require bandwidth selection and the underlying density approximation errors cancel by construction rather than contributing to the final estimate. This makes KSG more robust in higher dimensions where kernel-based density estimates degrade. Both KDE and KSG estimate the true multivariate $I(\mathbf{X}; \mathbf{Y})$, capturing cross-component dependencies that our histogram estimator's per-component decomposition misses.

**Jackknife Mutual Information** Bandwidth sensitivity and boundary bias are known limitations of KDE-based MI estimation. The JMI estimator [ZXT18] addresses both through a jackknife approach.

The method proceeds in three stages. First, a marginal transformation maps both variables to the unit interval using their empirical cumulative distribution functions. This standardizes the support and reduces technical complexity, since the estimator now works with uniformly distributed variables on a bounded domain rather than the original marginals. Second, the bandwidths used in the joint and marginal density estimators are equalized. Because the variables are now uniformly distributed, the estimator simplifies the bandwidth matrices to a single common scalar bandwidth across all dimensions. This cancellation structure automatically mitigates boundary bias, which is problematic when marginal density estimates appear in the denominator of the MI formula. Third, a leave-one-out (jackknife) KDE is used to estimate MI: for each sample point, the density is evaluated using all other points, preventing the estimate from overfitting to the observed data. The final MI is then computed by evaluating this jackknife objective over a grid of candidate bandwidths and selecting the maximum value. The resulting estimator is data-driven, automatically corrects boundary bias, and requires no predetermined tuning parameters.

Unlike the histogram and standard KDE estimators, JMI was shown by Zeng et al. to achieve smaller mean-squared errors across a range of dependence models and sample sizes, and to provide a more stable test for independence than competing methods including distance correlation.

## 4.5. Scalability

**Preprocessing Cost** Before the pipeline can serve queries, a one-time preprocessing step is required: extracting spectral features from the full MagnaTagATune dataset ($\sim$26,000 clips), fitting a scaler and PCA model,

transforming all reference songs, and fitting a global K-Means model on the transformed features. Both the scaler and PCA model are fitted incrementally in batches, keeping memory usage constant regardless of corpus size. Table 4.3 breaks down the computation time by stage. All measurements were performed on Google Colab Pro (8 vCPUs, 51 GB RAM). This cost is incurred only once; the resulting models and pre-transformed features are reused for all subsequent queries.

Table 4.3.: Preprocessing Time by Stage (25,860 clips)

| Stage | Time | Details |
|---|---|---|
| Feature extraction | ∼8 min | 8 workers, 55.5 songs/s |
| StandardScaler fitting | ∼8 min | 13 batches of 2,000 songs |
| IncrementalPCA fitting | ∼33 min | 13 batches, $K = 10$ components |
| Transform and save | ∼8 min | 13 batches, same batch size |
| **Total** | **∼57 min** | |

**Storage Cost** Table 4.4 summarizes the per-song and corpus-scale storage requirements. The raw spectral features dominate peak storage due to their uncompressed dimensional representation, but are deleted after the transformation step, reducing the final size to approximately 2 GB. The PCA-transformed features are smaller than the source MP3 files because the dimensionality reduction compensates for the overhead of uncompressed float32 storage.

Table 4.4.: Storage Requirements

| Artifact | Per Song | Full Corpus |
|---|---|---|
| Source MP3 (32 kbps) | ∼114 KB | 3.0 GB |
| Raw spectral features | ∼2.0 MB | 51 GB (temporary) |
| PCA-transformed features ($K$=10) | ∼80 KB | 2.0 GB |
| Scaler + PCA model + manifest | — | ∼7 MB |
| Pre-fitted K-Means model (24 clusters) | — | ∼1 KB |
| Peak disk usage | | ∼56 GB |
| Final disk usage | | ∼2.0 GB |

Additionally, a pre-fitted global K-Means model (24 clusters, 5 dimensions) is stored for EMD signature creation, adding negligible overhead (∼1 KB).

**Per-Song Compression** At query time, the per-song compression is significant. A 30-second MP3 file (∼500 KB) produces ∼2 MB of raw spectral features (249 dimensions × 2,000 frames). After PCA transformation, this reduces to ∼24 KB for 3 components or ∼64 KB for 8 components. The final k-means signature is less than 1 KB regardless of configuration. The PCA model itself is 25 KB and can be reused for all songs. The pre-fitted global K-Means model stores only the 24 cluster centroids in 5D space, requiring just 1 KB—this enables fast signature creation at query time via centroid lookup rather than re-fitting.

## 4.6. Generalizability

**New Songs** The pipeline processes audio files outside the original training set without modification. Feature extraction via Essentia is deterministic and independent of the training data. The pre-fitted PCA model transforms new songs into the same feature space, enabling direct comparison with existing signatures.

**Different Audio Lengths** The pipeline naturally handles varying audio lengths. Shorter clips produce fewer frames, longer clips produce more frames, but feature extraction and PCA transformation work identically regardless. The k-means signature step compresses any number of frames into exactly 24 centroids, so the final signature has the same shape for all songs.

# 5. Experimental Results

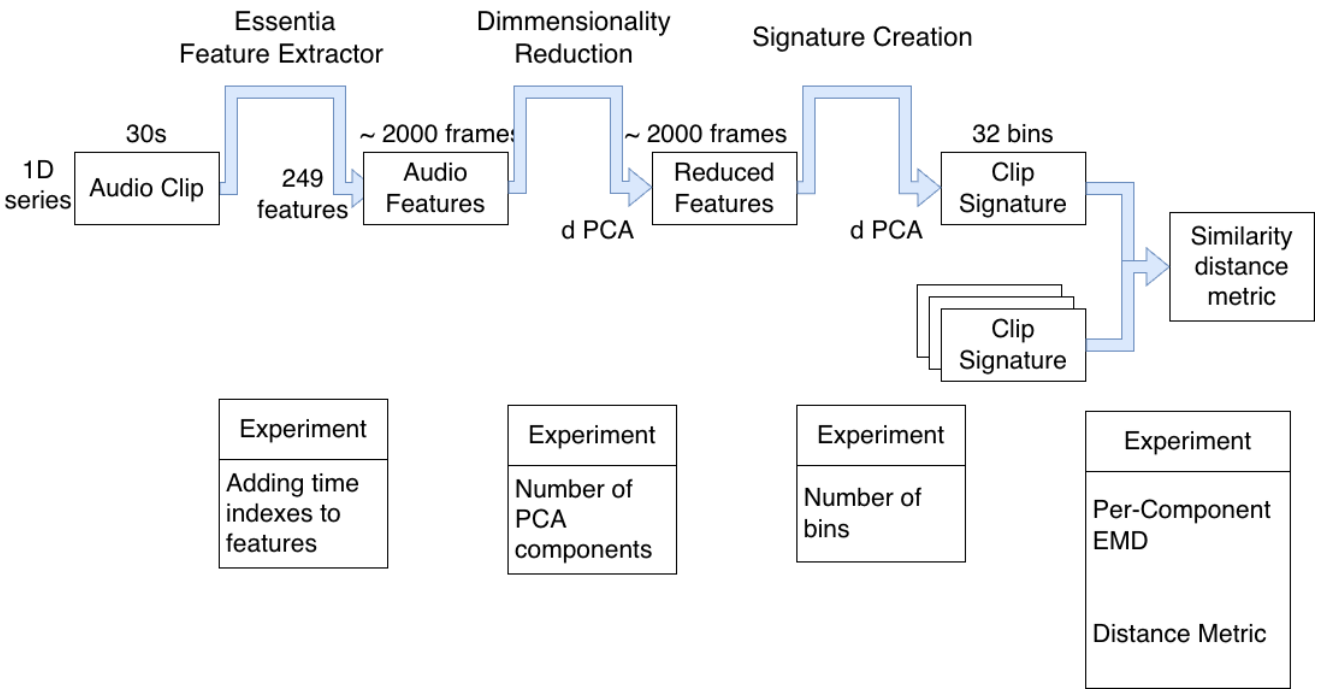



Figure 5.1.: Pipeline stages where experiments were conducted.

## 5.1. Time Dimension Experiment

We tested adding the frame index as an additional feature on the best configuration (5D PCA, 24 bins, cityblock):

Table 5.1.: Effect of Time Dimension Weight (5D PCA)

| Time Weight | Accuracy (%) |
|---|---|
| 0 (baseline) | 59.10 |
| 0.25 | 59.10 |
| 0.50 | 58.72 |
| 1.00 | 59.10 |
| 2.00 | 58.72 |

Note: The time dimension experiment uses K-means fitted on comparison songs only (1019 songs), not the full 25k reference corpus as it needs re-fitting of the K-means model, which explains the lower baseline accuracy compared to the grid search (60.79%). Adding time does not improve accuracy, suggesting temporal structure does not contribute to perceived similarity in this task. For this experiment we decide not to add the time-dimension on the final EMD pipeline.

## 5.2. Effect of PCA Components

We tested 2, 3, and 5 PCA components across multiple configurations (varying bins and distance metric). Table below shows both the average and best accuracy for each component count.

Table 5.2.: Accuracy vs. Number of PCA Components

| Components | Avg (%) | Best (%) | Best Configuration |
|---|---|---|---|
| 2 | 59.01 | 59.66 | 32 bins, cityblock |
| 3 | 58.63 | 58.91 | 32 bins, euclidean |
| 5 | **59.95** | **60.79** | 24 bins, cityblock |

The number of PCA components has minimal impact on average accuracy, suggesting EMD performance is robust to dimensionality within this range. However, the best configuration uses 5 components, achieving 60.79%.

## 5.3. Effect of Bin Count

We tested 24 and 32 bins across all configurations. Both bin counts perform similarly:

Table 5.3.: Best Accuracy by Bin Count (cityblock)

| Bins | Best at 2D (%) | Best at 5D (%) |
|---|---|---|
| 24 | 59.29 | **60.79** |
| 32 | **59.66** | 60.23 |

The best overall result (60.79%) uses 24 bins, though the difference between 24 and 32 bins is marginal.

## 5.4. Per-Component EMD

We experimented with computing EMD separately for each PCA component (treating each as a 1D distribution), then aggregating the per-component distances.

Table 5.4.: Per-Component EMD vs. Standard Approach (2D, 24 bins, cityblock)

| Method | Accuracy (%) |
|---|---|
| Standard EMD (2D joint) | **59.29** |
| Per-component EMD (2D, unweighted) | 58.35 |

## 5.5. Effect of Distance Metric

We compared Euclidean (L2) and Manhattan (L1/cityblock) distance metrics for computing EMD across all configurations:

Table 5.5.: Best Accuracy by Distance Metric and Components (24 bins)

| Components | Euclidean (%) | Cityblock (%) |
|---|---|---|
| 2 | 57.97 | **59.29** |
| 3 | 58.16 | **58.91** |
| 5 | 59.29 | **60.79** |

Cityblock (L1) distance consistently outperforms Euclidean (L2) across all tested dimensions. The best overall result (60.79%) is achieved with 5 components and cityblock distance.

## 5.6. Computational Efficiency

Table 5.6 shows the time per triplet for different methods. Timing data is from `src/outputs/evaluation/` (neural models) and `src/evaluation/emd/emd_timing_results.json` (EMD).

Table 5.6.: Time per Triplet Comparison

| Method | Time/Triplet (ms) | Total Time (533 triplets) |
|---|---|---|
| MERT-330M | 10115 | 89.9 min |
| MERT-95M | 4633 | 41.2 min |
| CLMR | 424 | 3.8 min |
| PCA-EMD (3D, 24 bins) | 5.92 | 3.2 s |

Table 5.7 shows the computational cost of MI estimators, measured on the same 533 triplets using 15 PCA components. Data from `src/evaluation/mi/mi_timing_benchmark.json`.

Table 5.7.: MI Estimator Timing (15 PCA components)

| Method | Time/Triplet (ms) | Total Time | Accuracy (%) |
|---|---|---|---|
| MI-Histogram | 3.17 | 1.7 s | 46.90 |
| MI-KSG (std=False) | 70.7 | 37.7 s | 54.97 |
| MI-KSG (std=True) | 133.8 | 71.3 s | 46.53 |
| MI-KDE | 143.8 | 76.6 s | 36.96 |

PCA-EMD (3.2s total) is faster than all MI estimators except MI-Histogram. MI-KSG without standardization is twice as fast as with standardization and achieves much higher accuracy.

## 5.7. MI: Effect of Standardization

We now evaluate Mutual Information as a complementary similarity measure compared to EMD. MI quantifies shared statistical structure between two songs' feature representations (Equation 4.1), capturing arbitrary forms of dependence rather than distributional distance. At default parameters (3 PCA components with standardization), MI estimators scored between 40% and 45%, well below EMD (60.79%). The gap between MI and EMD motivated a systematic parameter search, beginning with the KSG estimator's internal standardization.

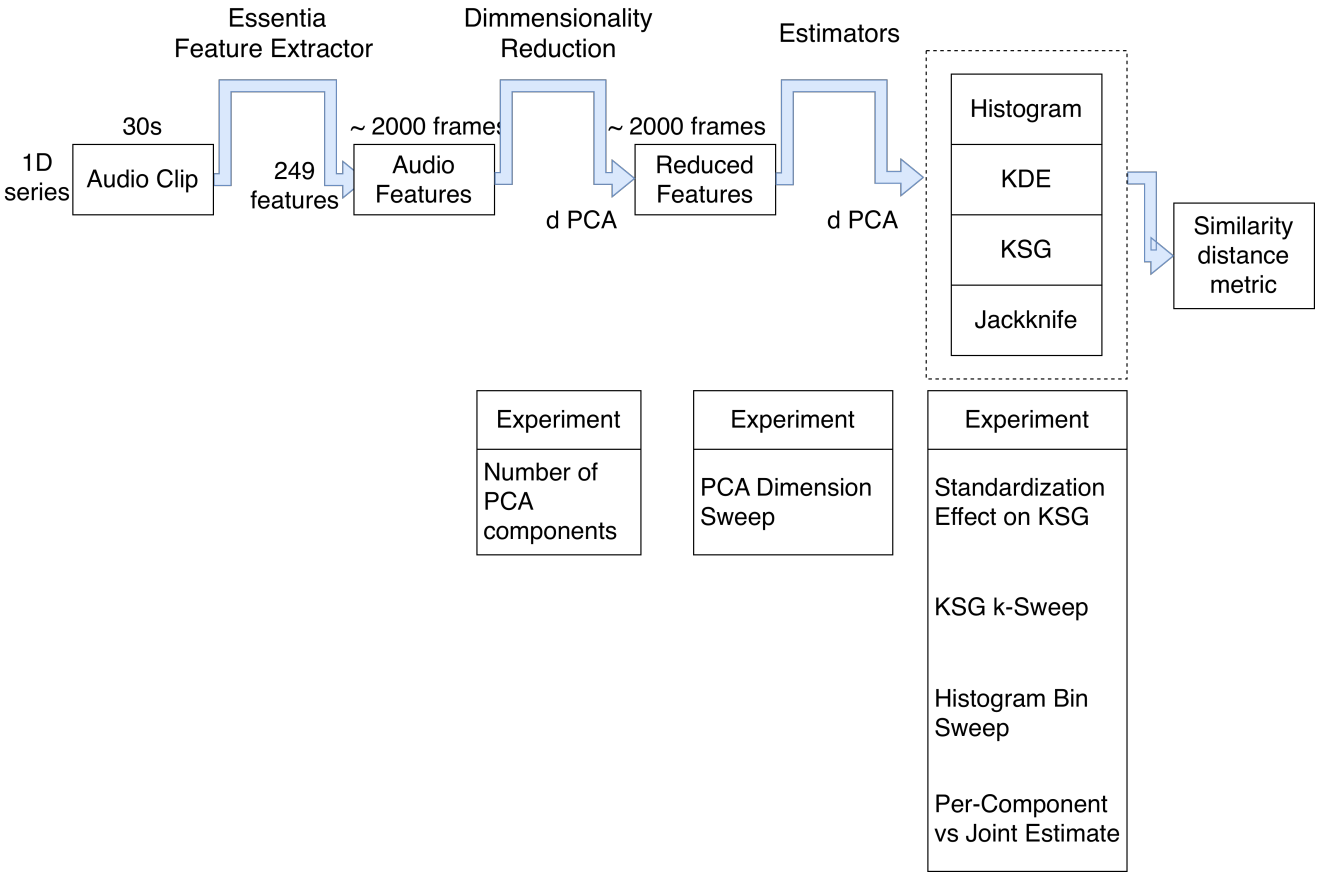



Figure 5.2.: MI pipeline stages where experiments were conducted.

The KSG estimator uses Chebyshev ($L_\infty$) distance in the joint space. Its `standardize` parameter centers and scales features to unit variance before computing distances. Our initial hypothesis predicted standardization would help by balancing $L_\infty$ distances across components. The result was the opposite, as shown in Figure 5.2.

Table 5.8.: Effect of KSG Standardization (3 PCA Components, $k = 5$)

| **Standardize** | **Accuracy (%)** |
|---|---|
| True | 45.03 |
| **False** | **53.66** |

Disabling standardization improved accuracy by **8.63%**. PCA components have a natural variance hierarchy: component 1 captures the most perceptually relevant spectral variation. Standardization equalizes all components to unit variance, destroying this information ordering and treating high-variance (informative) components the same as low-variance (noisy) ones.

## 5.8. MI: Effect of PCA Dimensions

We tested KSG without standardization across PCA dimensions 2–20, with a fine-grained search around the peak. Table 5.9 shows the results alongside the standard (standardized) KSG for comparison.

Table 5.9.: KSG Accuracy vs. PCA Dimensions ($k = 5$)

| **Components** | **KSG std=True (%)** | **KSG std=False (%)** |
|---|---|---|
| 2 | 44.47 | 52.53 |
| 3 | 45.03 | 53.66 |
| 5 | 43.15 | 54.03 |
| **10+** | **46.53** | **54.97** |

Disabling standardization consistently improves accuracy. KSG without standardization plateaus at 10+ components (54.97% with $k = 5$). Further optimization of $k$ at higher dimensions yields the best result: 57.41% at 15 components with $k = 7$. KSG's $k$-nearest neighbor approach uses adaptive neighborhoods that scale with local density, providing partial robustness against the curse of dimensionality.

## 5.9. MI: KSG Neighbor ($k$) Optimization

We explored with different values of $k$ at 3 PCA components and at the optimal 15 PCA components.

Table 5.10.: KSG Accuracy vs. $k$ (standardize=False)

| $k$ | **3 Components (%)** | **15 Components (%)** |
|---|---|---|
| 1 | 52.35 | 56.85 |
| 3 | 51.97 | 55.91 |
| 5 | 53.66 | 54.97 |
| 6 | 53.66 | 57.04 |
| **7** | **53.85** | **57.41** |
| 8 | 53.47 | 56.10 |
| 10 | 53.10 | 55.35 |
| 20 | 52.16 | 54.03 |

At 3 dimensions, $k = 7$ is optimal (53.85%). At 15 dimensions, the optimum is also $k = 7$, achieving the best MI result of **57.41%**. Accuracy degrades for $k \geq 10$, indicating over-smoothing.

## 5.10. MI: Per-Component vs. Joint Estimation

MI can be estimated in two ways: *jointly*, using all PCA components simultaneously in one high-dimensional estimate, or *per-component*, computing MI independently for each component pair and summing the results. Joint estimation captures cross-component dependencies but must operate in a $2d$-dimensional space; per-component estimation avoids this dimensionality cost but assumes components are independent.

Joint estimation consistently outperforms per-component estimation across all tested dimensions with KSG (standardize=False), with the gap widening at higher dimensions:

Table 5.11.: Joint vs. Per-Component KSG (standardize=False, $k = 5$)

| Components | Joint (%) | Per-Component (%) |
|---|---|---|
| 3 | **53.66** | 49.16 |
| 10 | **54.97** | 49.91 |

Joint estimation outperforms per-component by 4–5 percentage points, demonstrating that cross-component dependencies carry significant discriminative information. The KSG estimator exploits these dependencies while maintaining robustness in moderate-dimensional spaces.

## 5.11. MI: Combining Estimator Predictions

Since different MI configurations disagree on specific triplets, we tested whether combining predictions from multiple configurations could improve accuracy. For each triplet, three configurations independently predict the odd one out, and the final prediction is the answer chosen by at least two of the three.

Table 5.12.: Accuracy of Combined MI Configurations

| Configurations | Accuracy (%) |
|---|---|
| **Best individual (KSG 15d, $k$=7, no_std)** | **57.41** |
| KSG 10d $k$=5 + KSG 15d $k$=7 + KSG 5d $k$=5 | 56.47 |
| Ensemble + KSG-pc 10d | 56.47 |

The best individual KSG configuration (57.41%) outperforms the ensemble (56.47%). This indicates that the configurations make correlated errors, and majority voting cannot improve upon the best single estimator.

## 5.12. Final Comparison: EMD vs. MI vs. Neural Baselines

Table 5.13 presents the best accuracy achieved by each method across all experiments.
All timing measurements were performed on a MacBook Air with Apple M4 chip (10 cores: 4 performance, 6 efficiency) and 16 GB RAM. Sources: neural baselines from `src/outputs/evaluation/`, EMD from `src/evaluation/emd/emd_evaluation_results.json`, MI from `src/evaluation/mi/mi_evaluation_results.json`.

Table 5.13.: Complete Accuracy Comparison: All Methods

| **Method** | **Configuration** | **Accuracy (%)** | **Time** |
|---|---|---|---|
| Random Baseline | — | 33.33 | — |
| *Neural Baselines* | | | |
| MERT-95M | cosine similarity | 52.91 | 41.2 min |
| CLMR | cosine similarity | 54.41 | 3.8 min |
| MERT-330M | cosine similarity | 60.98 | 89.9 min |
| *EMD* | | | |
| PCA-EMD (2D, cityblock) | 2d, 32 bins | 59.66 | 3.2 s |
| PCA-EMD (3D, cityblock) | 3d, 24 bins | 58.91 | 3.2 s |
| **PCA-EMD (5D, cityblock)** | **5d, 24 bins** | **60.79** | **3.2 s** |
| *MI Estimators* | | | |
| MI-KDE | 15d, joint | 36.96 | $\sim$77 s |
| MI-Histogram | 15d, auto bins | 46.90 | $\sim$1.7 s |
| MI-KSG (std=True) | 15d, $k = 5$ | 46.53 | $\sim$71 s |
| **MI-KSG (std=False)** | **15d,** $k = 7$ | **57.41** | $\sim$38 s |
| MI-Ensemble | 3$\times$ KSG vote | 56.47 | $\sim$100 s |

1. **PCA-EMD matches MERT-330M**: Our best EMD configuration (5D PCA, 24 bins, L1 distance) achieves 60.79% accuracy, comparable to MERT-330M (60.98%) but $\sim$1700$\times$ faster (3.2s vs. 89.9 min).

2. **MI-KSG outperforms smaller neural models**: The best MI configuration (KSG, 15d, $k = 7$, no standardization) reaches 57.41%, surpassing both MERT-95M (52.91%) and CLMR (54.41%).

3. **Speed–accuracy trade-off favors statistical methods**: Both EMD (3.2s) and MI-KSG (38s) are orders of magnitude faster than neural baselines while achieving competitive accuracy.

# 6. User Application

To demonstrate the practical applicability of our similarity pipeline, we developed a full-stack web application that allows users to upload audio files and discover similar songs from a reference library. The application exposes the PCA-based similarity computation described in Chapter 4 through an interactive interface, requiring no knowledge of the underlying algorithms. The application processes individual user-provided songs on demand, comparing them against all reference songs in real time.

## 6.1. System Architecture

The application follows a client-server architecture with a separation of concerns between the user interface and the computation engine. Figure 6.1 shows the high-level component diagram.

The frontend is a single-page application built with Next.js [Ver16], providing a guided wizard interface for song upload, parameter configuration, and result exploration. The backend is a Python REST API built with FastAPI [Ram18], implementing the feature extraction, dimensionality reduction, and similarity computation pipeline.

The backend follows a layered architecture with a separation of concerns between the different layers:

- **Routers** handle HTTP request parsing, response formatting, and status codes. They contain no business logic and delegate all computation to services.
- **Services** implement the similarity algorithms and orchestration logic, independent of HTTP or data storage details.
- **Repositories** encapsulate data access, including asynchronous file I/O for uploaded songs and loading of pre-computed reference models.

**Reference Dataset** At startup, the backend loads a pre-computed reference dataset derived from the MagnaTagATune corpus into memory, which is used for similarity computation:

- A `StandardScaler` fitted on the training data for feature normalization
- A PCA model fitted on the same data, supporting up to 10 components
- Pre-transformed feature matrices for all reference songs (as NumPy arrays)
- A manifest mapping song identifiers to original filenames and audio file paths

This approach prioritizes query performance over startup time: once loaded, similarity searches require no disk access for reference data. The PCA model and scaler are compact ($\sim$31 KB), while the full set of reference features is cached in memory for fast comparison.

**Asynchronous Query Processing** Computing similarity against the full reference library takes noticeable time, particularly when multiple estimators are selected. A synchronous request-response cycle would leave users waiting with no feedback until all computations finish. To address this, the backend implements an asynchronous job-based pattern.

When the user submits an analysis request, the router creates a job record and responds immediately with HTTP status 202 (Accepted) and a job identifier. The service layer then dispatches the computation across parallel background threads:

1. The uploaded audio is decoded and spectral features are extracted using Essentia [BWG+13].
2. Features are normalized with the pre-fitted scaler and projected onto the PCA subspace using scikit-learn [PVG+11].
3. Each selected estimator runs in a separate thread via a thread pool executor, comparing the query song against all reference songs.
4. As each estimator completes, it updates the job record with its results and elapsed time.

The frontend polls the job endpoint every two seconds. When new results become available, they are rendered immediately—users see the first estimator's output while slower estimators are still computing. This progressive display reduces perceived latency and allows users to begin evaluating results before the full computation finishes.

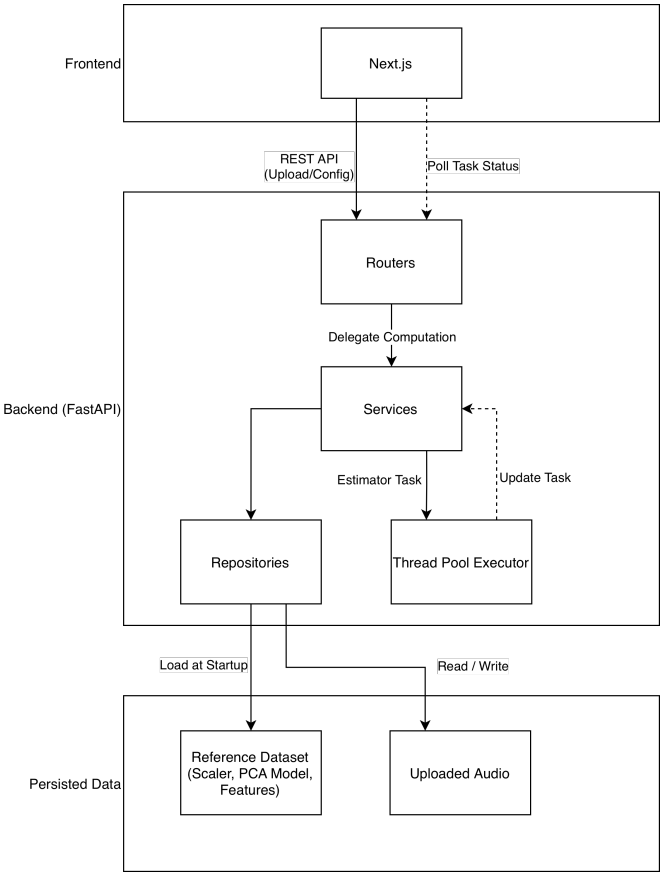
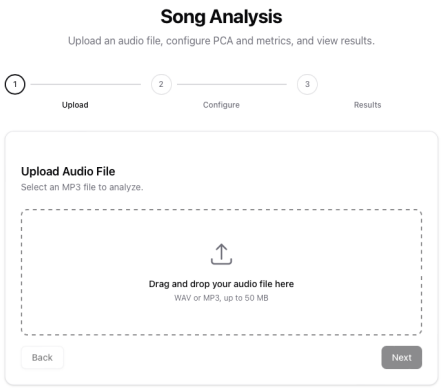



Figure 6.1.: System architecture of the MultiCorr application. The frontend communicates with the backend via a REST API. The backend loads pre-computed reference data at startup and processes similarity queries asynchronously in parallel background threads.

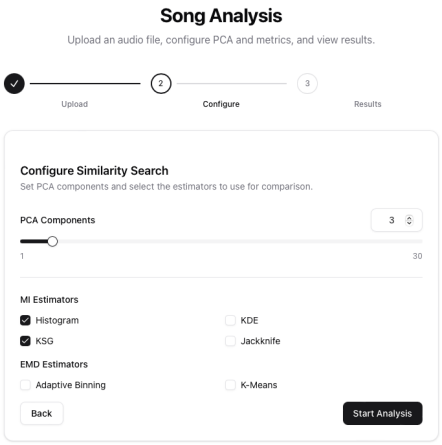
## 6.2. User Workflow

The interface guides users through a three-step wizard: upload, configure, and results. Figure 6.2 shows the interface at each stage.

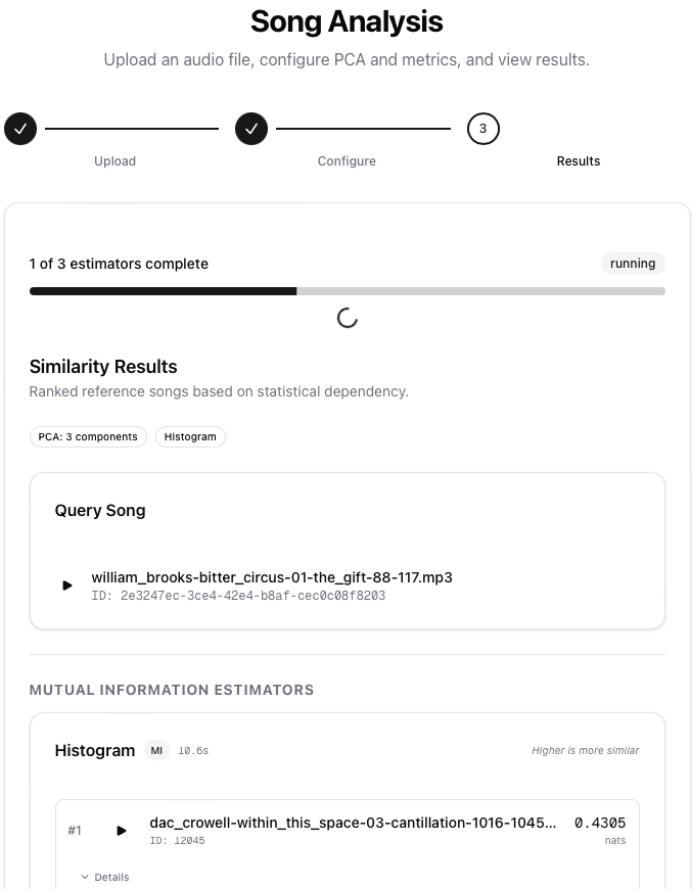
**Step 1: Upload** Users upload a WAV or MP3 file via drag-and-drop or file browser. The frontend validates the file format before transmitting it to the backend, which stores the file and returns a unique identifier for subsequent analysis requests.




(a) Upload: drag-and-drop or file selection




(b) Configure: PCA components and estimator selection




(c) Results: ranked similar songs with audio playback

Figure 6.2.: The three-step analysis wizard. Users upload an audio file, configure analysis parameters, and receive a ranked list of similar songs with playback controls.

**Step 2: Configure** Users select two parameters:

- **PCA components** (1–10): the number of principal components to retain after dimensionality reduction, adjustable via a slider with direct numeric input.
- **Similarity estimators**: one or more metrics to compute. Users can select from four mutual information estimators (histogram binning, KDE, KSG, jackknife) and two earth mover's distance estimators (adaptive binning, k-means).

Multiple estimators can be selected simultaneously, allowing users to compare how different methodologies rank the same reference songs in a single analysis run.

**Step 3: Results** Results are grouped by methodology (MI or EMD). For each estimator, the interface displays a ranked list of the most similar reference songs, according to the raw metric value for interpretability, the computation time, and audio playback controls for both the query and each result song, allowing for direct auditory comparison.

# 7. Conclusion

## 7.1. Summary

This project investigates whether classical statistical methods can compete with neural models for music similarity. We developed a pipeline combining spectral feature extraction, PCA dimensionality reduction, and two complementary similarity measures: Earth Mover's Distance (EMD) and Mutual Information (MI). Our best EMD configuration (5 PCA components, 24 bins, L1 distance) achieves 60.79% accuracy on the MagnaTagATune odd-one-out task, comparable to MERT-330M (60.98%) but significantly faster (3.2s vs 89.9 min). A systematic evaluation of four MI estimators reveals that the KSG estimator with optimized parameters (15 components, $k = 7$, no standardization) reaches **57.41%**, outperforming MERT-95M (52.91%) and CLMR (54.41%).

## 7.2. Experimental Findings

**EMD findings.** Performance peaks at 5 PCA components with 24 bins, achieving 60.79% accuracy.

**MI findings.** The most impactful discovery is that disabling KSG's internal feature standardization improves accuracy. MI benefits from higher dimensionality (optimal at 10–15 components vs. 5 for EMD). The best MI configuration (KSG, 15d, $k = 7$, no standardization) achieves 57.41% accuracy.

## 7.3. Limitations

- **Small evaluation set**: With only 533 triplets (87 with tied votes), distinguishing between configurations with similar accuracy is statistically difficult.
- **Feature imbalance**: The 249 Essentia features unevenly represent musical aspects—spectral characteristics span dozens of features while rhythm may be captured by few. This imbalance affects which properties dominate the PCA transformation.
- **Single dataset**: Results are specific to MagnaTagATune and may not generalize to other music collections or similarity definitions.
- **MI computational cost**: While MI-Histogram is faster than EMD ($\sim$1.7s vs. 3.2s), the best-performing MI configuration (KSG, 15 components, $k = 7$) requires $\sim$38s, making it slower than EMD despite achieving slightly lower accuracy.

## 7.4. Future Work

Each limitation suggests a direction for future research. Evaluation on larger datasets with more triplet comparisons would provide statistical power to distinguish between EMD and MI. Investigating feature weighting or selection could address the imbalance problem.

# A. Complete Experimental Results

Table A.1 contains all configurations tested during our grid search evaluation.

Table A.1.: Complete Grid Search Results

| Method | Comp. | Bins | Norm. | Distance | Acc. (%) |
|---|---|---|---|---|---|
| *Neural Baselines* | | | | | |
| CLAP | — | — | — | cosine | 50.47 |
| CLMR | — | — | — | cosine | 54.41 |
| MERT-95M | — | — | — | cosine | 52.91 |
| MERT-330M | — | — | — | cosine | 60.98 |
| *PCA-EMD Configurations* | | | | | |
| **PCA-EMD** | **5** | **24** | **No** | **cityblock** | **60.79** |
| PCA-EMD | 5 | 32 | No | cityblock | 60.23 |
| PCA-EMD | 2 | 32 | No | cityblock | 59.66 |
| PCA-EMD | 5 | 32 | No | euclidean | 59.47 |
| PCA-EMD | 2 | 24 | No | cityblock | 59.29 |
| PCA-EMD | 5 | 24 | No | euclidean | 59.29 |
| PCA-EMD | 2 | 32 | No | euclidean | 59.10 |
| PCA-EMD | 3 | 32 | No | euclidean | 58.91 |
| PCA-EMD | 3 | 24 | No | cityblock | 58.91 |
| PCA-EMD | 3 | 32 | No | cityblock | 58.54 |
| PCA-EMD | 3 | 24 | No | euclidean | 58.16 |
| PCA-EMD | 2 | 24 | No | euclidean | 57.97 |
| PCA-EMD | 3 | 32 | Yes | euclidean | 53.47 |
| PCA-EMD | 3 | 32 | Yes | cityblock | 52.72 |
| PCA-EMD | 2 | 32 | Yes | euclidean | 52.53 |
| PCA-EMD | 2 | 32 | Yes | cityblock | 52.35 |
| PCA-EMD | 5 | 24 | Yes | euclidean | 50.47 |
| PCA-EMD | 2 | 24 | Yes | cityblock | 50.09 |
| PCA-EMD | 5 | 32 | Yes | cityblock | 48.78 |
| PCA-EMD | 2 | 24 | Yes | euclidean | 48.78 |
| PCA-EMD | 5 | 24 | Yes | cityblock | 48.59 |
| PCA-EMD | 5 | 32 | Yes | euclidean | 48.41 |
| PCA-EMD | 3 | 24 | Yes | euclidean | 44.28 |
| PCA-EMD | 3 | 24 | Yes | cityblock | 43.90 |
| *PCA-EMD + Time Dimension (5D, 24 bins, cityblock)* | | | | | |
| PCA-EMD+time (w=0.00) | 5 | 24 | No | cityblock | 59.10 |
| PCA-EMD+time (w=0.25) | 5 | 24 | No | cityblock | 59.10 |
| PCA-EMD+time (w=0.50) | 5 | 24 | No | cityblock | 58.72 |
| PCA-EMD+time (w=1.00) | 5 | 24 | No | cityblock | 59.10 |
| PCA-EMD+time (w=2.00) | 5 | 24 | No | cityblock | 58.72 |

Table A.2 contains all MI configurations tested.

Table A.2.: Complete MI Experimental Results

| Estimator | Parameters | Comp. | Mode | Acc. (%) |
|---|---|---|---|---|
| *Base Comparison (3d)* | | | | |
| Histogram | auto bins | 3 | per-comp | 45.40 |
| KDE | $n_{\text{eval}}$=2000 | 3 | joint | 40.34 |
| KSG | $k$=5, std=True | 3 | joint | 45.03 |
| KSG | $k$=5, std=False | 3 | joint | 53.66 |
| *Base Comparison (15d)* | | | | |
| Histogram | auto bins | 15 | per-comp | 46.90 |
| KDE | $n_{\text{eval}}$=2000 | 15 | joint | 36.96 |
| KSG | $k$=5, std=True | 15 | joint | 46.53 |
| KSG | $k$=5, std=False | 15 | joint | 54.97 |
| *KSG std=False PCA Sweep ($k$=5)* | | | | |
| KSG | $k$=5, std=False | 2 | joint | 52.53 |
| KSG | $k$=5, std=False | 3 | joint | 53.66 |
| KSG | $k$=5, std=False | 5 | joint | 54.03 |
| KSG | $k$=5, std=False | 10+ | joint | 54.97 |
| *Histogram Bin Sweep (3d)* | | | | |
| Histogram | $B$=8 | 3 | per-comp | 41.28 |
| Histogram | $B$=16 | 3 | per-comp | 43.71 |
| Histogram | $B$=24 | 3 | per-comp | 43.34 |
| Histogram | $B$=32 | 3 | per-comp | 45.40 |
| Histogram | $B$=48 | 3 | per-comp | 45.40 |
| Histogram | $B$=64 | 3 | per-comp | 45.97 |
| *KSG $k$-Sweep (3d, std=False)* | | | | |
| KSG | $k$=1, std=False | 3 | joint | 52.35 |
| KSG | $k$=3, std=False | 3 | joint | 51.97 |
| KSG | $k$=5, std=False | 3 | joint | 53.66 |
| **KSG** | **k=7, std=False** | **3** | **joint** | **53.85** |
| KSG | $k$=10, std=False | 3 | joint | 53.10 |
| KSG | $k$=20, std=False | 3 | joint | 52.16 |
| *KSG $k$-Sweep (15d, std=False)* | | | | |
| KSG | $k$=1, std=False | 15 | joint | 56.85 |
| KSG | $k$=3, std=False | 15 | joint | 55.91 |
| KSG | $k$=5, std=False | 15 | joint | 54.97 |
| KSG | $k$=6, std=False | 15 | joint | 57.04 |
| **KSG** | **k=7, std=False** | **15** | **joint** | **57.41** |
| KSG | $k$=8, std=False | 15 | joint | 56.10 |
| KSG | $k$=10, std=False | 15 | joint | 55.35 |
| KSG | $k$=20, std=False | 15 | joint | 54.03 |
| *KSG Standardization (3d, $k$=5)* | | | | |
| KSG | $k$=5, std=True | 3 | joint | 45.03 |
| KSG | $k$=5, std=False | 3 | joint | 53.66 |
| *Per-Component vs. Joint KSG (std=False, $k$=5)* | | | | |
| KSG | $k$=5, std=False | 3 | joint | 53.66 |
| KSG | $k$=5, std=False | 3 | per-comp | 49.16 |
| KSG | $k$=5, std=False | 10 | joint | 54.97 |

| | | | | |
|---|---|---|---|---|
| KSG | $k$=5, std=False | 10 | per-comp | 49.91 |
| *Ensemble* | | | | |
| KSG 10d $k$=5 | std=False | 10 | joint | 54.97 |
| KSG 15d $k$=7 | std=False | 15 | joint | 57.41 |
| KSG 5d $k$=5 | std=False | 5 | joint | 54.03 |
| Ensemble (3-way vote) | — | — | — | 56.47 |

*Experiment 1 (baseline at 3 PCA components with default parameters) is subsumed by the ranges reported in Experiment 2 and the individual values in Experiment 9.

# Bibliography


[BWG+13] Dmitry Bogdanov, Nicolas Wack, Emilia Gómez, Sankalp Gulati, Perfecto Herrera, Oscar Mayor, Gerard Roma, Justin Salamon, José R Zapata, and Xavier Serra. ESSENTIA: An audio analysis library for music information retrieval. *Proceedings International Society for Music Information Retrieval Conference*, pages 493–498, 2013.

[KSG04] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical Review E*, 69(6):066138, 2004.

[Llo82] Stuart Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[LWM+09] Edith Law, Kris West, Michael I Mandel, Mert Bay, and J Stephen Downie. Evaluation of algorithms using games: The case of music tagging. In *Proceedings International Society for Music Information Retrieval Conference*, pages 387–392, 2009.

[LYZ+23] Yizhi Li, Ruibin Yuan, Ge Zhang, Yinghao Ma, Xingran Chen, Hanzhi Yin, Chenghua Lin, Anton Ragni, Emmanouil Benetos, Norbert Gyenge, et al. MERT: Acoustic music understanding model with large-scale self-supervised training. *arXiv preprint arXiv:2306.00107*, 2023.

[MRL95] Young-Il Moon, Balaji Rajagopalan, and Upmanu Lall. Estimation of mutual information using kernel density estimators. *Physical Review E*, 52(3):2318, 1995.

[OVK16] Olga Orlova, Yulia Vedeneeva, and Anton Korshunov. Music similarity measure based on Earth Mover's Distance. In *Proceedings International Conference on Analysis of Images, Social Networks and Texts*, pages 236–247, 2016.

[PVG+11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Ram18] Sebastián Ramírez. FastAPI: Modern, fast web framework for building APIs with Python, 2018. Available at `https://fastapi.tiangolo.com/`.

[RTG00] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The Earth Mover's Distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

[SB21] Janne Spijkervet and John Ashley Burgoyne. Contrastive learning of musical representations. In *Proceedings International Society for Music Information Retrieval Conference*, pages 673–681, 2021.

[Sil86] Bernard W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, New York, 1986.

[Ver16] Vercel. Next.js: The React framework for the web, 2016. Available at `https://nextjs.org/`.

[ZXT18] Xianli Zeng, Yingcun Xia, and Howell Tong. Jackknife approach to the estimation of mutual information. *Proceedings of the National Academy of Sciences*, 115(40):9956–9961, 2018.