UNIVERSITY OF BIRMINGHAM

Quantifying Musical Similarity through Audio Analysis

Oscar Lindenbaum

Student ID: 2005912

Date: 2022-2023

DA B.Sc. Computer Science with Digital Technology Partnership

Supervisor: Dr Ian Batten

Main Body Word Count: 9993

Abstract

Making comparisons between pieces of music is a complex task, made only more difficult due to the subjective nature of musical tastes. Yet, forming comparisons and understanding musical tastes is essential in designing comparative music suggestions and collaborative playlists. Many attempts have been made to compare songs for similarity using Neural Networks (NN) or Machine Learning (ML). Despite their benefits, explaining the musical reasoning behind the decisions is rarely possible and incredibly difficult using these methods. This dissertation uses a purely mathematical approach, allowing explanation of the reasoning behind the decisions made by the comparative algorithm. Comparisons were made from the following auditory features: beats per minute (BPM), key, and harmonics. Additionally, the most representative region of each song is selected and used to compare the timbral and rhythmic features.

To construct the fingerprint (analysis of the song), several techniques were employed to extract distinctive features from the audio signal. These techniques include the extraction of Mel Frequency Cepstrum Coefficients (MFCC), which are commonly used in speech recognition and music analysis to capture the spectral envelope of a signal. Additionally, the Root Mean Squared Energy (RMSE) is extracted to measure the energy of the signal to aid in selection a high entropy region of the song for subsequent analysis. Harmonic Pitch Classification Profiles (HPCP) are used to identify the most salient harmonic pitches present in the audio and to determine a key. Tonal Interval Vectors (TIV) are also extracted to capture the harmonic relationships between different signals in more detail than just the key. Other techniques that were used included the manipulation of Chromagrams, which represent the distribution of pitches and their energy over time, Tempograms, which measure the temporal structure of the rhythm and Spectrograms, which provide a visual representation of the frequency content of the signal over time, and finally BPM detection, which is used to determine the tempo of the song. By using these various techniques to extract different features from the audio signal, a comprehensive fingerprint is created that can be used to identify and compare different pieces of music.

The comparison algorithm will be used to create a cross-platform playlist which shows the similarity

results so that users can decide what songs in the playlist are more suitable to be played next. Comparison methods include the Small-Scale Compatibility (SSC) algorithm, the Harmonic Mean, and bitwise image comparison. The implementation will support multiple music streaming services as input. The goal is to provide a simple and intuitive tool that allows users to find music that matches the currently playing song well, as well as giving users the option to explore the musical theory behind the similarity score.

Contents

List of Figures

List of Tables

1 Introduction

Music is an essential part of the human experience. It spans continents, cultures, and individuals. Rarely does a human life go untouched by music. Music provides emotional regulation, entertainment and comfort, to name a few. However, following development of streaming services, many people have banks of millions of songs at their fingertips. With so much music available, people have increasingly varying tastes and know of artists unknown to their friends. Whilst not without its benefits, this has made creating playlists for road trips, parties or gatherings, a daunting task. As a solution, many music recommendation systems, commonly available within streaming services, have been developed. Often, these use Machine Learning (ML) based algorithms to analyse users' listening patterns and suggest music fitting their tastes (Wang and Wang, 2014). However, these systems are not designed to explain their reasoning. Furthermore, from a user perspective, most of these systems exist within one platform, restricting their usage to friends who subscribe to the same platform.

This paper proposes a new method for measuring musical similarity based on audio analysis, which does not require the use of ML. The proposed approach will be utilised in a playlist application that can run on multiple platforms. This application will display similarity results to users, allowing them to make informed decisions about which songs to play next. The application of this approach will be within a system allowing collaborative playlists to be created, supporting multiple music sources. Users connect via the web application and submit songs to the queue; songs are analysed and added to the queue for playback. Supporting the playback decisions will be the results from the analysis in the form of the comparison between the currently playing song and itself. It will be presented in simple colloquial language that touches on music theory.

The proposed implementation will leverage audio features such as timbre, rhythm, bpm, and harmonics to assess the compatibility between two songs. Additionally, it will support input from various music streaming services. To assess the effectiveness of the proposed method for determining song compatibility, user testing will be conducted. The primary objective is to create an intuitive and user-friendly tool that enables users to identify music which complements the track that is currently playing, while also potentially learning about the underlying music theory.

This solution has the potential to enhance users' listening experiences by providing a more coherent and seamless playlist, whether they are listening alone or with friends. Additionally, users may be exposed to new music, expanding their musical horizons, and further increasing their enjoyment.

Specific areas in this paper that are original contributions are:

- The idea of a mathematical approach to compare songs, rather than ML.
- The comparison methods and strategy to compare songs, specifically the spectrogram comparison.
- The region of interest selection for the fingerprint.
- The adaptation of TIV to determine a song key.
- The Root Harmonic Distance measure.

An instance of where this platform could be used of is at a birthday celebration. Suppose the host has left a computer accessible to guests to add songs to play throughout the night. The atmosphere is generally cheerful and lively, the song "Walking on sunshine" (Katrina and the Waves, 2009) is currently playing. At this point, a guest adds "Hurt" (Cash, 2002) to the playlist, which is inconsistent with the mood set by "Walking on sunshine". Such an addition could potentially disrupt the flow of the playlist and alter the overall ambiance of the celebration, leading to a less enjoyable experience for the guests. However, the utilization of this application can mitigate such an issue by providing the guests with the option to downvote the songs they dislike. This feature enables the host to maintain a pleasant ambiance throughout the party, allowing the atmosphere to develop naturally. Crucially, guests who may not be familiar with the songs queued can use the similarity score to determine how well it fits with the current song being played, making informed decisions on whether to vote it up or down.

1.1 Project Aims

The aim of this project is to create an application for creating collaborative playlists. These playlists will include similarity ratings between songs to indicate to the user which songs sound better playing after each other. Users can decide the order of the playlist by upvoting the songs based on their personal preferences and the system's similarity scores. The system will attempt to provide simple informative description on these similarity scores.

The objective of this project is to develop a collaborative playlist platform that enables users to add songs from various streaming services and allow them to vote on their preferred tracks. The focus is to provide users with a straightforward and informative way of choosing the next song to play. To achieve this, the project will research and develop a system that compares and quantifies the musical similarity of two tracks based on extracted features. A typical solution to this problem is to apply ML Models or Neural Networks (NN) to the extracted features. However, this is trivial and would achieve a similar result but wouldn't be able to explain the comparison decisions to the user. The success of the project will be evaluated based on the user's satisfaction with the system's ability to provide relevant recommendations for the next song to be played in collaborative playlists.

Another objective is to ensure that the system's similarity ratings are both accurate and cover a range of attributes. This means that the system should not only identify songs that are musically similar but also consider a range of factors, such as the mood, tempo, and energy of the song, when generating scores. By considering a diverse range of attributes, they can help users discover new music from others in the session, that they may not have otherwise encountered.

1.2 Challenges

Quantifying musical compatibility, or the degree to which two pieces of music are compatible or complementary, is a challenging task. One of the most significant hurdles is the subjective nature of musical taste. Musical preference and opinions hugely vary between individuals. Songs which to one person sound compatible, may not be to another. Additionally, music can evoke emotional responses that are difficult to quantify objectively.

In the application, ML/NN models will not be used. The reasoning behind this is so that human readable explanations can be transferred to the user (Adadi and Berrada, 2018). Once we have an audio file, we want to be able to compare the similarities to another, using pure intelligence[1] only. This means we can explain the decisions made by the comparison algorithms. The platform allows sessions to be created and joined from any device, allowing users to import songs from multiple different sources such as Spotify, YouTube, Soundcloud and Deezer. Without the use of ML/NN models, the difficulty of comparing tracks increases significantly.

There will be challenges across the whole project, critically, the speed of the DSP analysis and the API streaming support from audio platforms will be important. Decisions will need to be made to achieve a compromise between speed and accuracy.

[1] Pure intelligence is the ability to reason and make decisions without the use of ML/NN models.

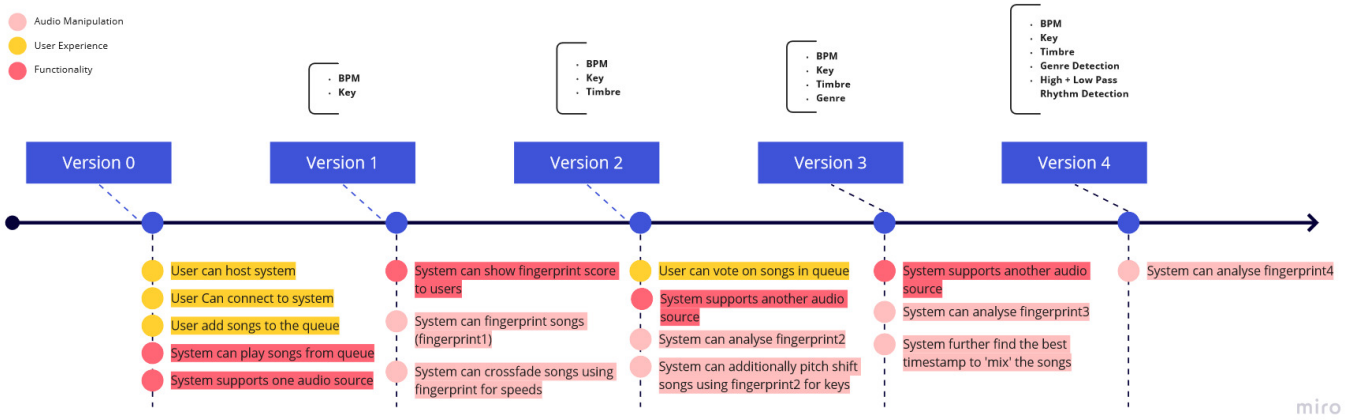


Figure 1: Initial Roadmap

An issue with some audio platforms, which may require further research, are the limits on track streaming. Due to licensing some platforms will only let you stream 30 seconds at a time (The Spotify Development Team, 2023), this means all analysis must be based on the streamable chunk, the application would have to buffer the whole song (platform dependant) or not utilize major streaming services. Figure 1 shows what the platform is expected to handle at varying stages of development.

1.3 Related Work

This paper attempts to compare and quantify similarities between two songs. Before we can compare songs, we need to define what music is. Müller et al. (2011) outline how "Musical creations (…) are among the most complex and intricate of our cultural artifacts, and the emotional power of music can touch us in surprising and profound ways".

1.3.1 Extracting Musical Content

Müller et al. (2011) describe several aspects of musical signals that could be used to describe the overall signal, "pitch, harmony, note onsets, beat, tempo, and rhythm, and timbre and instrumentation" are listed as methods to quantize them. These properties describe an audio signal well, touching on both structural and harmonic areas. The paper also discusses 'duration' as a descriptive property, while this does describe the structure of the signal, its information is covered within the other features.

A subset of the attributes described in the paper can be separated into two groups; those related to structure, and those related to the content of the sound. The attributes related to structure include: beat, tempo, phrasing, and rhythm. The attributes that are related to content of sound include: pitch, harmony, timbre, and instrumentation. Several other factors could include the context of the sound, such as genre, lyrics, and mood.

Both the content and the structure of a musical piece are important when considering a signal. Jacobson (2014) describes musical structure as "the element of 'time', the "structural rhythmic pulse of the music." The content of the sound, is more complex as it describes everything else that does not relate to the feeling of time. Structure is the important when describing a signal, but the most important aspect will be describing the content.

Given the complexities to describe music itself, attempting to compare two songs grows further in complexity.

The field of extracting information from audio for further analysis is called Music Information Retrieval (MIR). Karydis et al. (2013) detail an approach to compare audio based on its content. They carefully select features some of which are spectrally related (Energy, Beats, Flux) or other MIR methods like Linear Predictive Coding (LPC) and Mel-Frequency Cepstrum Coefficients (MFCC). Similarity is calculated using Artificial Neural Networks (ANN's). In this paper we will be using a combination of both a Spectral and a MFCC based approach. ANN's will not be used as they prevent us providing musical reasoning behind the decisions.

Gurjar and Moon (2018) apply several approaches to comparing features. One time-efficient method is a geometric approach. A geometric or visual approach could be applied to spectrograms to compare similarity. When treated like an image, a spectrogram contains information about location, intensity, and frequency. The paper also lists Markov Chains as an alternative approach. Features like a pitch, duration, or interval could correspond to states in the chains. Markov chains are good at modelling sequences of events and would be the preferred choice but are time consuming to implement well. A visual approach would be a better, as it is intuitive and easy to understand. Instead, an exclusive OR (XOR) comparison per pixel requires only a few mathematical operations, making it less resource-intensive than Markov Chains. Additionally, XOR comparison does not require the use of complex algorithms or techniques, making it simple and easy to implement.

The visual representation provided by the XOR approach is intuitive and easy to understand. This visual representation allows for easy identification of areas of similarity and dissimilarity between the two inputs.

Astley et al. (2023) use a range of evaluation metrics to evaluate the performance of lung segmentation using Convolutional Neural Network's (CNN). These include the overlap-based Dice similarity coefficient, the distance-based Average Hausdorff distance, and the error-based XOR metric. The XOR method is fast and can give a quick indication on similarity/error for images. When applied in the context of a spectrogram, XOR would be able to pick out areas of difference not visible to the human eye in both intensity and subtle variations in pixel position.

Bello et al. (2005) explore several different ways of detecting note onset, which can then be used to deduce the BPM. The paper indicates that the 'negative-log likelihood' method achieves the fewest false positives and is not computationally intensive. It details how a spectral approach would be more suited to complex inputs (more precisely either Phase Deviation or Spectral Difference). Approaches that contain pre-processing and thresholding prior to picking peaks outperform others.

The BeetRoot system (Dixon, 2001), is a robust method to estimate the tempo and identify the beats within a musical piece. Oliveira et al. (2010) aim to extend this to allow for better estimations in noisy tracks and to improve the parallel processing capabilities so it can be applied to continuous sounds. This extension to BeetRoot has a good reputation within the IEEE and has a C++ implementation within the Marsyas framework (Tzanetakis, 1994). Oliveira et al. found good results using a Spectral Flux based approach.

The Krumhansl-Schumuckler Key-finding algorithm, uses the idea of profiles to determine a songs key. A profile being a 12-element vector of representing the stability of the pitch. The correlation between the defined profiles and the songs profile is then used to determine which key fits best. Temperley (1999) outlines how "modelling a task performed by humans" is complex as it is usually judged "by music theorists and other highly trained musicians", and also subject to human preference. Temperley argues that the key-profile model is a successful solution to the key-finding problem, and that several other factors, such as spelling, harmony, and the primacy factor, are needed rather rarely. Although a key-profile model can perform key finding well without harmony, it does not mean that a harmony-based model cannot perform the task well without key-profile information. Although a harmony-based model is not required, harmonic information is still useful in the comparison process and would aid the key-prediction.

The definition of most musical genres is subjective, and there is no broad agreement concerning the name of the genres. Barbedo and Lopes (2006) describe a strategy to detect genre within a musical signal. The strategy presented divides the audio signals into frames and extracts 4 features: bandwidth, spectral roll-off, spectral flux, and loudness. These features are used to calculate three summary features: mean, variance, and prevalence of the main peak. Initially, this paper included genre as a factor to describe the signal, while genre itself will not be used, the features used to

describe it will be used in the content analysis.

Many approaches have been developed to solve the problem of musical structure detection. The most used features are often based on timbre, pitch and harmony, rhythm, or a set of multiple descriptors. Wankhammer et al. (2010) proposed an approach for MIR regarding structure. Mel Frequency Cepstrum Coefficients (MFCC) are used to represent vocal and timbral qualities and Chroma Vectors are used to incorporate pitch and harmonic content to the extracted structure. Chroma is a successful basic indicator of the harmonic and melodic progressions of music. It is calculated by summing the energy over all the bins[2] belonging to one tone.

Ruptures is a Python library for change point detection, created by Truong et al. (2020). It can be used to separate an audio signal into sections, or musical phrases. Segmentation happens based on abrupt changes to sections of signal which are measured using a variety of search and cost functions. Signal segmentation is important to identify phrasing/structure within a musical piece. In our implementation, focus is placed on the segment of the signal, which has the best representation of the whole track, demonstrating the importance of selection a region which does not span across phrases.

1.3.2 Subjectivity of Music

For many, music is an important element of everyday life and serves multiple functions, including entertainment, comfort, and affirming group belonging (Boer, 2009). Music also serves important social functions, such as fostering social relatedness and being richly linked with other social activities such as road trips or gatherings.

The importance and personal nature of music means musical tastes, by nature, are subjective. Furthermore, people are generally opinionated. Therefore, creating an artificial opinion on similarity is challenging given this may vary between individual tastes. Rentfrow et al. (2011) attempt to understand the reasoning for different musical tastes. They attempt to distinguish using descriptive words like “Mellow”, “Urban” and “Sophisticated”, but finds other factors at play like the users

[2]A bin is a container for each musical note, that contains all of its octaves[3].

[3]An octave refers to the interval between two pitches where the higher pitch has twice the frequency of the lower pitch. For example, the pitch of A4 (440 Hz) is one octave higher than A3 (220 Hz).

emotional state or personal use for music.

Karydis et al. (2013) state that measuring musical similarity "is a hard problem" and attempts to compare pieces using the content of the music. As humans, we have a powerful ability to use contextually semantic information to identifying similarities and differences between sounds (Lotto and Holt, 2010). This is a difficult task for computers, but the system should be able to extract as much semantic information as possible to form the comparisons.

Park and Kaneshiro (2021) explore the idea of collaborative playlists and how they socially affect groups. Successful collaborative playlists can expand musical tastes. This can be attributed to the diversity of tastes from different collaborators, and the privileged status of social music recommendations. "Millions of listeners worldwide consume music on streaming platforms such as Spotify (…) and platforms have placed great import on music recommendation systems that suggest content tailored to individual users' tastes." Therefore, creating a good comparison algorithm to be implemented in the system would be beneficial to the user.

1.4 Other developments

Currently on the market there are 3 comparable products.

- Festify - A shared Spotify streaming app, which allows users to choose the next song.
- Jukebox - A YouTube based streaming app which allows users to listen to the same songs and chose the next song.
- JQBX - A Spotify based platform which allows users to take it in turn to 'DJ'.

All products restrict themselves to one platform e.g., Spotify or YouTube. The system will address this challenge by facilitating usage across multiple streaming platforms. The closest product to this projects goal is JQBX, which gives the users the most control over what is playing, but this has a complicated user interface. The implementation and user experience of Festify is better due to its simplicity. However, none of these products offer information to advise users of which song to play next based on the current song and programmatic blending. Furthermore, they do not provide users insight into some of the musical theory behind the songs.

2 Background

2.1 Spectrograms

A spectrogram displays changes in an audio signals frequencies over time, with the amplitude represented by the intensity of colour. To extract a spectrogram, we start with the raw waveform, an example waveform is shown in Figure 2.

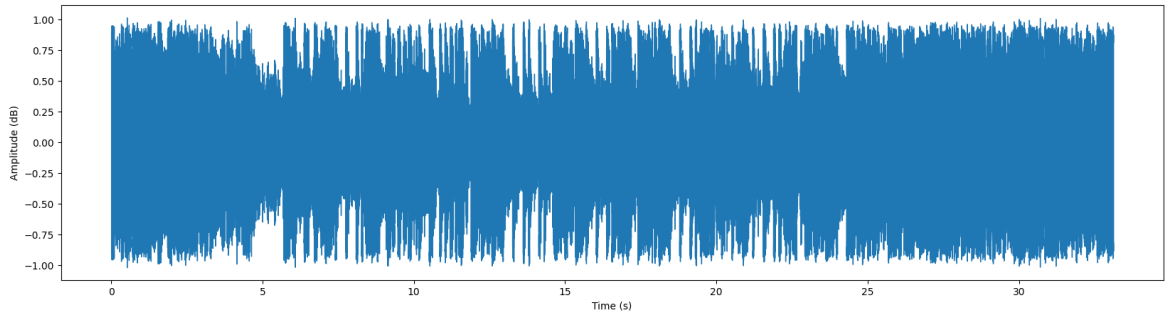


Figure 2: A waveform of 'As it Was' (Styles, 2022)

The first step is to take the waveform and apply the Fast Fourier Transform. The Fourier Transform converts the waveform from amplitude values to frequencies (Brigham and Morrow, 1967).

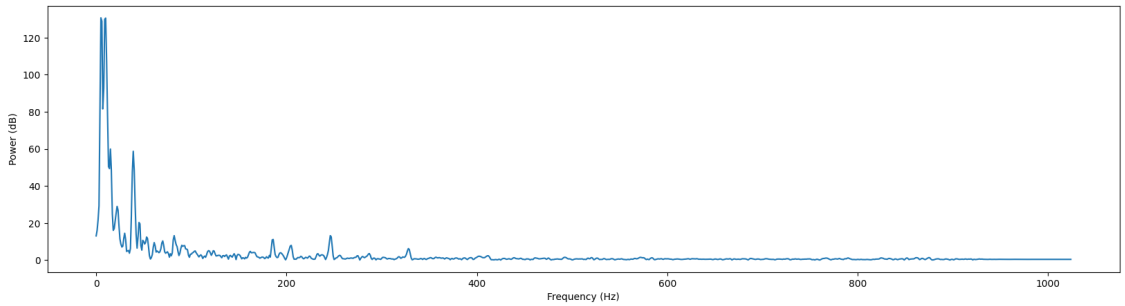


Figure 3: Short Time Fourier Transform of 'As it Was', for one hop (Styles, 2022).

The result of applying the Fourier Transform to one hop[4] of the above waveform is shown in Figure 3. The graph shows the power of each frequency for the hop, or the amount of each frequency that is present in the signal. To convert this to a spectrogram, we plot the amplitude along each transform

[4]A hop is usually 512 samples. Samples are the number of data points per second. Typically $\sim$ 21,000.

for the whole track. Additionally, we will convert the y-axis to the logarithmic scale as frequency is exponential.

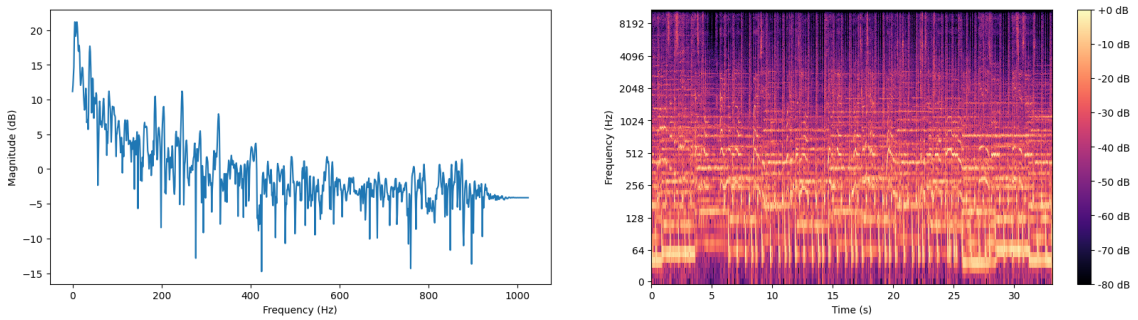


Figure 4: Log-Power Spectrogram of 'As it Was' (Styles, 2022)

Looking at the spectrogram, we can begin to understand the structure and harmonics of the song. For example, a spectrogram can reveal the different instruments being played in a particular section of the song, as well as the different harmonics produced by those instruments. When multiple instruments or voices are playing at once, the spectrogram will show a complex interplay of different frequencies and harmonics. These are represented in the horizontal bars, showing multiple harmonizing frequencies, with varying shapes (Figure 4).

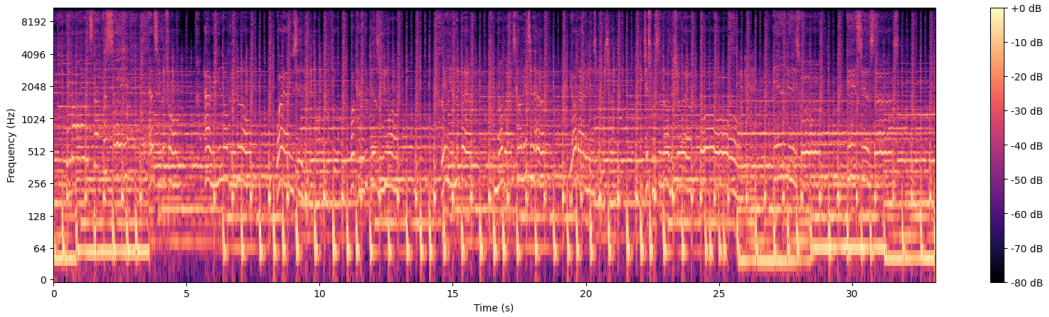


Figure 5: Mel-Spectrogram of 'As it Was' (Styles, 2022)

Another type of spectrogram is the Mel-Spectrogram (Figure 5). This is very similar to the standard spectrogram, but its scale is shifted from Hertz to the Mel scale which is a perceptually-based scale that more closely approximates human hearing (Pedersen, 1965).

A Mel-Spectrogram can provide additional information about the structure and harmonics of a song (Ullrich et al., 2014). It can also demonstrate how the overtones or harmonics of a particular instrument change over time, revealing subtle variations in timbre that may be difficult to perceive with a standard spectrogram. It can also help to distinguish between several types of instruments that produce similar frequencies but have distinct timbral qualities, such as a piano versus a guitar.

2.2 Mel-Frequency Cepstrum Coefficients

Mel-Frequency Cepstrum Coefficients (MFCCs) are a representation of the short-term power spectrum of a sound. They are derived from a type of cepstrum of a signal. The cepstrum[5] of a signal is the inverse Fourier transform of the log of the Fourier Transform of a signal. The MFCCs that are going to be used are the first 39 cepstral coefficients, the first 12 are related to the amplitude of frequencies and the rest help describe the signal. The MFCCs are derived from the following steps (Gupta et al., 2013):

1. Take the Fourier Transform of (a windowed excerpt of) a signal.
2. Map the powers of the spectrum obtained above onto the Mel scale, using triangular overlapping windows.
3. Take the logs of the powers at each of the Mel frequencies.
4. Take the Discrete Cosine Transform of the list of Mel log powers, as if it were a signal.
5. The MFCCs are the amplitudes of the resulting spectrum.

The first three steps are used to produce a Mel-frequency spectrogram (Figure 5), which is a representation of the short-term power spectrum of a sound. The last two steps are used to produce the cepstral coefficients, the coefficients that help to describe the characteristics of the sign. The resulting cepstral coefficients are shown in Figure 6.

[5]The Cepstrum, is the Inverse Fourier Transform of the Spectrum.

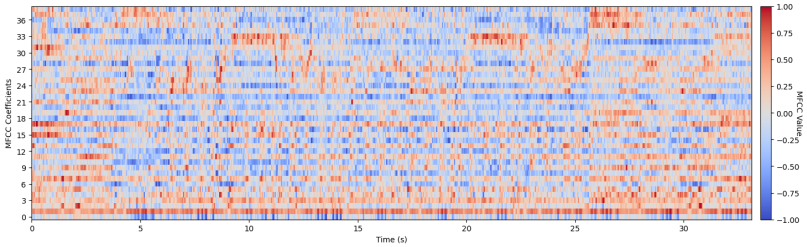


Figure 6: ‘As it Was’ MFCC’s (Styles, 2022)

A MFCC graph shows us both structure and harmonic information about the signal. Interpreting the MFCC is difficult due to the fine detail it can show, and the loose description of timbre. The structures appearing from the coloured bars can indicate certain musical phrases or melodies Additionally, visually repeating patterns of bars could indicate a repeating melody/harmony. Whilst MFCCs were developed to focus on sounds made by the human vocal tract, specifically the glottal pulse, they manage to describe timbral features well.

2.3 Tempogram

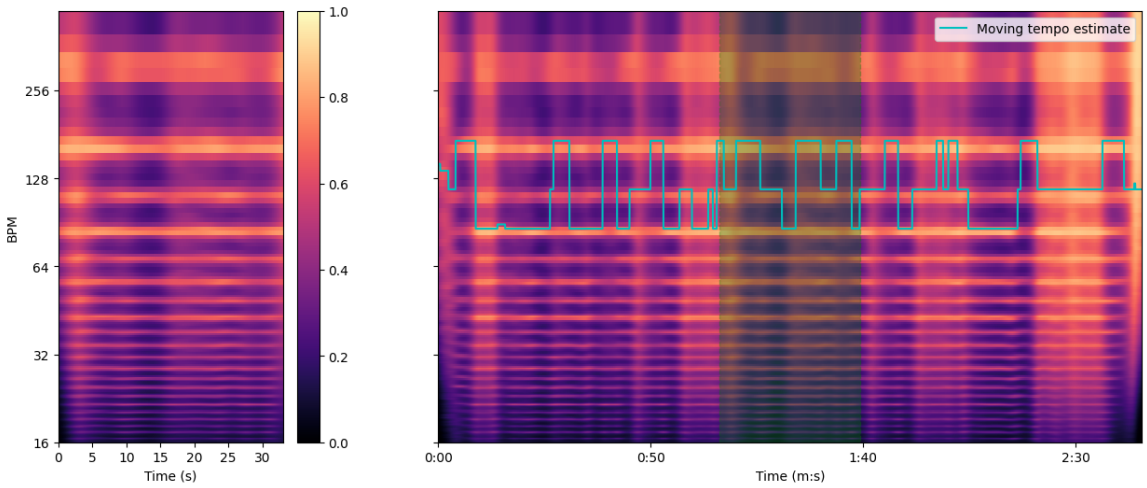


Figure 7: Tempogram of selected region of the song 'As it Was', and the full Tempogram (Styles, 2022)

Unlike a traditional spectrogram, which displays the frequency content of a sound over time, a tempogram displays the rhythmic content of a sound over time. The horizontal axis represents time, while the vertical axis represents tempo, measured in BPM. To better understand how to interpret the tempogram shown (Figure 7), rather than considering BPM as the speed of song consider it as the derivative of speed per hop. Furthermore, also plotted is the movement of the tempo as shown by the blue line, this allows us to see movements and phrases more easily in the selected section of music.

To create a tempogram, the audio signal is first analysed to extract the rhythmic content of the music. This is typically done using an algorithm such as onset detection, which identifies the points in time where new musical events or notes begin. Once the rhythmic content has been extracted, the tempo of the music can be estimated at each point in time using a technique such as autocorrelation or phase analysis (Bello et al., 2005). The resulting tempo estimates are then plotted on the vertical axis of the tempogram. The resulting tempogram provides a visual representation of the tempo or rhythmic structure of the music, this is useful for analysing the overall structure and dynamics of the music, as well as for identifying specific rhythmic patterns or sections of the song (Figure 7).

2.4 Chromagrams

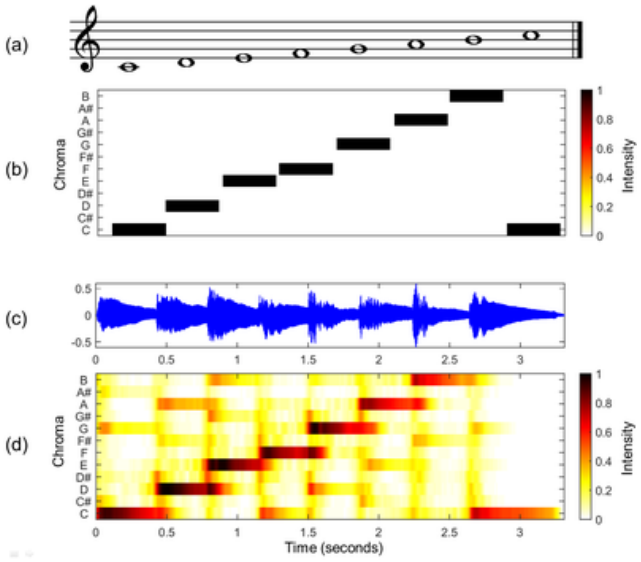


Figure 8: Chromagrams Explained (Wayne, 2021)

Chromagrams show how the different pitches or musical notes are distributed through a song.

To create a chromagram, the audio signal is first analysed to extract the pitch content of the music. This is typically done using the Fourier Transform or a variant such as the Constant-Q transform. This breaks down the sound into its fundamental frequencies, the whole audio signal gets mapped onto 12 musical notes. It shows the magnitudes of each note and their harmonics which will also be present, allowing us to picture the notes used within the song.

Once the pitch content has been extracted, the chromagram is created by plotting the intensity of each pitch class over time. A pitch class is a set of pitches that are separated by octaves, such as all the C notes on a piano keyboard. The intensity of each pitch class is typically calculated as the sum of the magnitudes of the frequencies that correspond to that pitch class.

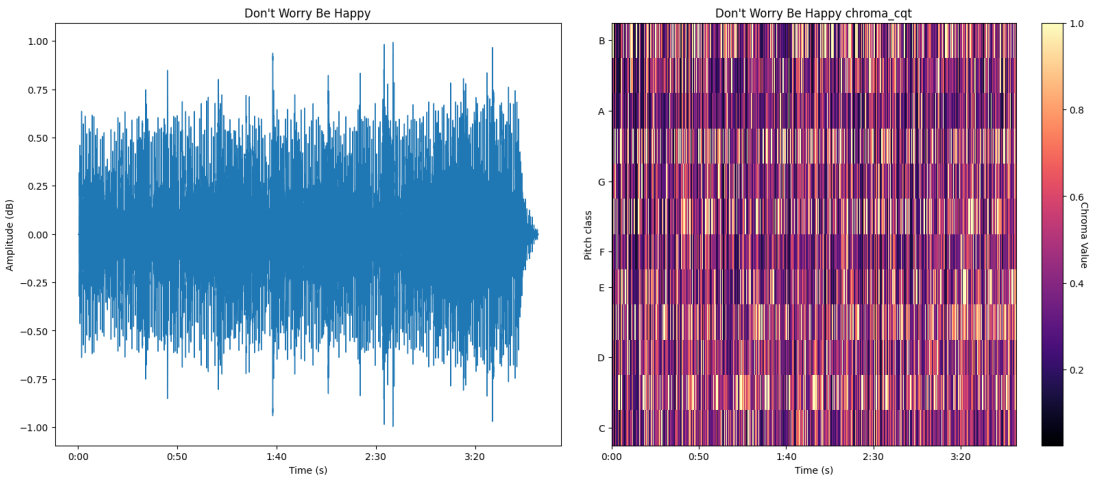


Figure 9: Chromagram of (McFerrin, 2009)

The resulting chromagram provides a visual representation of the pitch content of the music, allowing us to identify how the different pitches are distributed over time. This can be useful for analysing the melody and harmony of the music, as well as for identifying specific chord progressions or sections of the song.

Shown is an example chromagram, the chromagram shows a complex structure consisting mainly of $C\sharp$, *E and* $G\sharp$ (Figure 9).

2.5 Pitch Classification Profiles

A single pitch classification profile (PCP) is a 12-dimensional vector that represents the pitch content of a music segment (Fujishima, 1999). Each element indicates the presence or absence of the corresponding pitch class. PCPs can be used for a variety of music analysis tasks, including genre classification, melody extraction, chord recognition, and music similarity assessment.

A chromagram is a two-dimensional representation of the pitch content of a musical signal, where the rows represent the 12 pitch classes, and the columns represent time frames. Each element in the chromagram represents the energy or intensity of the corresponding pitch class in the corresponding time frame. PCPs are a widely used and effective representation of the pitch content of music, we will use them to define the key and harmonics of our songs.

Additional Areas

2.5.1 Change Point Detection

Truong et al. (2020) explain change point detection as "a model selection problem, which consists in choosing the best possible segmentation τ according to a quantitative criterion $V(\tau, y)$ that must be minimized."

Ruptures is a Python package that segments a signal into n segments based on a search criteria. In our application we will be using the Multilinear model C_{AR} (Truong et al., 2020, Figure C7) as the cost function. This is the most suited as it responded best in our tests to segment the sections of songs. It is also best suited to electrocardiogram (ECG) and speech recognition tasks. Furthermore, the search method chosen is the Sliding Window. Once again this is most suited to speech tasks but additionally, the Sliding Window technique is applied heavily across other areas of musical analysis (Truong et al., 2020, Figure 8).

Ruptures will be used to separate a song into distinct parts such as the intro, bridge, and outro. It will be used to select a region of the song that describes the whole song with the highest entropy.

2.5.2 Tonal Interval Vector Library Explanation

TIVLib is a Python library "A Python library for the content-based tonal description of musical audio signals" (Framires, 2019). The Tonal Interval Vector (TIV) is a mathematical representation of the tonal content of a musical piece. It is based on the idea that the tonal structure of a piece of music can be characterized by the distribution of the intervals between the pitches played.

It is a vector that represents the frequencies of occurrence of each possible interval class in the music. Interval classes are defined as the distance between two pitches in terms of the number of diatonic steps (e.g., major second, minor third, perfect fourth, etc.).

For example, the TIV for a piece of music might be `[0.1, 0.05, 0.15, 0.2, 0.25, 0.1, 0.15]`, which indicates that 10% of the intervals in the music are minor seconds, 5% major seconds, 15% minor thirds, 20% major thirds, 25% perfect fourths, 10% are tritones, and 15% perfect fifths.

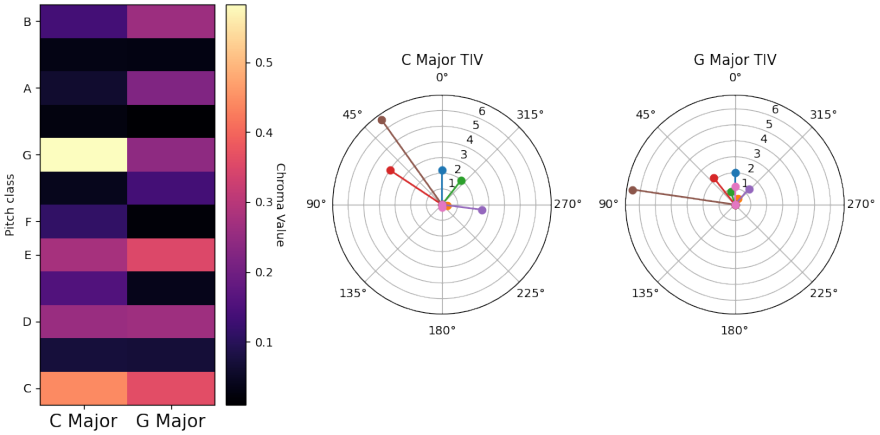


Figure 10: Chromagrams of Two Chords and their Tonal Interval Vectors

The TIV can be used to characterize the tonal structure of a musical piece and to compare different pieces of music. Figure 10 shows two chords, C Major, and G Major. The first graph is a Harmonic PCP (HPCP) of the chord, and following is the corresponding TIV, plotted in the polar space. Similarities can be seen between the two with the main difference being the brown vector. The brown vector indicates the *Diatonicity* of the chord. The other attributes we can extract using TIVLib are Diatonicity, Chromaticity, Dyadicity, Wholetoneness, Triadicity and Diminishment.

3 Implementation

3.1 Obtaining Audio

The raw waveform is required for conducting any analysis on a given song. However, acquiring the raw file can be a non-trivial task. The following section discusses some of the tools employed to acquire the raw file.

3.1.1 YouTube API

YouTube provides an API for us to search its videos where we can query for a musical piece (Google Developers, 2023). Further filtering can be applied to limit the query to only music videos and shorter tracks (to avoid 30-minute videos being processed by our application). A major drawback of the API is its credit cost. Per day you can use 10,000 credits, each search requires 100 credits. To minimize the number used; we will not be using a live search where results update as you type.

3.1.2 YouTube-DL

Youtube-dl (youtube-dl Development Team, 2006) is a program to download videos from YouTube, that also allows you to specify the quality required. We can use this to download the raw video in the best audio quality available. To then extract the audio track from the video file, we will need to use FFmpeg.

3.1.3 FFMPEG

"FFmpeg is the leading multimedia framework, able to decode, encode, transcode, mux, demux, stream, filter and play pretty much anything that humans and machines have created" (FFmpeg developers, 2000). Using the tool, we can take our MP4 file and separate the audio channel and save it as a 128-bit waveform. FFMPEG is highly respected by the community and is included in base Ubuntu installations.

3.1.4 YT-DLP

Our analysis is hidden behind an API endpoint, where users will submit requests and have to wait for them to complete before they can see the results of their request. The analysis will take a few minutes to finish, part of which includes downloading the audio file.

To minimize the time spent downloading the file, a fork of *youtube-dl* has been written with several improvements and is more actively maintained. One of which is to utilize the extra bandwidth allocated by YouTube on the initial request of a video, normally this is used to show previews of a video, yt-dlp repeatedly stop-starts the connection to achieve a higher bandwidth to achieve a faster download. Another improvement is the inclusion of FFMPEG, normally we would have to spawn another process to separate the audio channel, ffmpeg is inbuilt into yt-dlp so one less process is needed to be spawned (yt-dlp Development Team, 2020).

3.1.5 Spotify API

Spotify allows us to interact with their songs via an API, this is well documented on its usage. We can search for tracks and play them on a device. Unfortunately, we can only get 30 seconds of each track a time due to copyright limitations (The Spotify Development Team, 2023). We will not be able to access the raw files unless we can find them elsewhere. To keep support for Spotify, the users can still search and submit songs via search, but we will have to convert them to YouTube.

We first query Spotify and get the user to select the exact track, then we use the YouTube API to query the song name and artist. We will then take the first result and proceed to process it as a YouTube track. In order to support native playback via the Spotify SDK, the Spotify URI will be saved along with the fingerprint to the database.

3.2 Architecture

During development, consideration was taken as to which database to use. Both the client and host device would need to listen to a queue for changes and be accessible over a simple SDK/API. Options included: (i) Firebase (ii) Supabase (iii) PocketBase

Firebase initially seemed the best choice as it is very well documented and is commonly used, but was not used due to tier limitations. The application would need to either constantly poll or listen to the collections[6]. With each record in a collection costing credits, the limit would easily be met. Similar negatives occur with Supabase but more with their Channel Model being complex.

```
// PocketBase
pocketbase.collection('chats').subscribe('*', async (e) => {
    allChats = await getChatWithUsers();
});

// Supabase
chatsWatcher = supabase.channel('custom-all-channel').on(
    'postgres_changes', { event: '*', schema: 'public', table: 'chats' },
    async () => {
        ({ data: allChats } = await getAllChats());
    }
).subscribe()
```

Figure 11: Code snippet comparing listeners for Supabase and PocketBase, from programonaut.

PocketBase was chosen for the simplicity of its codebase (Figure 11), and as another benefit, being self-hostable. This allows it to be deployed within a Docker container, which is running the API server in a separate image. The design of the system is shown in Figure 12. As PocketBase can be self-hosted, we can run it within the same docker image that is also running our server code, this makes the system portable and restricts access to the database.

The basic uses of each component are as follows,

- GoFiber - The server code, which will be used to manage users, sessions, compare fingerprints, and interact with the frontend.
- Librosa - The audio framework in Python that will handle the graphing and fingerprint generation.
- PocketBase - The database for storing data.
- ReactJS - Frontend for both the user and host device.

[6] A collection is a grouping of related documents stored in a database.

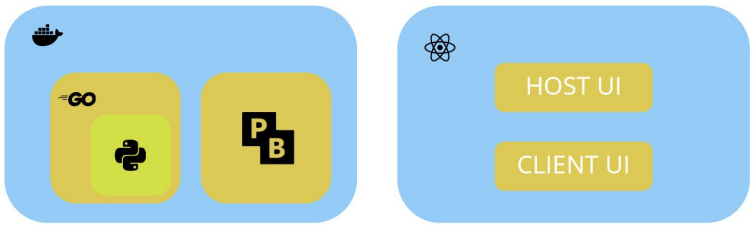
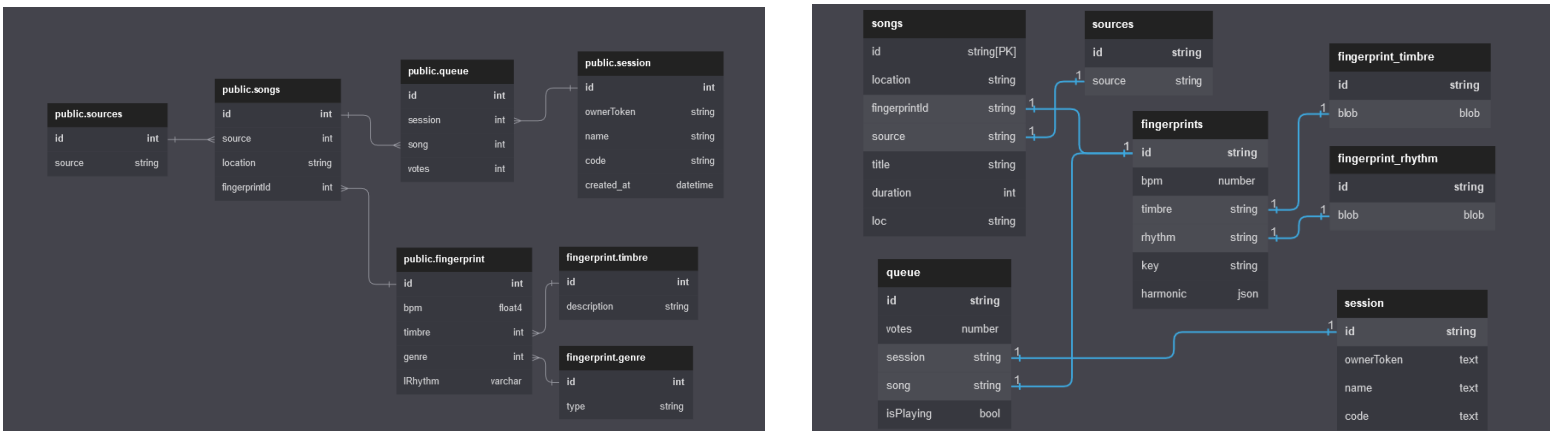


Figure 12: Platform Architecture

The database schema is detailed in Figure 13b, it is designed to comply with Third Normal Form (3NF). The benefits of 3NF allow for better efficiency and to ensure no data duplication. This schema allows for the reuse of fingerprints, this will be beneficial for future implementations where we can avoid re-fingerprinting songs that have already been analysed. The change in database structure is also shown in Figure 13. The initial schema was designed to be simple and easy to understand, however, it was not the most efficient. The new schema is more complex but allows for better performance.



(a) Initial Schema

(b) Actual Schema

Figure 13: Database Structure (See Appendix)

3.3 API Server

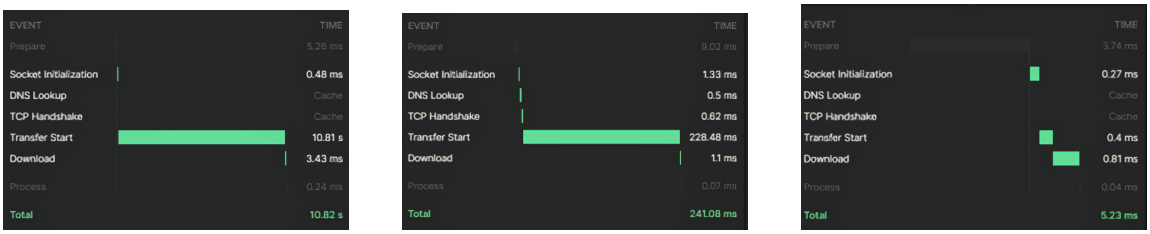
NodeJS vs GoLang

Initially, the backend was written in NodeJS with the express framework handling the requests. The server would spawn a subprocess to analyse a song once a request had been received. Analysis consists of downloading the audio file of the song, calculating values for each attribute, and publishing the results to PocketBase. While it was not the quickest, NodeJS performed well initially.

An issue appeared when multiple fingerprint requests were sent at once. Although each request was spawned in its own thread, the parent thread still got blocked until the child processes had terminated. This happened because NodeJS is inherently single threaded. Due to this drawback, the backend was rewritten in a low-level language, GoLang (Griesemer et al., 2022), with speeds comparable to C.

GoFiber, created by Çanbay et al. (2023) is a GoLang package similar to ExpressJS. Rewriting the backend in GoFiber improved caching, speed, and computation time. To demonstrate the improvements, Figure 14 shows an increase of 45% without caching and 2069% with caching.

GoLang, also allowed proper parallelization with further improvements such as Prefork. Prefork allocates subprocesses to run the server and they themselves use GoRoutine's to spawn processes. This is similar to load balancing.



(a) (b) (c)

Figure 14: Speed improvements of Go over NodeJS

3.3.1 Authentication

A user can connect to any session as long as they know the session code. Once connected they can view the queue and add songs to the session. To correctly authenticate the user and attach information to the session, JWT tokens are used. Jones et al. (2015) created the idea of JSON Web Tokens, a way to encode claims or information in a JSON object that is then signed by the server.

On a correct session code, the server creates a JWT token which contains; *UUID, Session Code and an expiry timestamp*. This is sent back to the user to be stored. Every subsequent request must contain this token as a Bearer in the request header.

This authentication layer is enough to validate the user is who they say they are, and that they have joined via the correct endpoint. It also attaches their requests to a certain session. A further authentication feature which was not added would have been to password protect the sessions. This would allow private sessions to be hosted but was out of scope.

3.3.2 HTTP/2

HTTP/2 (RFC9113 - Thomson and Benfield (2022)) is an improvement on HTTP/1 (RFC9112 - Fielding et al. (2022) in many ways. One of which allows the server to send multiple responses to client without the need for WebSocket's (RFC6455 - Fette and Melnikov (2011)). HTTP/2 introduced the idea of *Server Push*, where the server can send additional resources in a single client requests, improving performance. Server Push could be used to send notifications to the client regarding the status of their song submitted to the queue/fingerprinted. Unfortunately, Server Push has been removed in some browsers due to lack of support and usage (Google Chrome Developers, 2021). Google recommends using an alternative called EarlyHints (RFC9112 - Oku (2017)).

EarlyHints, is another way that could be used to indicate to the client that a point in their request has been reached. For example, send `[103]: Downloaded` when the song has finished downloading, `[103]: Fingerprinted` once analysis has been completed and finally `[200]: OK` when the analysis has been committed to the database. EarlyHints would allow finer status updates to be sent to the client without the need for opening a WebSocket which is resource consuming.

GoFiber uses `FASTHTTP` (fasthttp developers, 2016), a rapid framework for its underlying networking. `FASTHTTP` has support for HTTP/2 but unfortunately GoFiber does not support the feature yet. Had more time been available, HTTP/2 would be attempted to be implemented to take advantage of EarlyHints. The application currently sends a `[200]: OK` at the end of analysis.

3.4 The Fingerprint

For each song, we want to construct information that can be easily compared to each other per attribute we are interested in. Figure 15 shows a plan on how to create the fingerprint.

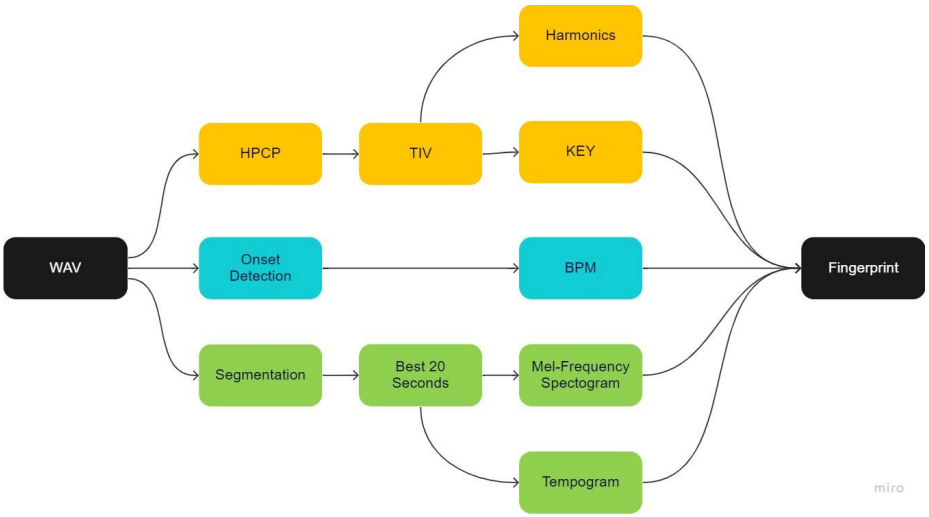


Figure 15: Fingerprint Plan

Starting from the most basic format, a waveform (WAV) file, we can extract information about the Harmonics, Key, BPM, Timbre, and Rhythm.

Each attribute explains a different aspect of the track. These have been chosen because it is enough to describe the majority of the track but not too many to overfit out comparisons.

- **BPM** is the speed of the song.
- **Key** is the set of notes used in a song that give it a specific sound.
- **Harmonic** is the musical emotion/feeling.
- **Timbre** is the unique quality or tone colour.
- **Rhythm** is the pattern and structure of the beats.

The attributes can be analysed in groups.

- Harmonics and Key.
- BPM.
- Timbre and Rhythm.

The reasoning for these groups is the relating information and the nature of the methods to be applied. We can identify more information about the harmonics once we have the key. For the Timbre and Rhythm, we will want to end up with an image that we can compare later. Timbre and Rhythm are harder to quantify so an image contains the all the original information, but in an easier format to manipulate.

With all comparisons we are going to start with a waveform.

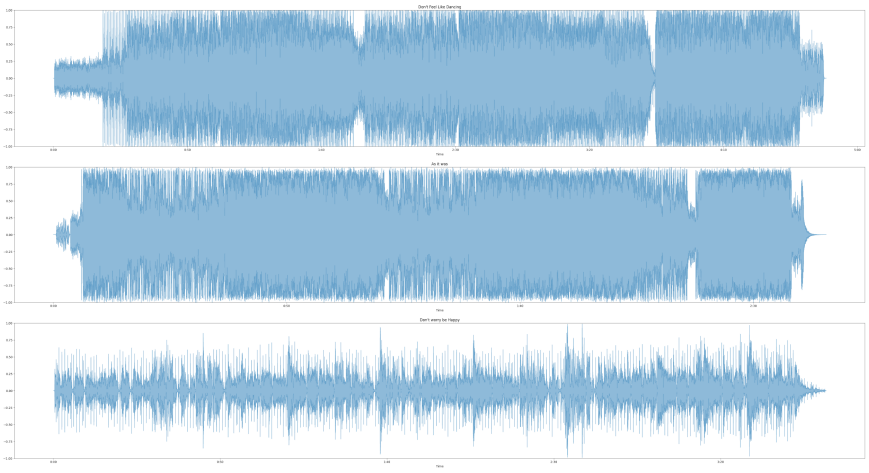


Figure 16: Waveforms of ‘Don’t feel like dancing’ (Sisters, 2006), ‘As it Was’ (Styles, 2022) and ‘Don’t Worry Be Happy’ (McFerrin, 2009)

3.4.1 Considerations

The concept of rhythm matching was explored, and potential algorithms were researched, but the idea of adapting a medical electrocardiogram (ECG) to analyse the pulses of an audio signal was also considered. An ECG would be an interesting way to think about rhythm but ended up being unsuitable to apply to a complex signal. The analysis of timbre was explored through the generation of many spectrograms. Possible methods for comparison were identified as Euclidean distance or Hamming distance. The use of images was preferred due to the ability to store a base64 representation of them, facilitating easy retrieval and utilization for future comparisons.

A larger decision was an audio framework to use. Choices included:

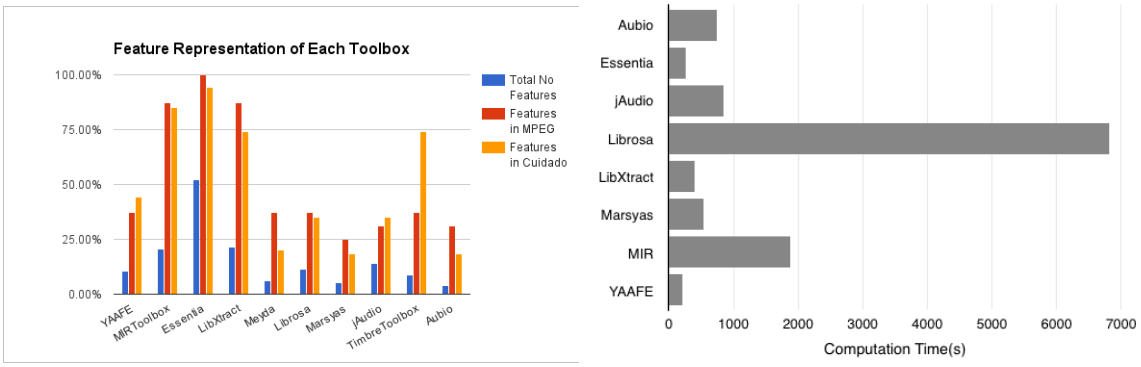
1. Essentia - Bogdanov et al. (2013).
2. Librosa - McFee et al. (2015).
3. Marsyas - Tzanetakis (1994).

Librosa is built on top of the NumPy and SciPy libraries which are well researched and documented, so is a desireable choice for a Python based project. Jupyter notebooks are also an effective way to experiment and evaluate different feature methods.

Essentia is a C++ library that focuses on performance and robustness. It is also well documented and has a large community. Essentia would have been the best choice, but issues were encountered with the installation process and support on windows systems.

Marsyas is also written in C++ focusing on flexibility and extensibility. Marsyas was discovered at a later stage in the project and was not considered as a viable option due to the time constraints.

Moffat et al. (2015) compare multiple frameworks for analysing audio and found that Essentia (Bogdanov et al., 2013) and Marsyas (Tzanetakis, 1994) were strong choices due to the amount of features they provide as well as their computation performance.



(a) (b)

Figure 17: Comparison of audio frameworks (Moffat et al., 2015)

Even though Librosa scores poorly on computational speed (Figure 17b), it is still a viable choice for this project due to the benefits of using Jupyter notebooks. Additionally, the graphical capabilities of Librosa are strong, supporting the usage of image comparison for the timbre and rhythm attributes.

Aubio, another Python library was also considered, however, it is not as well maintained as Librosa and is not as well documented.

3.4.2 BPM

There are many ways to identify BPM, ranging from Phase Deviation to Spectral Difference. As it is a well-defined area, and applying different methods only affects the processing time, Librosa's implementation will be used. We depend on Librosa anyway, so there is no benefit to using a different method. Librosa uses the Spectral Flux approach. (McFee et al., 2015).

3.4.3 Harmonics and Key

There is a lot of information hidden within a waveform but requires a lot of processing to obtain.

To extract this information, we will be focusing around using the Chromagram and Harmonic PCP (HPCP). HPCP's are going to be used to actually identify the key, and we will create it from the chromagram.

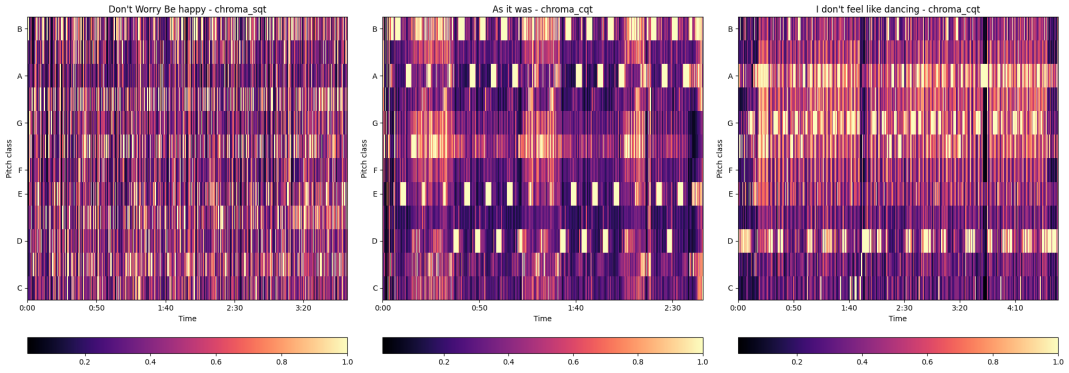


Figure 18: Chromagrams of (McFerrin, 2009), (Styles, 2022), (Sisters, 2006)

The chromagrams show the song segmented into hops (the vertical bars), and split horizontally into musical notes. Pop songs, like 'As it Was' by Styles (2022), typically centre themselves around a few notes, e.g., the notes A, E and D for 'As it Was' and have a simple structure. Whereas 'Don't feel like dancing' by Sisters (2006) has a less simple structure but clearly about A, G and D. The HPCP model will have some additional trimming to make identification easier. This will feed into the TIVlib library to describe the tonal content of audio signals using the vector space.

For each frame, or PCP we can identify a key using TIV (Figure 19). If we compute this for the whole song we can see the harmonic movement of the song over time (Figure 20).

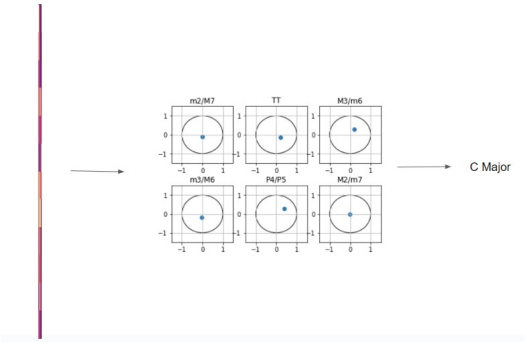


Figure 19: TIV explained graphically

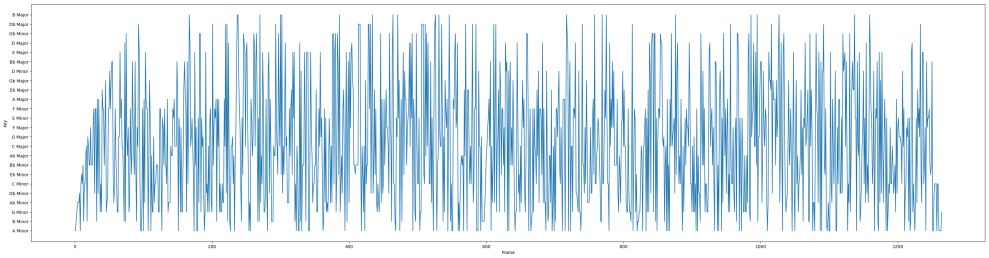


Figure 20: Harmonic Movement

The next step to identify the overall key, will be to construct a set of chords within each key, and then map to a new key. This is done by first predefining the chords within each key, adding all chords that are observed in the track into a set, and then matching it to a new key based on the distribution of chords. In Figure 22 we have multiple choices for the key. Whilst this would not be good for informing about the song to a musician, it is okay in our case as the algorithm is deterministic and will only be comparing against itself. For "As it Was" Styles (2022) *Gb Major* was decided. The key of 'As it Was' is actually *F♯ Minor*. The key predictions for the other songs are as follows.

Song	Predicted	Actual
Don't Feel like Dancing	G Major	G Major
As it Was	$G\flat$ Major	$F\sharp$ Minor
Don't worry be Happy	$D\flat$ Major	B Major

Figure 21: Key Predictions

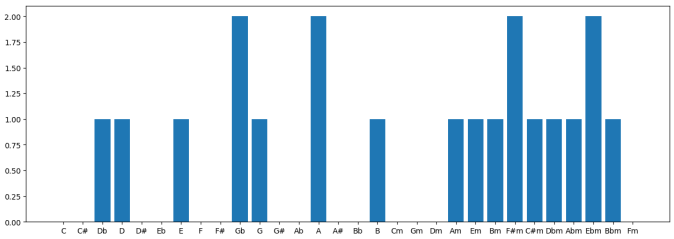


Figure 22: Chord Usage

3.4.4 Further Harmonics

Utilizing TIV we can extract further information than just the key. Using the overall HPCP, we can compute information such as the type of scale used (Chromatic, Triadic, Diatonic …).

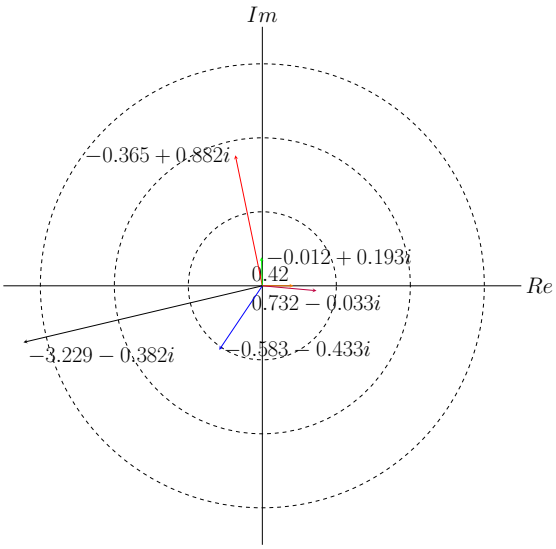


Figure 23: Polar plot of TIV


```
{
    "Chromatic": 0.06729134707780758,
    "Triadic" :  0.060221633044270424,
    "Dyadic" :   0.08726394782337539,
    "Diatonic":  0.2208208745432335,
    "Diminished":0.06647204120292993
}
```

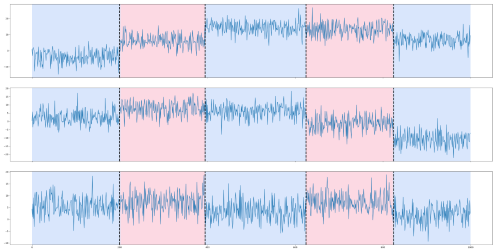

Figure 24: Harmonic Values of TIV

3.4.5 Choosing a Section

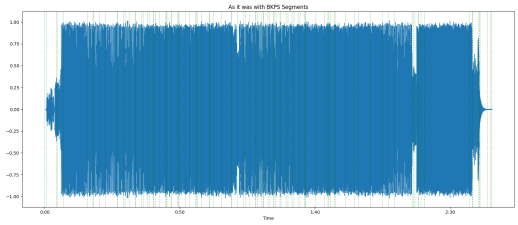
When analysing timbre and rhythm we are going to extract a Chromagram and a Tempogram. Computing these measures takes an amount of time that is not small. As we are interested in the speed of the analysis, we will attempt to select a section of the song that describes most of the song first, and then compute the measure. By reducing the scope, we also enable the ability to compare the measurers visually due to the measure being the same resolution.

To decide the best section of the song, we must divide the track into sections and then decide a chunk based on a cost function.

BKPS Segmentation : To begin selecting the slice of the song that best describes its overall character, we will separate the signal into sections or phrases. Ruptures aids us by allowing us to select from different Cost and Search Models without tuning the hyperparameter's (Truong et al., 2020).



(a) Example of segmentation using Ruptures

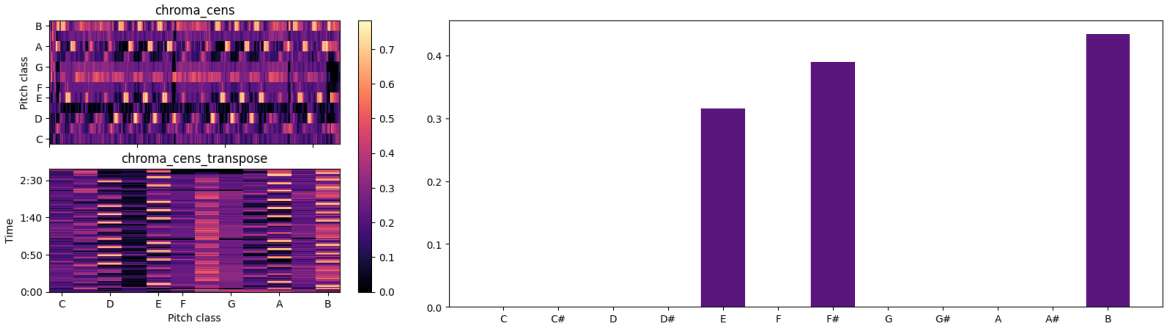


(b) Segmentation of 'As it Was' (Styles, 2022)

The sliding window search algorithm with a Piecewise autoregressive cost model (C_{AR}) will be used as it shows good results for time series data. The model has been applied on EEG/ECG time series, functional magnetic resonance imaging, time series and speech recognition data which fits the application. To select the section, we will create a scoring function and attempt to maximize it.

- **Factor 1, 'Root Harmonic Distance'** → Distance from the most typical harmonic tones.
- **Factor 2, 'Root Mean Squared Energy'** → Distance from the average energy per BKPS segment (phrase).

Factor 1 : Root Harmonic Distance (RHD) First generate a normalized chromagram of the whole song. This will identify primary notes used and give us a 12-element vector per hop.



(a) Chroma Energy Normalized profiles of 'As it Was' (Styles, 2022)

(b) 3-Tuple Chroma Vector profiles of 'As it Was' (Styles, 2022)

Figure 25: RHD Explained

As each chord is normally situates around 3 notes, we will find the largest 3 tuple set $\overline{\boldsymbol{W}}$, and measure the distance between the unit vector $\overline{\boldsymbol{W}}$ and the normalized vectors (Figure 25b). The smaller the distance, the closer the hop is to the overall texture of the song.

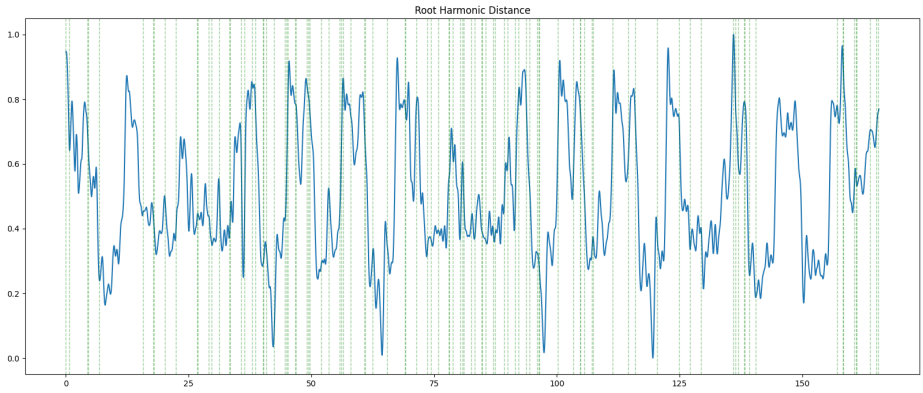
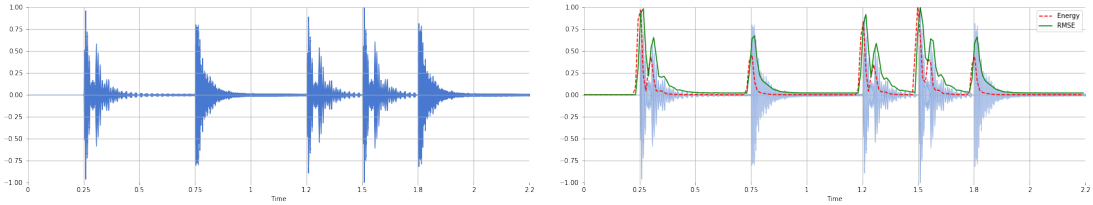


Figure 26: RHD with segments plotted for (Styles, 2022)

The larger the value, the further away we are. We want the smallest value from the graph, this will be the standard harmonic/tones in the song.

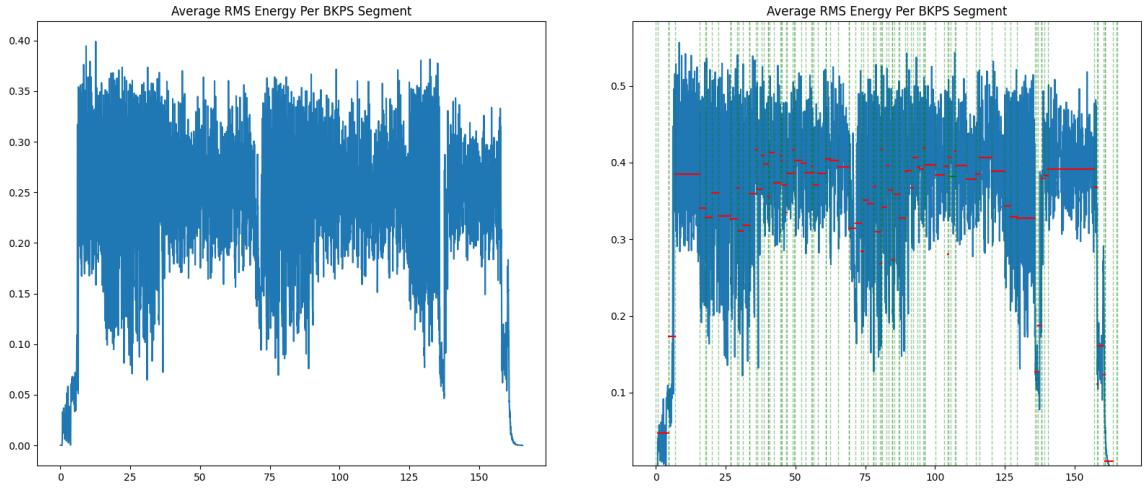
Factor 2 : Root Mean Square Energy (RMSE) per Signal Breakpoint (BKPS) The RMSE defined as $\sqrt{\frac{1}{N}\sum_n |x(n)|^2}$ is a measure of the overall loudness or energy of the signal. In other words, it is a way to quantify the average energy level of the audio signal.



(a) (b)

Figure 27: Diagrams from https://musicinformationretrieval.com/energy.html

The RMSE can be used to infer important parts of a track by identifying sections of the audio signal with high energy levels. For example, in a pop song, the chorus is often the most energetic part of the track, with a higher RMSE than the verses. Similarly, in a rock song, the instrumental breaks or guitar solos may have higher RMSE levels than other parts of the song.



(a) RMSE (b) RMSE with BKPS Averages

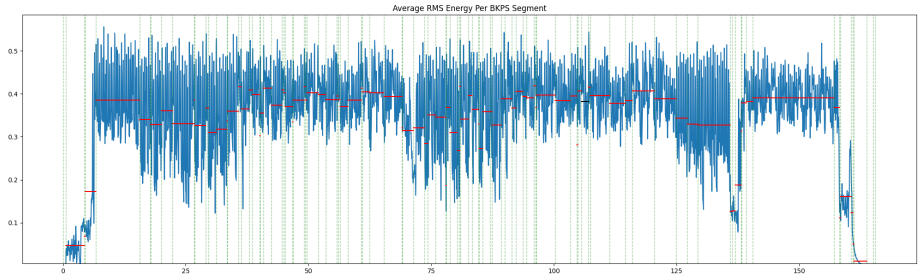


Figure 28: Average RMSE per BKPS (See Appendix 46)

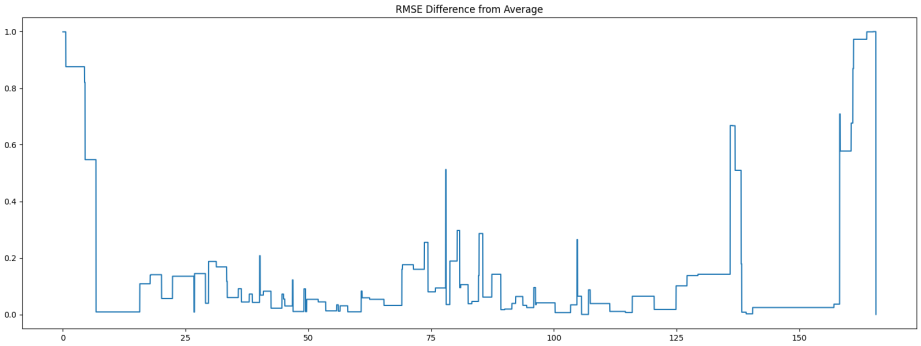


Figure 29: Stepwise graph of Average RMSE per BKPS

Shown in red, are the averages of each RMSE per BKPS segment (Figure 28). Around the 110 second mark, is highlighted in black as the average of the averages. This is the most important part of the song according to Factor 2. This will be clearer as the minimum point on Figure 29.

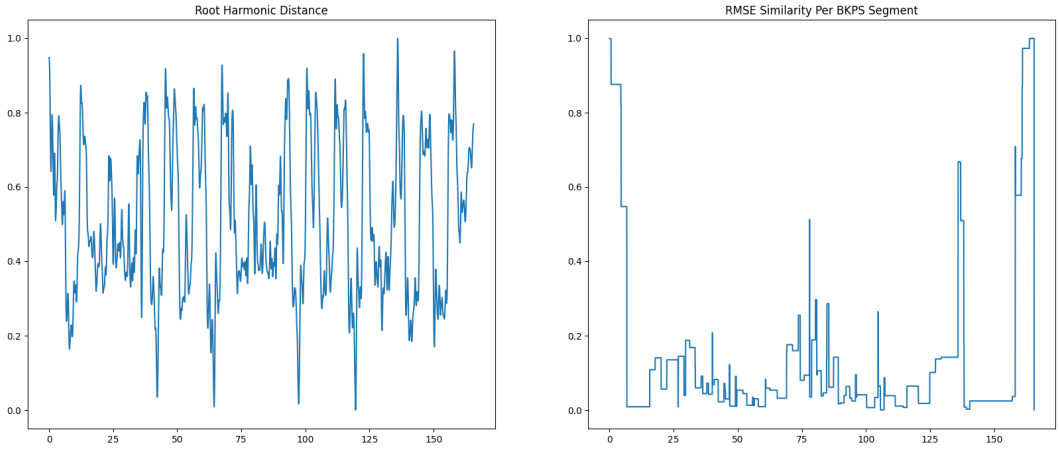


Figure 30: Factor 1 and Factor 2

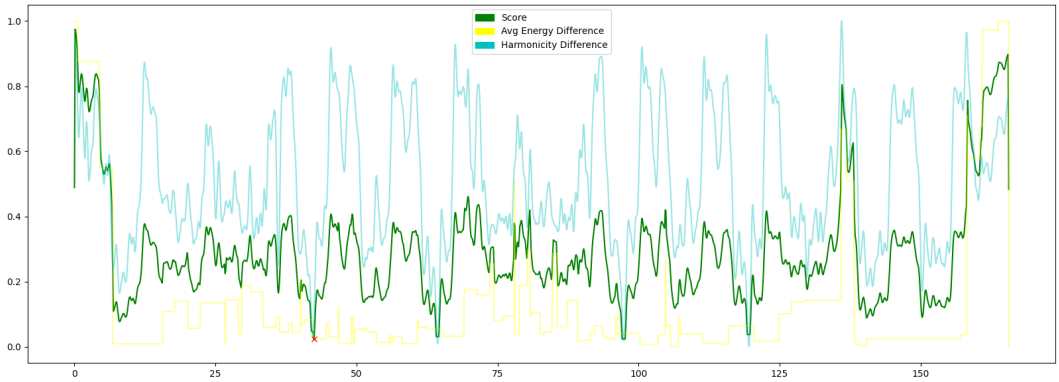


Figure 31: Combined Factors with Minimum point highlighted

$$score = \frac{3 * (max(0, min(15, factor_1)) + 4 * factor_2)}{7}$$

Where both factors are in the range 0-1, and smaller values represent a more similar value.

The peak point identified might not be the best part of the song as it happens during the intro so it might be dissimilar to the rest of the song; the same applies for the outro. We will clamp the intro and outro to direct away from choosing this peak.

We will do this by reducing the magnitude of the score in the edges of the song but reducing the value proportionally from how far away from the beginning or start of the song it is. This will clamp the first and last 15% of the song. We will also invert the scores to allow peak picking.

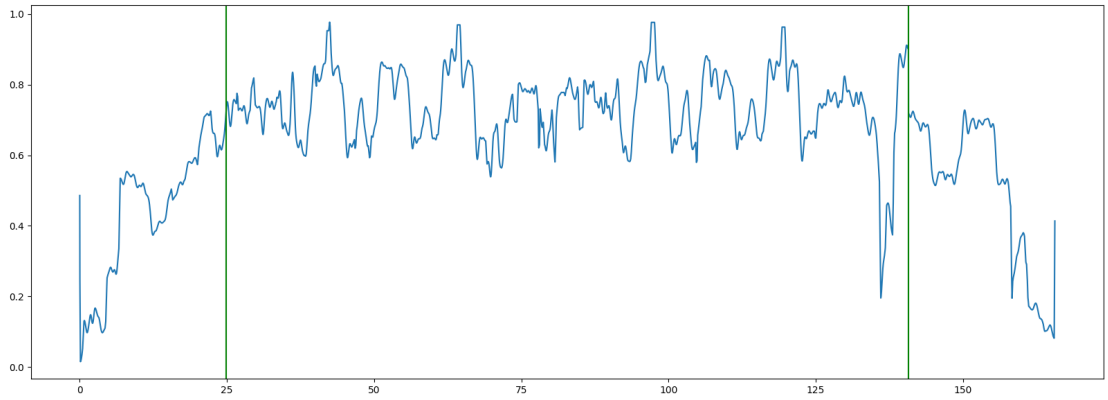


Figure 32: Objective Function with clamping thresholds marked

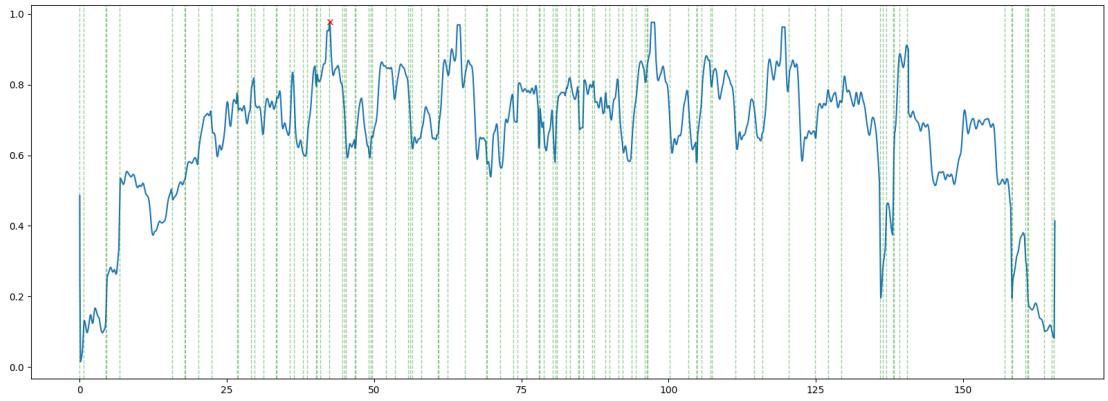


Figure 33: Objective Function with BKPS Segments, Maximum Highlighted

Now we have identified a correct, unbiased point, which describes most of the song can we expand outwards consuming each section until we reach the size of 20 seconds.

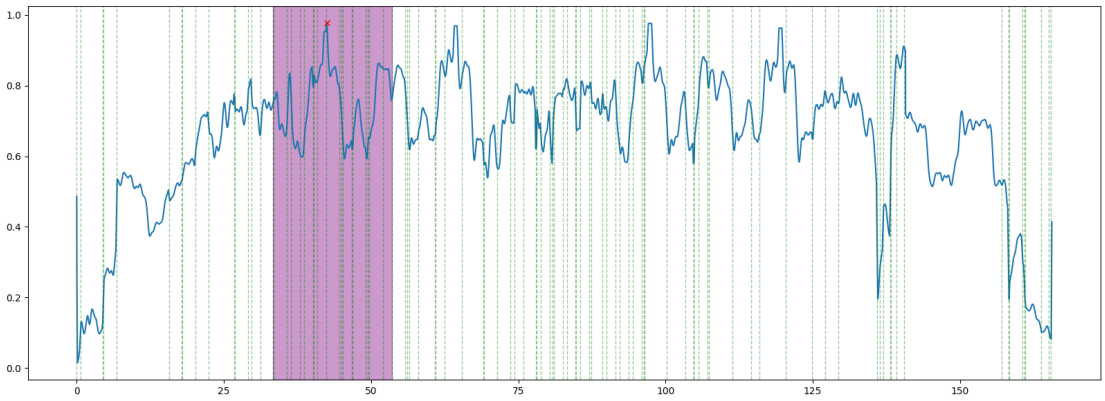


Figure 34: Objective Function with selected region shaded

3.4.6 Timbre

To describe timbre, we will be extracting a MFCC graph, for the section identified as the most representative part of the song. We will then use the MFCC graph to describe the timbre of the song. By analysing the distribution of energy across the different frequency bands, we can get an idea of the overall spectral shape or 'colour' of the sound. For example, a sound with more energy in the higher frequency bands might be perceived as brighter or sharper, while a sound with more energy in the lower frequency bands might be perceived as warmer or deeper. Additionally, the presence or absence of certain harmonics or frequency ratios can also contribute to the perception of timbre and can be identified by looking at the peaks and valleys in the MFCC graph.

Because MFCC's are best used to analyse vocal sounds, we will alter the Mel-Filterbank to reflect this (Table 1). By weighting the bands, we can emphasize certain frequencies and de-emphasize others. This will make the MFCC output more suitable for analysing music.

Frequency Range	**Weighting**	**Explanation**
$F \leq 1000Hz$	0.85	Important as they are fundamental frequencies
$1000Hz \leq F \leq 8000Hz$	0.7	Contains a lot of percussive notes and overtones
$8000Hz \leq F$	0.6	Tones greater than $8KHz$ are still important but less relevant

Table 1: Mel-Filterbank Weightings

3.4.7 Rhythm

To describe rhythm, we will be extracting a Tempogram, for the section identified as the best part of the song. We will then use the Tempogram to understand the patterns of rhythmic accents and the tempo fluctuations that occur throughout the piece.

The resulting tempogram can then be plotted as a graph, with time on the x-axis and density of onsets on the y-axis. This graph can reveal important information about the rhythmic structure of the music, such as the underlying tempo, the presence of syncopation or accent patterns, and the degree of variation in tempo and rhythm.

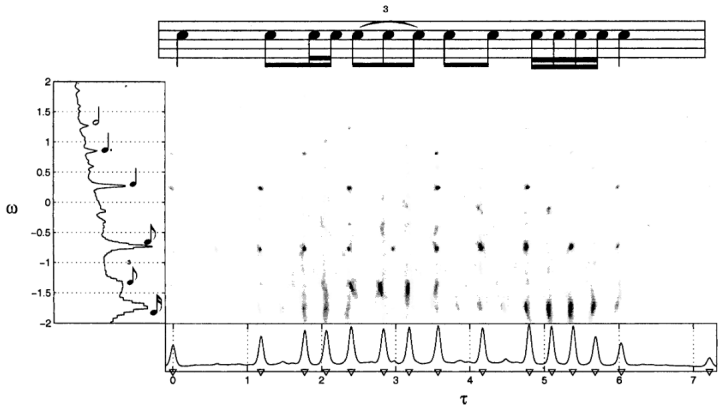


Figure 35: Tempogram illustrating simple rhythm (Cemgil et al., 2000, Figure 4)

For example, a regular, metronomic rhythm might produce a tempogram with a consistent, evenly spaced pattern of peaks and valleys, indicating a consistent tempo and rhythmic density. On the other hand, a more complex rhythm with irregular accents or tempo changes might produce a more complex and varied tempogram, with peaks and valleys of varying heights and spacing (Figure 36).

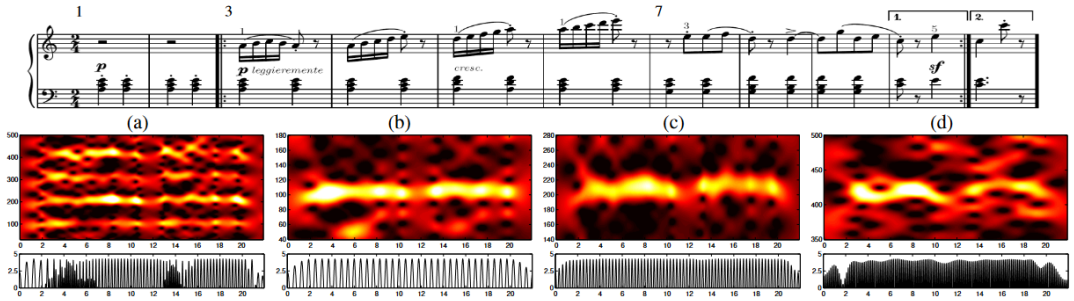


Figure 36: Multiple complex rhythms (Hirata et al., 2009, Figure 4, Page 193)

By analysing the tempogram, we can gain insights into the underlying rhythmic structure of a piece of music.

4 Comparing fingerprints

The comparison will be focused on 5 defining features. BPM, Timbre, Rhythm, Key and Harmonics. Key and harmonics will be grouped together because the key of a song is determined by the tonal centre of the song, and the notes that are used in the song are typically selected to harmonize with this tonal centre[7].

1. **BPM** detection will primarily be used to identify the tempo of the song for calculating a songs similarity score. Inaccurate BPM detection can result in clashing beats and awkward transitions which can be jarring to the listener which goes against the idea of the system.

2. **Timbre**, the colour or the perceived quality of a musical tone, is a very important attribute to compare songs with. It will be the most important attribute to decide if two songs are similar as it is the most defining attribute of a song. Timbre is then a broad term for the distinguishable characteristics of a tone, so it is hard to define what timbre is. It could be described as the combination of attributes that make a song unique.

3. **Rhythm** can help identify songs which have a similar underlying beat structure. It can also help identify songs which have a similar feel or mood. Moreover, the rhythm can also be used to identify songs that are in the same genre or sub-genre, as many genres have specific rhythmic characteristics.

4. **Key and Harmonics.** Harmonics is the study of the frequency relationships between musical notes played simultaneously. A key feature in music is the relationship between notes played in a specific key, which creates a harmonic structure. Therefore, identifying the key of a song is important in analysing its harmonics. Harmonic analysis can reveal relationships between songs based on shared key signatures and chord progressions. This can provide insights into the musical structure of a piece and assist in identifying songs that have similar harmonic content.

[7]The tonal centre refers to the main or central pitch around which a musical composition or song is structured.

When evaluating the similarity between different attributes, the most effective approach is to assign a numerical value between 0 and 100 to each attribute. This method simplifies the comparison process and allows for a clear understanding of how similar the attributes are to each other.

4.1 BPM

BPM is the easiest attribute to compare as it is a linear scale, we want the greatest difference to tend to 100, and the same to tend to 0. Faster BPM songs will be more similar to other fast songs, and not similar to slow songs. We can calculate this using the percentage difference.

$$BPM_{sim} = 100 * \frac{1 - |x - y|}{x + y} \tag{1}$$

Figure 37: Percentage Difference Formula

4.2 Harmonics

TIVLib has an implementation of the Small-Scale Compatibility (SSC) between two Tonal Interval vectors (TIV). Compatible vectors tend to share similar tonal characteristics. For example, two tonal interval vectors that share the same tonal centre, or tonic, are likely to be more compatible than vectors with different tonics. SSC will score these higher (Bernardes et al., 2016).

$$\begin{gathered} Q_i(\bar{V}, \bar{W}) = \bar{V}_i - \bar{W}_i,\ D_i(\bar{V}, \bar{W}) = \frac{\bar{V}_i + \bar{W}_i}{2}\ TIV_{norm} = 26.396022427631024 \\ r = \frac{\sqrt{real(\Sigma_i(Q_i(\bar{V}, \bar{W}) * conj(Q_i(\bar{V}, \bar{W}))))}}{2 * TIV_{norm}} \\ d = 1 - \frac{\sqrt{real(\Sigma_i(D_i(\bar{V}, \bar{W}) * conj(D_i(\bar{V}, \bar{W}))))}}{TIV_{norm}} \\ ssc = \frac{100}{e^{|d*r|}} \end{gathered} \tag{2}$$

Figure 38: SSC Formula

4.3 Key

To calculate the difference of key, we can use the concept of the circle of fifths.

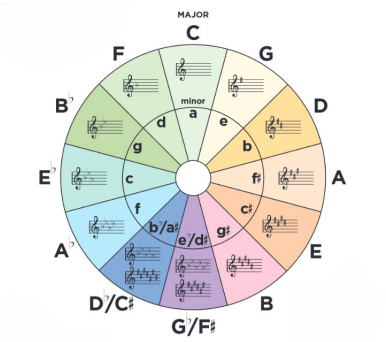


Figure 39: Circle of Fifths by D'addario

$$Key_{sim} = 100 * \frac{1 - |i - j|}{12} \tag{3}$$

Where i, j are the indexes of the keys.

Figure 40: Percentage Difference Formula

This diagram allows us to visualize all keys, their relative minor, and their accidentals. Additionally we can picture the distance keys are from each other, by how many steps around we have to take. By taking the two indexes of the keys in this order, we can calculate their difference.

4.4 Comparing Spectrograms

A fast method to compare images for areas of similarity, is the XOR function. We can apply it to images and expect to see either: a blank image for identical images or a combination of both images for difference images.

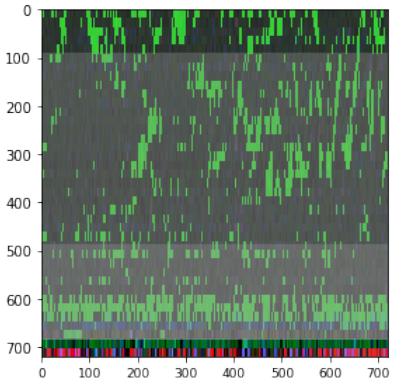


Figure 41: Result of XORing two spectrograms

We can leverage this operation and apply it to each colour channel. The colours we are using in the spectrograms will be from a defined gradient and will result in specific colours being generated when we XOR. With the most similar resulting in a less colourful image.

Another benefit of applying a pixel-wise XOR, is that it not only compares the colours (Spectrogram intensity) but also position. If songs have a similar structure this will show in the spectrogram as blank patches.

4.4.1 Timbre

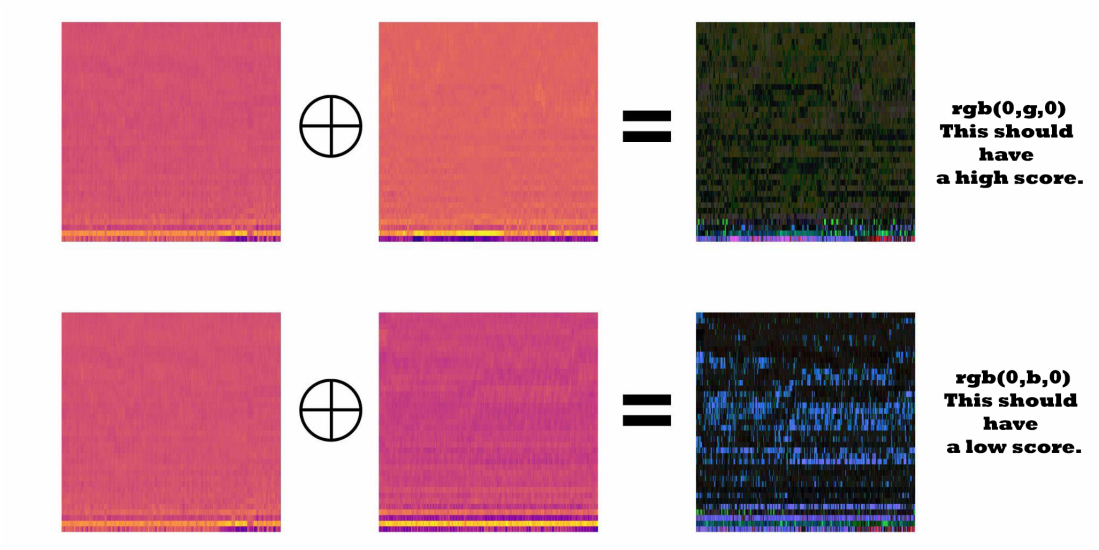


Figure 42: XOR of similar(Top) and dissimilar(Bottom) MFCC's

In the specific case of timbre, we compare a 39 band MFCC spectrogram. After applying the XOR operation, we can see two different types of images. When we compare similar colours, i.e. at two different ends of the spectrum we get a blue colour. But when we XOR two colours closer together on the spectrum we either get white, or a green. Due to this effect, we can filter out the Green channel from our image to increase the severity of difference in the score.

4.4.2 Rhythm

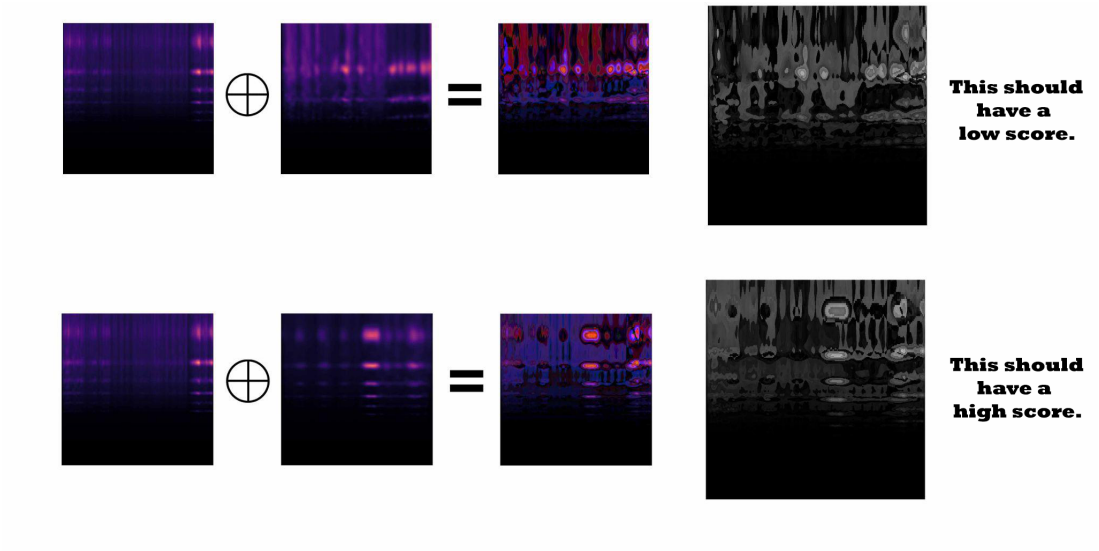


Figure 43: XOR of similar(Top) and dissimilar(Bottom) Tempograms

Tempograms have a less defined structure than MFCC's. This makes it harder to compare as it introduces irregular shapes, this removes the ability to filter an unimportant colour channel out. To combat this we can convert the image to greyscale for an easier approach (Figure 43). We can now XOR as normal.

4.4.3 Scoring

After the XOR operation, we can simply sum the RGB value (0-255) for each pixel, then divide it by $255 * colour\_depth * resolution$.

$$Spec_{sim} = \frac{\sum_{i=0}^{r}\sum_{j=0}^{r}\sum_{c=0}^{d}(x_{(i,j,c)} \oplus y_{(i,j,c)})}{255 * d * r} \tag{4}$$

Where $r, d = resolution, depth$ and x, y are the two images.

Figure 44: Spectrogram Comparison Formula

4.5 Combining Attribute Values

To combine each attribute, we could use either the Arithmetic Mean, or the Harmonic Mean (Ferger, 1931). The latter is more applicable due to the ability to weight each value.

In general, the harmonic mean is lower than the standard mean when the data set includes values that are widely dispersed, and it is higher than the standard mean when the data set includes values that are clustered around a few low values. For example, if we have a song which has a similar key, but the timbre comparison scores badly, it will negatively scale the overall compatibility.

In the implementation used, we also penalize large differences regardless of type. This combats a single attribute reducing the score more significantly than it should. Such as key which has a higher chance to score poorly (Table 2).

BPM	**Key**	**Harmonic**	**Timbre**	**Rhythm**
90	5	50	55	20

Table 2: Harmonic Mean Weighting for Compatibility

5 Evaluation

5.1 Testing

5.1.1 Approach

Testing the results of the fingerprinting and comparisons was a challenge due to the subjectivity of music taste. User testing was adopted to identify the best section of a song, and what people believe to be good songs to play after another (Table 3). Users were asked to listen to a selection of songs which had some which could be recognized as slightly similar and others which were very different. The users were asked to select the best section of the song, and then to create a playlist using these songs and rate how well they went together.

	Song
1	As it Was
2	Single Ladies
3	Mr Brightside
4	Fly Me To The Moon
5	Get Lucky

Table 3: Song Choice for User Testing

The aim of this survey was firstly to understand how well the system was able to identify the best section of a song, from which we compare attributes in, and additionally how well we can compute similarity between songs. A concern was the algorithms would be overfitted, or biased to the taste of it's designer. To combat this, the survey was designed to be as objective as possible, asking users to first select a song of their choice, identify the best section of the song, and then to create a playlist using these songs and rate how well they went together. The results of the playlist will only be considering the adjacent songs, and not the whole playlist. This is because the system is only concerned with the next song to be played, and not the whole playlist.

Additionally, unit testing was also used to test the server code. The server was unit tested to cover 57% of statements. The Python code for fingerprinting was not unit tested but the results of the analysis was tested via the user testing.

5.1.2 Results

After creating their playlists, users were asked "How well does song y sound after song x?". The results of this question are shown in Figure 45, shown are the average scores for each permutation of songs. In almost all cases, except "Fly me to the Moon", the results show that the top 2 songs to be played after the current song, agree with the results shown in Table 4. The fact that "Fly me to the moon" wasn't selected frequently during the research could be a reason for its incompatibility, a wider audience could more effectively evaluate the algorithm. This is a good example of how the results of the comparisons are subjective and depend on the user's taste in music.

	As It Was	**Single Ladies**	**Mr Brightside**	**Fly Me To The Moon**	**Get Lucky**
As It Was	-	0.49	49.88	39.67	72.53
Single Ladies	0.49	-	7.79	36.87	29.47
Mr Brightside	49.88	7.76	-	27.31	80.37
Fly Me To The Moon	39.67	36.87	27.31	-	33.08
Get Lucky	72.53	29.47	80.37	33.08	-

Table 4: Compatibility Ratings from Implementation

The evaluation also included the identification of the best section of the song. Users were asked to provide a timestamp of the best section of the song. The results of this are shown in Table 5. The results show that the best section of the song is selected correctly in most cases, within a region of 30 seconds. This is a good result as it proves that the segment selection is suitably tuned, but perhaps would be better if the question that was asked was not "Select the **best** section of the song", but to select the region which **describes** the song the best, as this is what the region selection code is doing.

	Algorithm	**Modal of Users**
As it Was	00 : 56	00 : 30
Single Ladies	01 : 37	01 : 03
Mr Brightside	03 : 15	02 : 00
Fly Me To The Moon	01 : 35	00 : 48
Get Lucky	01 : 16	01 : 05

Table 5: Timestamp identified as the best of the song by algorithm and users

5.2 Evaluation against Aims

Overall, this project achieved its main aim of calculating two songs compatibilities based purely on the waveform. It also achieved its goal of supporting multiple audio services and providing insights to the user on how compatible their songs are with considerations of comparison speed taken into account.

Currently, the application supports two streaming services, Spotify, and YouTube. With little extra work, the application could support more services such as SoundCloud or Deezer by implementing the API for the service. This is due to the simple and scalable nature of the code. More support drastically improves the user experience, as the user is able to use the application with their preferred service. Additionally, it allows for a more seamless collaborative playlist. Another addition would be to extend the authentication layer to include password-protected sessions, enabling users to host secure private sessions.

The application also successfully provides the user with lots of information about the songs which are playing. This information includes the song's tempo, key, and harmonic's. This information is provided to the user in a way that is easy to understand. More complex features, used for comparisons, are presented in a way that is easy to compare between songs. This implementation ends up being superior to the JQBX platform in terms of it's cross-platform support and the informative musical information provided to compare songs. Part of the initial aims of this project, was the mixing of tracks together. Unfortunately, the project did not touch on any programmatic blending of songs together using techniques like pitch shifting or beat tracking. The reasons for this not being achieved were due to the unexpected complexity of generating the fingerprint.

A major requirement for the application was the accuracy of the similarity scores. From the user tests, we see the way the algorithm works is very similar to what a human would have picked. Not all genres have been user tested and further testing might uncover limitations of the algorithm, which could be improved by using a more complex fingerprint. Areas that could be

further developed include the expansion of the fingerprint. The content-based fingerprint currently only uses the spectrogram of the song, but it could be expanded to include more audio features such as the density of instruments in the song and the density of vocals in the song. This would allow for a more accurate similarity rating, ultimately leading to a better user experience. Another feature could be the suggestion of songs based on the pre-fingerprinted database, if there are no songs in the users queue we could suggest a suitable song. This would expand the application use cases.

With regards to the codebase, enhancements could be made by utilizing either low level languages such as C or using a GPU-enabled audio analysis library, such as nnAudio. Low-level languages like C are typically faster and more memory-efficient than high-level languages like Python. By writing the codebase in C, the system could execute faster and more efficiently.

This project has achieved its main objectives of calculating the compatibility of two songs based on waveform and supporting multiple audio services, while providing valuable insights to users. While further development is required to enhance the application's similarity rating, this project has the potential to change the way users collaborate on music playlists, providing a more efficient and informative solution than existing platforms.

6 Conclusion

In conclusion, the aim of this project was to develop a system that could identify the similarity between songs based on their audio content using a mathematical approach and reason the results to the user in colloquial language. The project was successful in achieving its primary objectives by accurately fingerprinting songs and identifying similar songs using the chosen attributes of BPM, Timbre, Rhythm, Key, and Harmonics. However, there were some limitations encountered during the development process, such as the inability to programmatically mix songs together and the support of more than two streaming services.

Despite these limitations, the project demonstrated adaptability by allowing easy extension to other services and attributes, which can be implemented in future development to enhance the system's functionality. The project's overall successes were the musical colloquialism of the system and ease of use, which makes it user-friendly.

The project also contributed to the research field of MIR by developing a method to compare spectrograms and identify a songs modal harmonic value, representing the most frequent and central tonal component (Root Harmonic Distance).

Overall, this project has achieved its objectives by developing a system that accurately identifies the similarity between songs and reasons the results to the user in colloquial language. With further development, this system has the potential to be extended to other services and attributes, making it a valuable tool for music lovers to discover new music and listen together.

7 Bibliography

References

Adadi, A. and Berrada, M. (2018). Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6:52138–52160. [Accessed March 21st, 2023].

Astley, J., Biancardi, A., Hughes, P., Marshall, H., Collier, G., Chan, H.-F., Saunders, L., Smith, L., Brook, M., Thompson, A., Rowland-Jones, S., Skeoch, S., Bianchi, S., Hatton, M., Rahman, N., Ho, L.-P., Brightling, C., Wain, L., Singapuri, A., and Tahir, B. (2023). Implementable deep learning for multi-sequence proton mri lung segmentation: A multi-center, multi-vendor, and multi-disease study. *Journal of magnetic resonance imaging : JMRI*. [Accessed March 21st, 2023].

Barbedo, J. G. A. and Lopes, A. (2006). Automatic Genre Classification of Musical Signals. *EURASIP Journal on Advances in Signal Processing*, 17(1):064960. [Accessed March 21st, 2023].

Bello, J., Daudet, L., Abdallah, S., Duxbury, C., Davies, M., and Sandler, M. (2005). A tutorial on onset detection in music signals. *IEEE Transactions on Speech and Audio Processing*, 13(5):1035–1047. [Accessed March 21st, 2023].

Bernardes, G., Cocharro, D., Caetano, M., Guedes, C., and Davies, M. E. (2016). A multi-level tonal interval space for modelling pitch relatedness and musical consonance. *Journal of New Music Research*, 45(4):281–294. [Accessed March 21st, 2023].

Boer, D. (2009). Music makes the people come together: Social functions of music listening for young people across cultures. [Accessed March 21st, 2023].

Bogdanov, D., Néstor, W., Emilia, G., Gulati, S., Perfecto, H., Mayor, O., Gual, R., Salamon, J., and Zapata González, V. (2013). Essentia: An audio analysis library for music information retrieval. In *Proceedings of the 14th International Society for Music Information Retrieval Conference (ISMIR)*, pages 493–498. International Society for Music Information Retrieval (ISMIR). [Accessed December 15th, 2022].

Brigham, E. O. and Morrow, R. E. (1967). The fast fourier transform. *IEEE Spectrum*, 4(12):63–70. [Accessed November 2nd, 2022].

Cash, J. (2002). Hurt. https://www.youtube.com/watch?v=8AHCfZTRGiI. [Accessed March 21st, 2023].

Çanbay, F., Tokak, B., and contributors (2023). Fiber. https://github.com/gofiber/fiber. Accessed: March 26, 2023.

Cemgil, A. T., Kappen, B., Desain, P., and Honing, H. (2000). On tempo tracking: Tempogram representation and kalman filtering. *Journal of New Music Research*, 29(4):259–273. [Accessed March 21st, 2023].

Dixon, S. (2001). Automatic extraction of tempo and beat from expressive performances. *Journal of New Music Research*, 30(1):39–58. [Accessed January 28th, 2023].

fasthttp developers (2016). fasthttp. https://github.com/valyala/fasthttp. fasthttp is actively maintained to this day. [Accessed March 21st, 2023].

Ferger, W. F. (1931). The nature and use of the harmonic mean. *Journal of the American Statistical Association*, 26(173):36–40. [Accessed March 21st, 2023].

Fette, I. and Melnikov, A. (2011). The websocket protocol. RFC 6455, RFC Editor. [Accessed March 21st, 2023].

FFmpeg developers (2000). FFmpeg. https://github.com/FFmpeg/FFmpeg. FFmpeg is actively maintained to this day. [Accessed March 21st, 2023].

Fielding, R., Nottingham, M., and Reschke, J. (2022). Http/1.1. STD 99, RFC Editor. [Accessed March 21st, 2023].

Framires, A. (2019). Tivlib: An open-source library for the tonal description of musical audio works. https://github.com/aframires/TIVlib. [Accessed March 21st, 2023].

Fujishima, T. (1999). Realtime chord recognition of musical sound: Asystem using common lisp music. In *Proceedings of the International Computer Music Conference 1999, Beijing*. [Accessed March 21st, 2023].

Google Chrome Developers (2021). Removing push. *Google Chrome Developers Blog*. [Accessed March 21st, 2023].

Google Developers (2023). Google data api for youtube. https://developers.google.com/youtube/v3.

Griesemer, R., Pike, R., and Thompson, K. (2022). The go programming language specification - go.dev. [Accessed March 21st, 2023].

Gupta, S., Jaafar, J., Ahmad, W. W., and Bansal, A. (2013). Feature extraction using mfcc. *Signal & Image Processing: An International Journal*, 4(4):101–108. [Accessed March 21st, 2023].

Gurjar, K. and Moon, Y.-S. (2018). A comparative analysis of music similarity measures in music information retrieval systems. *Journal of Information Processing Systems*, 14:32–55. [Accessed March 21st, 2023].

Hirata, K., Tzanetakis, G., and Yoshii, K., editors (2009). *Proceedings of the 10th International Society for Music Information Retrieval Conference, ISMIR 2009, Kobe International Conference Center, Kobe, Japan, October 26-30, 2009*. International Society for Music Information Retrieval. [Accessed March 21st, 2023].

Jacobson, D. (2014). The elements of music. [Accessed March 21st, 2023].

Jones, M., Bradley, J., and Sakimura, N. (2015). Json web token (jwt). RFC 7519, RFC Editor. [Accessed March 21st, 2023].

Karydis, I., Lida Kermanidis, K., Sioutas, S., and Iliadis, L. (2013). Comparing content and context based similarity for musical data. *Neurocomputing*, 107:69–76. [Accessed March 21st, 2023].

Katrina and the Waves (2009). Walking on sunshine. https://www.youtube.com/watch?v=iPUmE-tne5U. [Accessed March 27st, 2023].

Lotto, A. and Holt, L. (2010). Psychology of auditory perception. *WIREs Cognitive Science*, 2(5):479–489. [Accessed March 21st, 2023].

McFee, B., Raffel, C., Liang, D., Ellis, D. P., McVicar, M., Battenberg, E., and Nieto, O. (2015). librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, volume 8, pages 18–25. [Accessed March 21st, 2023].

McFerrin, B. (2009). Bobby mcferrin - don't worry be happy (official video). https://www.youtube.com/watch?v=d-diB65scQU. [Accessed March 21st, 2023].

Moffat, D., Ronan, D., and Reiss, J. D. (2015). An evaluation of audio feature extraction toolboxes. [Accessed March 21st, 2023].

Müller, M., Ellis, D., Klapuri, A., and Richard, G. (2011). Signal processing for music analysis. *Selected Topics in Signal Processing, IEEE Journal of*, 5:1088 – 1110. [Accessed March 21st, 2023].

Oku, K. (2017). An http status code for indicating hints. RFC 8297, RFC Editor. [Accessed March 21st, 2023].

Oliveira, J. L., Gouyon, F., Martins, L. G., and Reis, L. P. (2010). IBT: A Real-time Tempo and Beat Tracking System. In *ISMIR*, pages 291–296. [Accessed March 21st, 2023].

Park, S. Y. and Kaneshiro, B. (2021). Social music curation that works: Insights from successful collaborative playlists. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW1):1–27. [Accessed February 16th, 2023].

Pedersen, P. (1965). The mel scale. *Journal of Music Theory*, 9(2):295–308. [Accessed March 21st, 2023].

Rentfrow, P. J., Goldberg, L. R., and Levitin, D. J. (2011). The structure of musical preferences: A five-factor model. *Journal of Personality and Social Psychology*, 100(6):1139–1157. [Accessed March 21st, 2023].

Sisters, S. (2006). I don't feel like dancin'. https://www.youtube.com/watch?v=4H5I6y1Qvz0. [Accessed March 21st, 2023].

Styles, H. (2022). As it was. https://www.youtube.com/watch?v=H5v3kku4y6Q. [Accessed March 21st, 2023].

Temperley, D. (1999). What's key for key? The Krumhansl-Schmuckler key-finding algorithm reconsidered. *Music Perception*, 17(1):65–100. [Accessed March 21st, 2023].

The Spotify Development Team (2023). Spotify developer terms of use. https://developer.spotify.com/terms. Accessed: March 27, 2023.

Thomson, M. and Benfield, C. (2022). Http/2. RFC 9113, RFC Editor. [Accessed March 21st, 2023].

Truong, C., Oudre, L., and Vayatis, N. (2020). Selective review of offline change point detection methods. *Signal Processing*, 167:107299. [Accessed March 21st, 2023].

Tzanetakis, G. (1994). Marsyas - Music Analysis, Retrieval and Synthesis for Audio Signals. [Accessed March 21st, 2023].

Ullrich, K., Schlüter, J., and Grill, T. (2014). Boundary detection in music structure analysis using convolutional neural networks. In *ISMIR*, pages 417–422. [Accessed March 21st, 2023].

Wang, X. and Wang, Y. (2014). Improving content-based and hybrid music recommendation using deep learning. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 627–636. [Accessed March 21st, 2023].

Wankhammer, A., Vaughan, I., Bradley, L., and P, A. (2010). Music structure discovery based on normalized cross correlation. In *Proc. of the 13th International Conference on Digital Audio Effects (DAFx-10)*, pages 488–493. [Accessed December 27th, 2022].

Wayne, J. (2021). Lecture 18: Music information retrieval. https://www.cs.bu.edu/fac/snyder/cs583/Lectures/Lecture%2018.pdf. Slide 11, Boston University, CS 583 [Accessed March 28th, 2023].

youtube-dl Development Team, T. (2006). youtube-dl. https://github.com/ytdl-org/youtube-dl. [Accessed: March 27, 2023].

yt-dlp Development Team, T. (2020). yt-dlp. https://github.com/yt-dlp/yt-dlp. [Accessed: March 27, 2023].

A Appendix

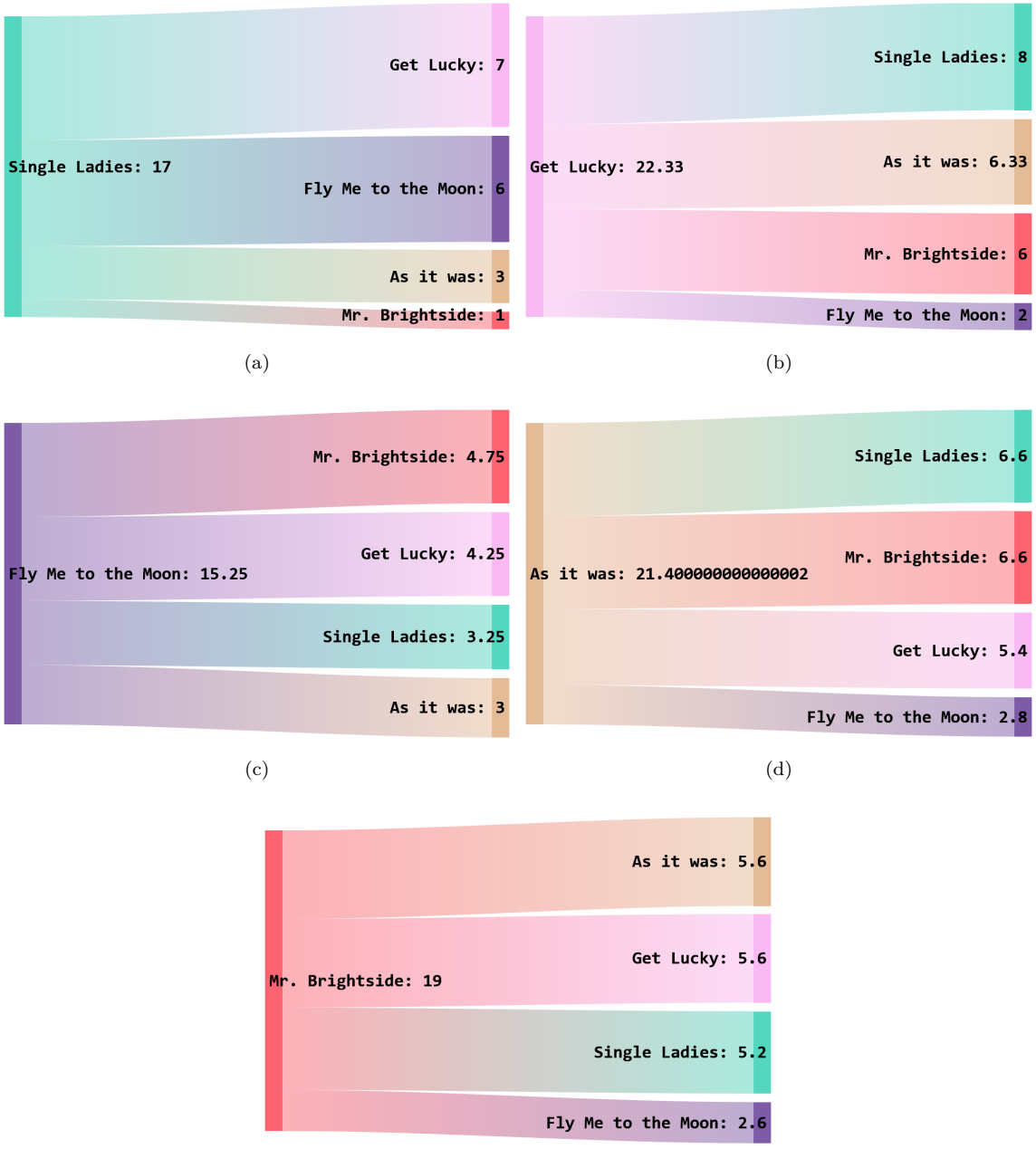


Figure 45: User testing results for song compatibility.

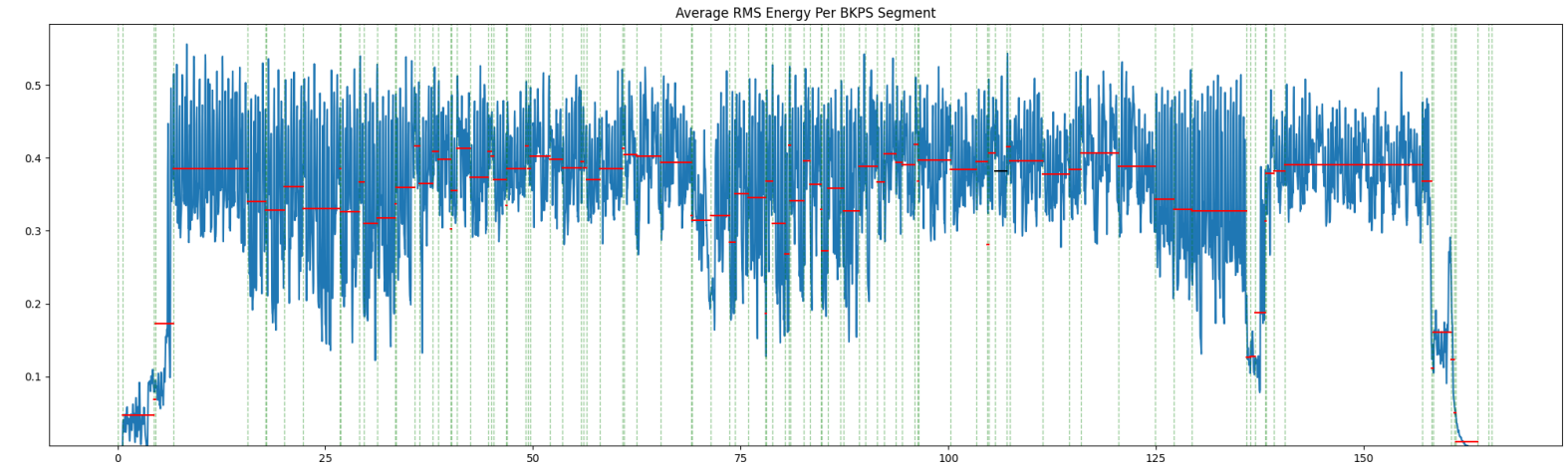


Figure 46: Average RMSE per BKPS

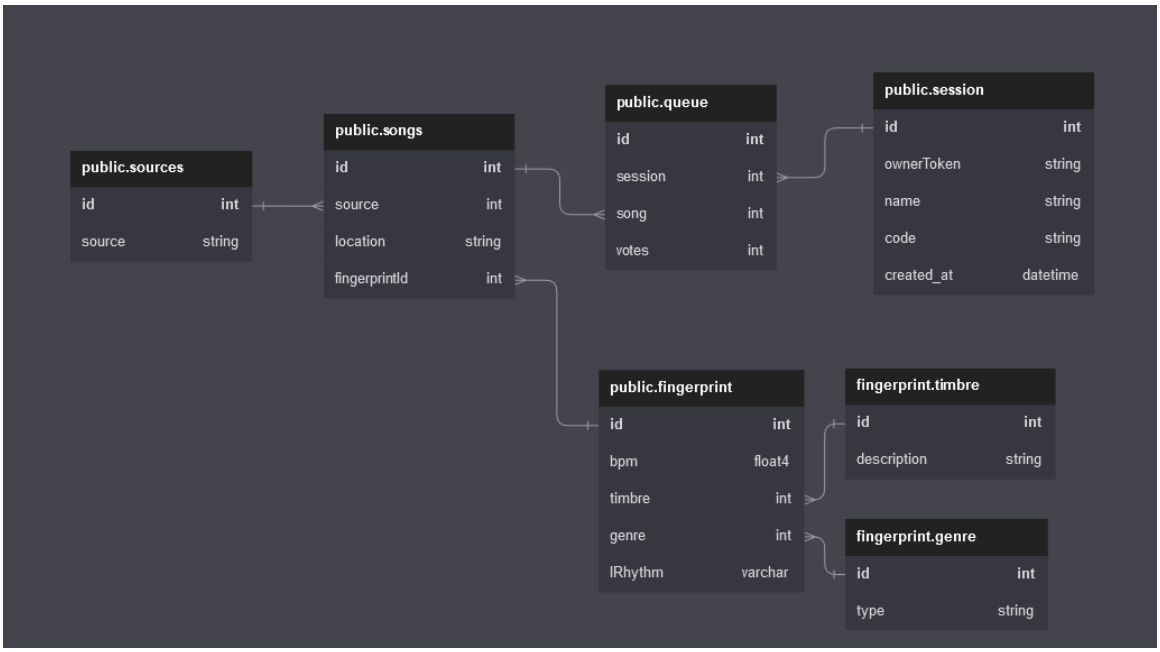


Figure 47: Initial Database Structure

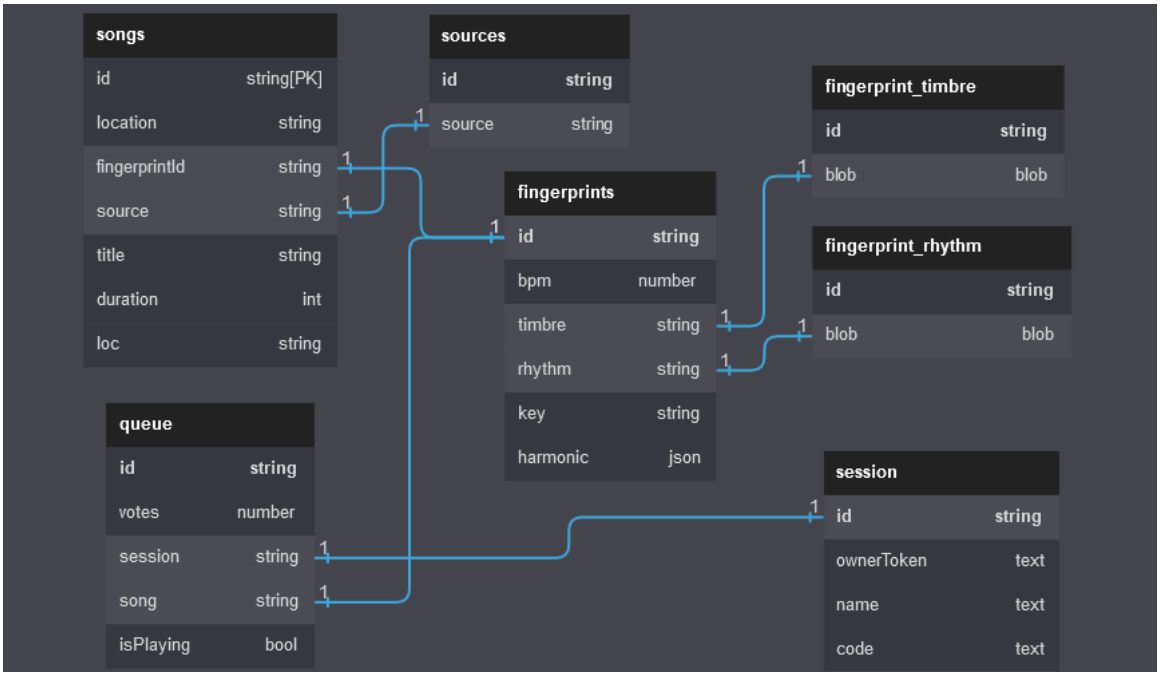


Figure 48: Actual Database Structure