An Automatic Mashup System for music creation

Project ID: HO2

Final year Project Final Report

Auto Masher

An Automatic Mashup System for music creation

Submitted in partial fulfillment for the course COMP4981 (Final Year Project) in the Department of Computer Science,
The Hong Kong University of Science and Technology
in the academic year 2023 - 2024

Group members:

- Lau Yan Hei 20765046
- Chang Joshua 20777233
- Chau Yu Foon Darin 20777063
- Chan Chun Yuen 20774190

Advisor:

- Prof. Andrew B. Horner

Date of submission: 18 April 2024

Abstract

Pop songs are an extremely popular form of contemporary art. Given the considerable volume of new pop songs released periodically, there inevitably exist combinations of them that harmonize well together. Musicians have long sought methods to blend these similar songs together, a process known as musical mashups. However, achieving a successful mashup requires extensive familiarity with many pop songs across different genres and languages, and a profound understanding with music theory. To address these challenges, we propose an innovative system designed to generate musical mashups automatically. We researched and developed the system to generate short musical mashups given a song specified by the user. We achieve this by an in-depth analysis of a wide range of contemporary music, then creating a database containing these pre-analyzed data, and lastly, an automatic mashup algorithm that interweaves the given song with the most harmonious song from the database. We believe this system can be a great source of inspiration for amateur and professional mashup creators alike, saving time and increasing accessibility to the world of music creation.

Table of contents

Section 1: Introduction

1.1 What are musical mashups?

A *musical mashup* is a genre of music created by superimposing two or more existing pieces of music, typically pop songs, together in a way such that the outcome is harmonious. In tonal music theory, harmony can be broadly defined as the combination of different notes together in a manner which results in a pleasant tone to the ear. For example, within the same *octave*[1], the *notes*[2] C and G produces a perceivably pleasant (known as *consonant* in music theory) sound when played simultaneously as their *interval*[3] is *simple*[4], while the note C and C-sharp produce a perceivably unpleasant (known as *dissonant* in music theory) sound as their interval is *not simple*[5]. Although what constitutes being "harmonious" is nuanced, at minimum we would expect the harmony and rhythm of the songs used to mashup to match; the songs should sound pleasant when combined together.

One prominent way of making music mashups has been vocal swapping. Mashup artists will take two different pop songs and overlay the vocal track of one on top of the instrumental track of another, and this has proven to give promising results in the past. Even with only two songs, the outcome can already be pleasant. In fact, sometimes having more than two songs may clutter the resulting mashup.

Figure 1: A successful mashup of two songs by mashup artist oneboredjeu [1]:
(1) Call me maybe, by Carly Rae Jepsen
(2) X Gon' Give It To Ya, by DMX

[1] Generally speaking, an octave consists of 12 pitches. And all pitches within the same octaves maintains a fundamental frequency ratio at most 2.

[2] A musical *note* is a sound with a specific frequency played by an instrument. It is what we hear when we press one piano key or pluck one guitar string.

[3] An *interval* is the "vertical" distance between two notes.

[4] A *simple* interval is a means that the fundamental frequency ratio between the two notes can be approximated by a rational number with a small integer denominator. Notes with simple intervals to one another produce a consonant sound.

[5] An interval is *not simple* means the fundamental frequency ratio between the two notes cannot be approximated by a rational number with a small integer denominator. Notes with simple intervals to one another produce a dissonant sound.

The workflow of a typical mashup procedure is as follows:

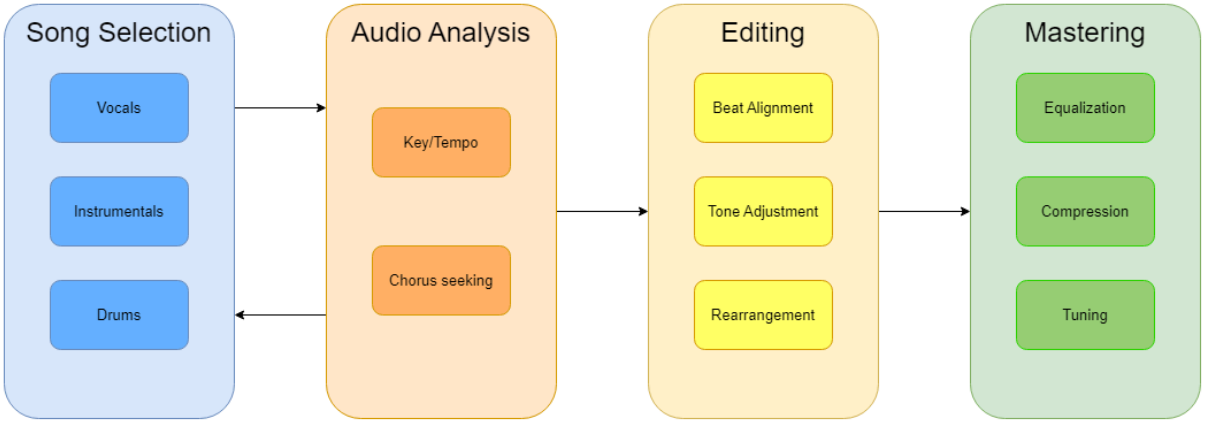


Figure 2. The workflow of a typical mashup procedure

The mashup artist first selects the source material, usually consisting of two or more popular songs. Careful consideration is given to the musical elements within each source, such as vocals, instrumentals, and rhythms, to determine how they can be harmoniously merged.

Next, the artist analyzes the *key*[6] and *tempo*[7] of each source to identify potential areas where synchronization and blending can occur. The tempo of a song describes how fast or slow the song is, and should match so that the vocals have roughly the same pacing as the instrumentals. The key tells us what chords[8] and notes are used, and should also match as the notes would clash and sound off otherwise.

Once the sources are chosen and analyzed, the artist begins the process of editing and rearranging the individual elements. This may include mixing different elements and carefully aligning beats, phrasing and musical cues to create a coherent and engaging composition.

The final step involves refining the mashup through post-production techniques such as equalization, compression, pitch adjustment and more. This enhances the overall sound quality to create a polished final product.

[6] The *key* describes the tonal center of a song. In the context of this report, the key can be “Major” or “Minor”, along with 12 different *keys* on the keyboard, for example “C Major” or “D minor”.
[7] The *tempo* of a song describes the speed of the song. If a *beat* is short, then the *tempo* is fast, and vice versa.
[8] The chord of a song is a label used in music theory to describe the harmony of a song at a particular point in time. Often, words like Major, Minor, Augmented, Diminished, Seventh and Suspended are used to describe the tone quality of the chord.

1.2 Challenges in creating mashups

The approach to musical mashups mentioned above faces two key difficulties:

- **Difficulty 1 - Finding candidate songs that sound nice together is not easy**

  Finding candidate songs to mashup is easily the most important part of the process. Without two songs which sound harmonious together, it is hard to create a pleasant mashup. However, given that a mashup artist cannot possibly expose themself to the virtually infinite amount of pop songs that exist nowadays, it can be difficult for a mashup artist to select good candidate songs for the mashup.

- **Difficulty 2 - A deep knowledge in music theory is necessary**

  During the editing and mastering process, it takes a deep level of music theory knowledge, as well as a long time, to produce a satisfactory result. A lack of technical know-how may result in dissonance, or produce a track with components too overwhelming for a pleasurable listening experience.

Our project aims to address these two difficulties in mashup creation by using a database to intelligently select compatible songs for use in a mashup, and using algorithms to mash songs to reduce the reliance on music theory knowledge. Our system automates these processes, making mixing songs more accessible to a wider audience. With user-friendly interfaces and intuitive controls, even those with limited technical expertise can engage in creating their own unique musical mashups.

1.3 Our project proposal

In light of the challenges faced by mashup artists, we propose our novel system which combines the successful results of previous researches on musical mashups into one seamless, automated experience.

Our system's architecture is as follows:

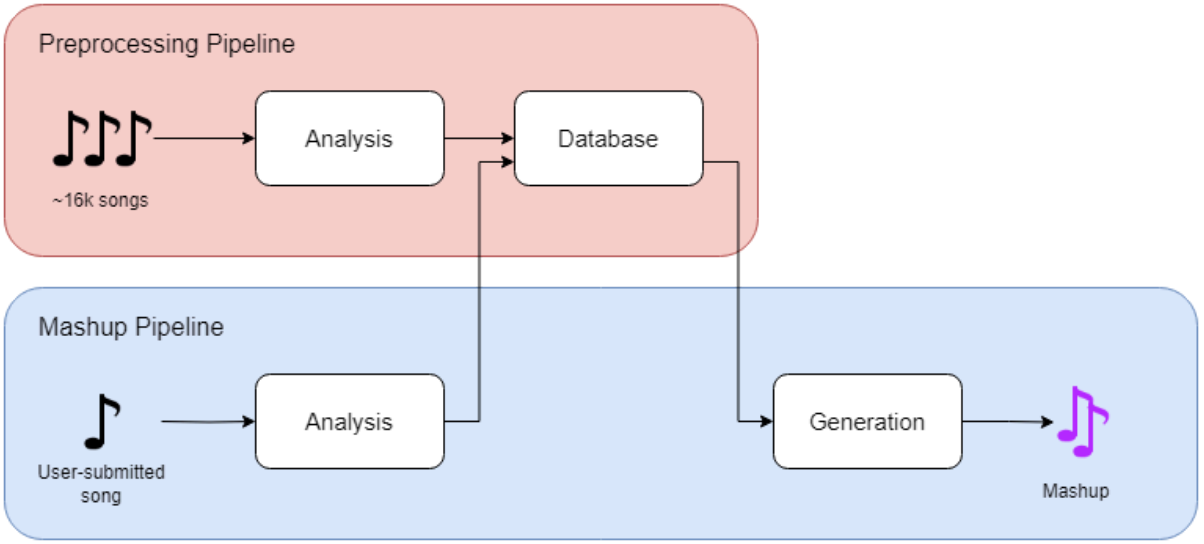


Figure 3. A high level overview of our system's architecture.
The system is split into 2 pipelines: the preprocessing pipeline and mashup pipeline

We also created a simple graphical user interface to interact with our system.

1.3.1 Preprocessing Pipeline

The goal of the preprocessing pipeline is to generate a large database of known pop songs, so that we can find the best possible songs to mashup. We collected a list of 16,082 popular pop songs, and for each song analyzed their harmony and rhythm. After that, these analyzed information are stored in our database for lookup later.

1.3.2 Mashup Pipeline

The mashup pipeline is where the mashup process occurs. The user first submits a YouTube URL of the song they wish to use in a mashup. The harmony, rhythm, and chorus is then analyzed in the same way that we analyzed each of the 16,082 songs in the preprocessing pipeline. After that, we compute the similarity between the user-inputted song and each song in the database to find the closest few songs. We then use the user-inputted song and one of the closest songs in the database to generate a mashup.

To simplify our task, we will only focus on mashing up 2 songs, because mashing three or more songs is a much more challenging task which does not necessarily produce a better output.

1.3.3 Frontend Graphical User Interface

To aid users in creating mashups easily, we designed a simple frontend graphical user interface (GUI) to communicate with the backend system.

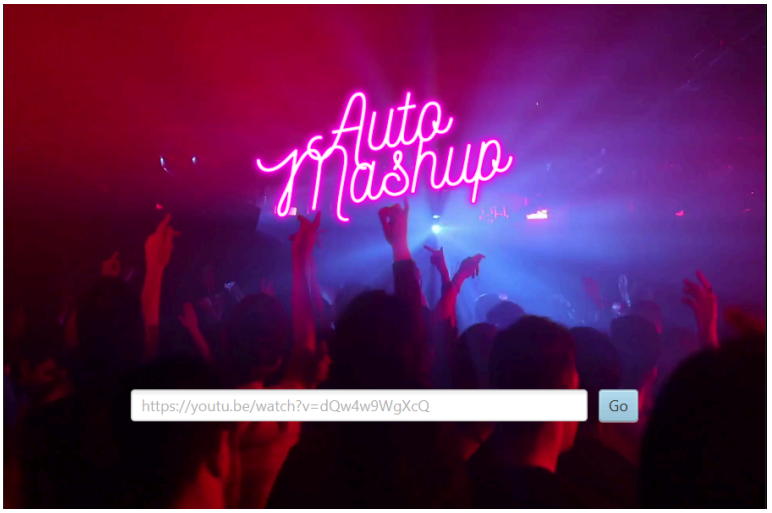


Figure 4: Our graphical user interface

Creating the graphical user interface allows us to abstract away our system's complex architecture, and enhances user interaction by providing visual elements like buttons, menus, and icons, making our system more intuitive and user-friendly. It simplifies the complicated procedure of creating a mashup, and enhances overall user experience.

1.4 Objectives

1.4.1 Goals

Throughout this project, our group achieved the following goals.

1. Design a complete automatic mashup system pipeline
2. Compile a comprehensive database of contemporary music for improved song-matching in mashups
3. Transform and adapt multiple trained signal and audio related machine learning models to the context of music creation
4. Utilize machine learning to aid the creation of artistic works
5. Gain hands-on experience in project team management and collaboration, using software that integrates multiple machine learning models and libraries

1.4.2 Outcomes

With reference to the goals above, we successfully achieved the following.

1. Creation of an automated mashup pipeline that can produce around 30-40 seconds of mashup music with pleasant quality
2. Compilation of a database containing the pre-analyzed information of over 1000 hours of audio
3. Design of our own algorithm capable of finding matching pop songs that produce a similarity score of over 80%[9].
4. Design and implementation of a graphical user interface that communicates with the aforementioned machine learning models

[9] Our experiment on a test set of 20 randomly selected songs showed that the system was capable of finding matching songs where the top 5 results achieved an average of 83.4% mashability. The mashability metrics used can be found in section 3.3.

1.5 Literature review

To gain a better understanding of algorithmic and/or machine learning approaches to automatic mashup systems, we conducted literature review of past successful systems on mashup systems as well as audio processing.

1.5.1 Prior works on automated mashup systems

There are a various number of ways automated mashup systems can be created. In preparing for our project, we took reference from a popular automated mashup system available on the internet, called Rave[10]. Rave provides an AI-incorporated method for users to produce a mashup between two different songs automatically. It does so by transposing song A and song B to a key closest to both songs, and matching the tempo of both songs by speeding up the slower song, and then interpolating the vocals and instrumentals of A in B.

Yet, from our observation, forcefully interpolating two songs with an incompatible structure often results in a poor mashup generated. The vocals may become distorted due to a speedup, creating an oversaturated atmosphere with conflicting musical patterns simultaneously present in the audio track. Some mashup songs generated by Rave such as the mashup between Hikaru Utada's *One Last Kiss* and Arianne Schreiber's *Komm, Süßer Tod* demonstrates the shortcomings of forcefully interpolating songs. Therefore, we aimed to offer an alternative approach to creating mashups by integrating the concept of a database query to identify similar songs.

1.5.2 Prior works on database creation

Mashup systems involving databases have existed and been proposed in the past, some being fully automated and some requiring fine user adjustments. For the database part, we drew inspiration from Horner's MashBash [2] to create a database of popular songs. In MashBash's proposal, they utilized a user-defined pop song list as their database, which also archived the mashups they created. This inspired us to consider approaches such as storing the analytical data of the songs in the database, such as the beat and chord detection results.

1.5.3 Prior works on database querying

When searching the database, we drew inspiration from Matthew et al's AutoMashUpper [3], which utilized a "mashability score" to find best-matching songs from our database. Their mashability score consisted of three parts: Harmonic

[10] Rave.dj can be accessed at https://rave.dj/.

compatibility, rhythmic compatibility, and spectral balance. This inspired us to use the song's harmony, rhythm, and spectrogram in designing our own algorithm for finding the best songs to mash.

1.5.4 Prior works on mashup music generation

Various proposals have been made by researchers for determining the best ways to mash up two songs. Baixi et al's PopMash [4] model used lyrical agreement to produce better transitions between segments of the two songs. This involved analyzing the lyrics of the songs to determine the best timestamps to transition between the two songs. Xinyang's Rhythm Fusion model [5] focused on mashing up songs by swapping the drums of the songs. These inspired us to consider the possibility of improving on our mashup algorithm by first segmenting the song into vocals, drums, instrumental and bass, then mixing and matching different components of the two songs.

Section 2: Our Architecture in Detail - Preprocessing Pipeline

2.1 Overview of preprocessing pipeline

The goal of the preprocessing pipeline is to construct a large database of contemporary pop songs across a wide range of genres and languages, and pre-analyze their information so that our system can find better mashup matches more easily, thereby solving the issue in traditional mashups in which mashup artists need to expose themselves to a large number of songs.

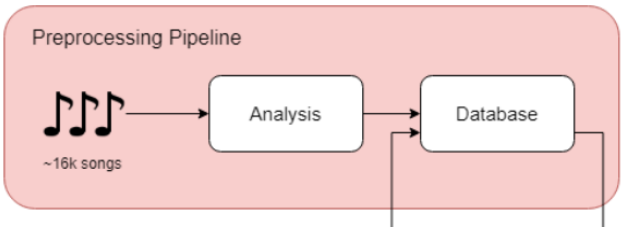


Figure 5. A high-level overview of the preprocessing pipeline

We achieve this by first compiling a list of 16,082 popular pop songs, then analyzing their harmony *chord progression*[11] (harmony analysis) and beat locations (rhythm analysis), and storing the analyzed data in a database.

The following diagram shows a high-level overview of how we compiled such a list of songs.

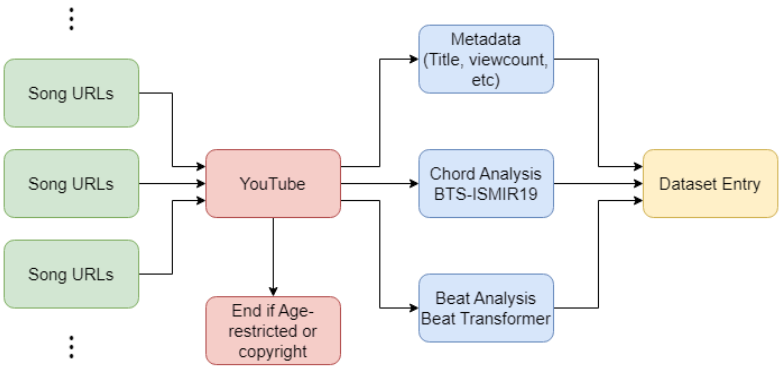


Figure 6: How we collected the list of 16,082 songs

[11] A *chord* progression refers to a sequence of chords which describes the way that the harmony of the song evolves over time.

We first manually collected the songs in the form of YouTube playlist links. Then, the playlist contents are extracted to video URLs and determined if they contain any age-restricted or copyright-restricted content. This minimizes the chances of violating any copyright-related laws. We then took the audio of each video and analyzed them using the following harmony analysis model and a rhythm analysis model.

This section will go into detail about the exact analysis we have conducted on each song.

2.2 Harmony and rhythm analysis

2.2.1 Harmony analysis

Based on our testing, we have settled on a deep-learning based approach for harmony analysis.

Design Justification - Why Deep Learning?

The harmony of a song, although quantifiable, can be subjective and difficult to determine by algorithmic approaches. In practice, composers often employ musical ornamental techniques to make the song more "interesting" without necessarily changing the underlying chord. These techniques include passing notes, suspensions, and grace notes. Furthermore, the overtone series introduces secondary harmonies that are difficult to account for, diminishing the reliability of pure-algorithmic approaches. Our experiments show that AI-based approaches are more versatile and work better than the algorithmic approaches.

The BTC-ISMIR model is a transformer-based neural network developed by Park and Choi for the International Society for Music Information Retrieval [6]. It takes in the spectrogram[12] of a song, and outputs a list of chords and their timestamps.

[12] The *spectrogram* of a song is a visual encoding of the song's energy level at different pitches. It allows us to visualize the frequency distribution of a song over time, as well as the energy levels of each frequency.

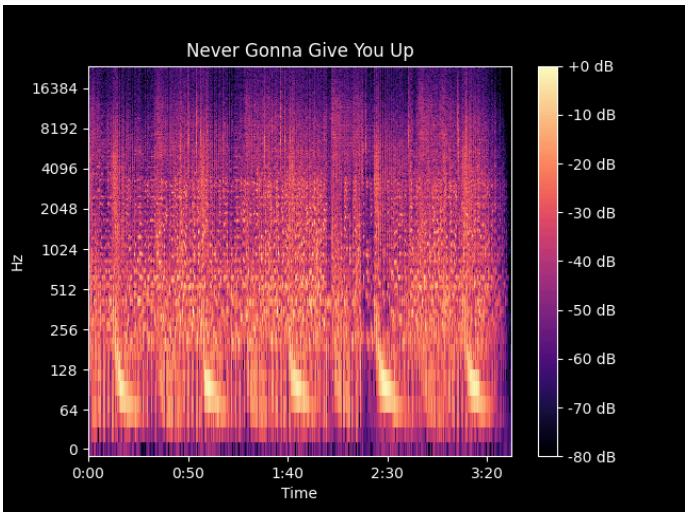


Figure 7. The spectrogram of the song *Never Gonna Give You Up* by Rick Astley. Bright regions represent a higher energy level.

The chord information provides quantized harmony information about the song, which serves as a crucial metric to determine the harmonic compatibility of two songs, directly influencing the quality of the final mashup.

The BTC-ISMIR19 model introduces a self-attention mechanism within a bi-directional Transformer architecture for chord recognition. The use of this attention mechanism allows the model to focus on specific regions of chords, addressing limitations in capturing long-term dependencies seen in previous models. Importantly, BTC-ISMIR19 demonstrates competitive performance with a single-phase training process, distinguishing itself from traditional approaches that rely on separate feature extractors or additional decoders.

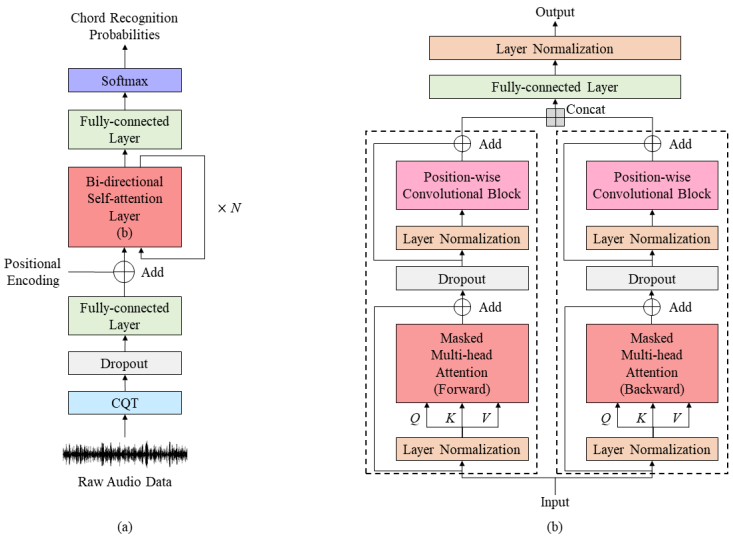


Figure 8. The architecture of the BTC-ISMIR model for chord recognition, developed by Park and Choi

The output of the BTC-ISMIR model is a sequence of chord labels. There are 170 possible chord labels, with 168 chord quality labels as well as an “Unknown chord” label

and a “No chord” label. The chord quality labels are classified as one of the 14 classes (see table below) multiplied by 12 keys of the traditional *12-tone equal temperament*[13] of Western music.

Chord Name	Notes (in the key of C)
Major	C, E, G
Minor	C, Eb, G
Augmented	C, E, G#
Diminished	C, E, Gb
Major 6th	C, E, A
Minor 6th	C, Eb, A
Dominant 7th	C, E, G, Bb
Major 7th	C, E, G, B
Minor 7th	C, Eb, G, Bb
Diminished 7th	C, Eb, Gb, Bbb (≅A)[14]
Half-diminished 7th	C, Eb, Gb, Bb
Minor-major 7th	C, Eb, G, B
Suspended 2nd	C, D, G
Suspended 4th	C, F, G

In practice, however, we found that out of the 1000 hours of audio we have collected, the database did not contain any minor-major 7th labels[15]. One possible reason for this could be due to a bias in the BTC-ISMIR model during training, where the training data did not contain many minor-major 7th labels. However, since the minor-major 7th is

[13] 12-tone equal temperament: The music system commonly used by the west since the Classical Period (c.a. 18th century). An octave consists of 12 tones, and each pair of consecutive notes maintains a ratio of 12th root of 2.

[14] Remark: The note Bbb (B double-flat) is *harmonically equivalent* to the note A. Although they have the exact same sound up to octaves, they are not theoretically equivalent.

[15] The minor major 7th label (in C) describes the harmony formed by C, Eb, G, and B. This type of harmony is often depicted as mysterious and somewhat spooky, and is more commonly seen in game or film music with this exact purpose, such as Dummy! in Undertale, the overworld theme of Final Fantasy 7, and some versions of the James Bond theme.

quite an exotic type of harmony to use in music, it is rarely found in pop songs, and it is also possible that our collection did not contain any minor-major 7th chords.

For the rest of the 158 labels, we counted the duration of music that had each of these labels, and found a surprisingly almost-exponential relationship between them. The **log**-plot of the duration of the music against each label is shown below.

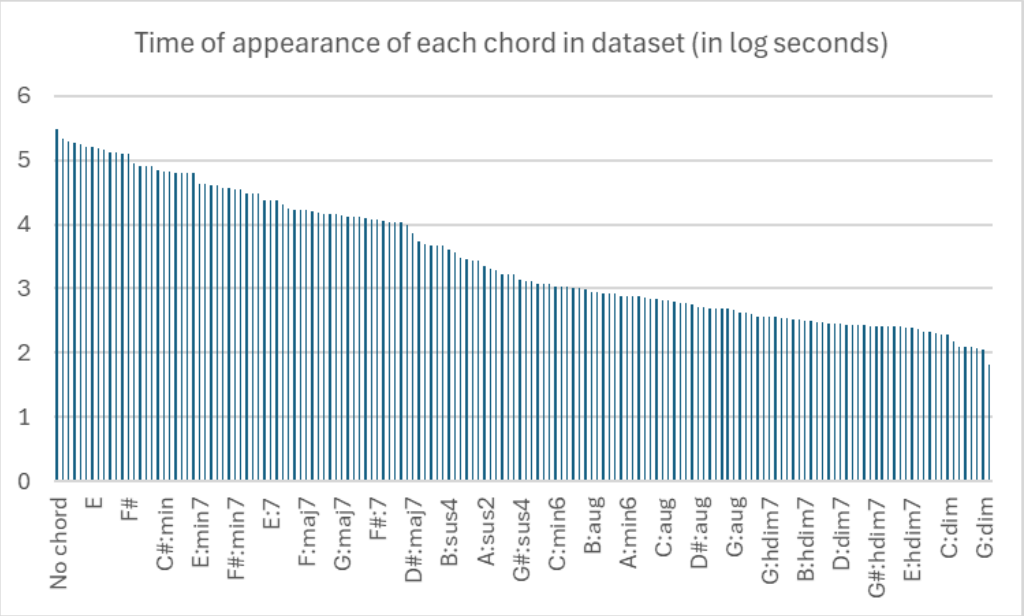


Figure 9. An exponential relationship between the different labels. The *y*-axis is in log-scale.

2.2.2 Rhythm analysis

The goal of the rhythm analysis section is to obtain the timestamps of where the *beats* and *downbeats*[16] of the song occurs.

Design Justification - Why do we need a rhythm analysis model?

In music theory, beats and downbeats are units of time quantization that gives the rhythmic structure of a piece of music. This is paramount because it will allow us to have concrete markers to match songs with different tempo together, and also handles cases where a song may change its tempo throughout the song.

We chose to use a beat transformer model [7] for this task. The beat transformer model presents a novel approach to beat and downbeat tracking, incorporating demixed spectrograms with multiple instrument channels. Unlike conventional models that focus solely on the spectrogram of an audio mixture, the beat transformer

[16] A downbeat is the beat that indicates the beginning of a measure (or bar). Intuitively, it can be thought of as the beats to which the audience would clap to in a concert.

acknowledges the importance of richer musical contexts, such as chord progression and instrumentation, in human perception of metrical structures.

The transformer architecture of the model introduces both timewise and instrument-wise attention mechanisms, aiming to capture intricate metrical cues embedded in the unmixed spectrograms. This dual attention approach reflects the complexity of musical contexts, allowing the model to discern deep-buried metrical features and produce accurate results.

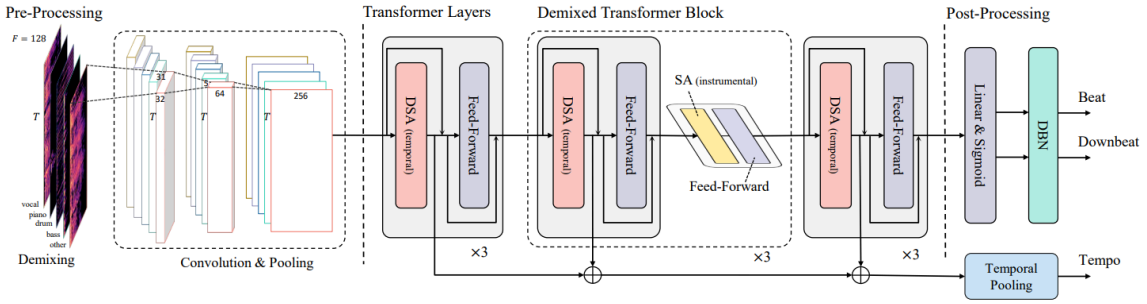


Figure 10. The architecture of the beat transformer model.

Implementation Detail

The beat transformer in the official repository uses Spleeter for audio source separation, which has been shown to be less accurate (refer to section 3.2.1). We have adapted the model with some workarounds so that it is compatible with Demucs. In particular, Spleeter is a 5-stem model with an additional piano part as its output. In our implementation, we assumed the piano part was empty and produced the other four parts using Demucs. We found the results to be almost identical.

We tested the result of the beat transformer on various songs, and it produced promising results. The graph below shows our testing of the beat transformer on the intro of the song *Racing into the Night* by Yoasobi.

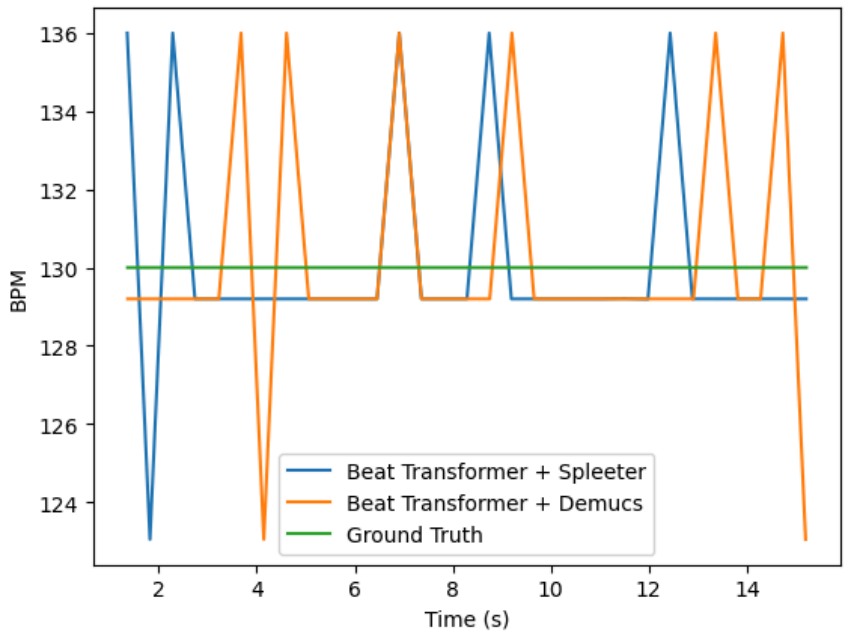


Figure 11: The detected BPM of the intro section of Yoasobi's "Racing into the Night". The Ground Truth of 130 beats per minute is human-labeled with the help of the music production software FL Studios.

YOASOBI「夜に駆ける」Official Music Video

The ground truth *beats-per-minute (BPM)*[17] was 130. The detected difference of 6 bpm lasted for only approximately 0.02 seconds, and since the fluctuations offset each other, the resulting *beats*-per-minute detected by the beat transformer proved to be fairly accurate.

2.3 Database

The harmony and rhythm analysis results of all 16,082 songs are stored in our database. The database contains the following columns:

- **Chords and Chord times**
  A list of symbols and timestamps indicating the harmony of the song at a particular point. This information is calculated using the BTC-ISMIR model described in section 2.2.1.
- **Beat times and downbeat times**
  Two lists of timestamps indicating the rhythm of the song. This piece of information is created using the beat transformer described in section 2.2.2.
- **Title**
  The title of the song.

[17] Apart from musical vocabularies like *Andante*, and *Allegro*, *beat-per-minutes (bpm)* is also a common indicator of a tempo of a song, especially for pop-songs. For example, Rick Astley's *Never Gonna Give You Up* has a BPM value of 113.

- **Length**
  The duration of the audio. We only collected songs between 2-10 minutes. This is a rudimentary measure to filter out song snippets, segments, playlists or medleys.
- **Genre**
  The genre of the song. In our database, each song is marked with one of the following 13 genre labels:

pop	nightcore[18]	jp-anime	kpop	broadway[19]
country	jp-pop	cantopop	rock	folk
hip-hop	vocaloid[20]	reggaeton[21]		

- **URL**
  The URL of the YouTube video
- **Views**
  The view count of the video at the time of data collection. We only collected songs with more than 500,000 views.

The resulting database consisted of over 1000 hours of analyzed audio. Notice that we did not store the actual audio inside the database. We did this for two reasons

(1) To avoid violating any copyright related laws such as the DMCA, and
(2) Storing 1000 hours of audio takes up a large amount of unnecessary disk space.

[18] Nightcore is a genre of music characterized by speeding up and often pitch-shifting existing songs, usually from the electronic dance or pop genres. The resulting tracks typically have a faster tempo and higher pitch, giving them a more energetic and upbeat feel.
[19] Broadway, or musical theater, is a distinct genre of music characterized by theatrical productions that combine music, lyrics, and dance. It often features catchy melodies, elaborate staging, and storytelling through songs.
[20] Vocaloid is a modern genre of synthesized music created using vocaloid softwares, enabling users to produce pop songs with virtual singers. One of the most well-known vocaloid virtual signers is Hatsune Miku, released by Crypton Future Media on August 31, 2007.
[21] Reggaeton is a modern genre of electronic music, emerging from Panama in the late 1980s. It blends elements of hip hop, Latin American, and Caribbean music, and features vocals that often include a mix of rapping and singing in Spanish.

The graphs below show some statistics of the songs we collected in our database.

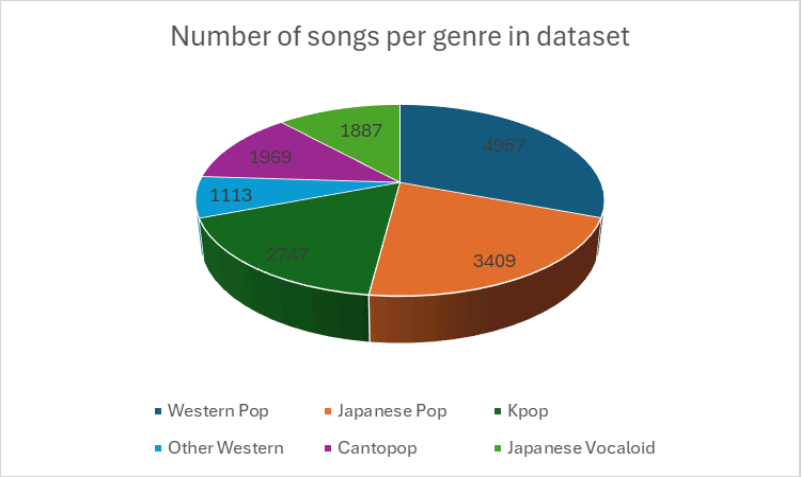


Figure 12: The number of songs per genre present in our database

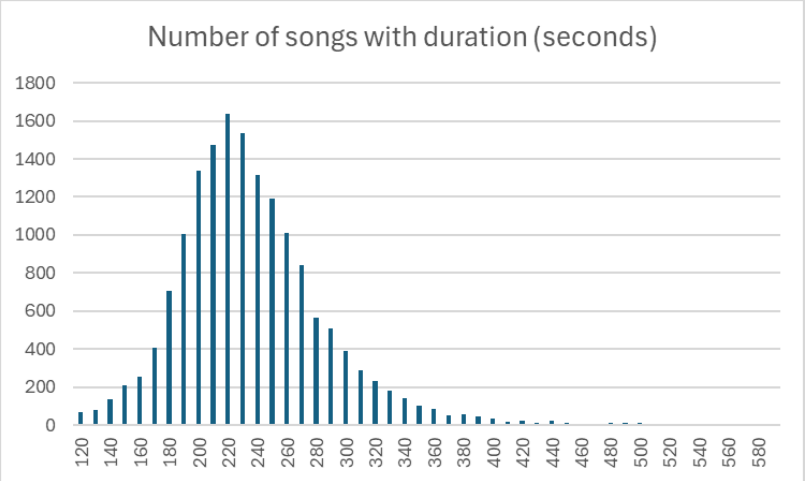


Figure 13: The distribution of the duration of songs in our database

Section 3 - Our Architecture in Detail - Mashup Pipeline

3.1 Overview of the mashup pipeline

The mashup pipeline forms the core data flow of the system, combining the user-facing part of the system with our internal AI models and algorithms.

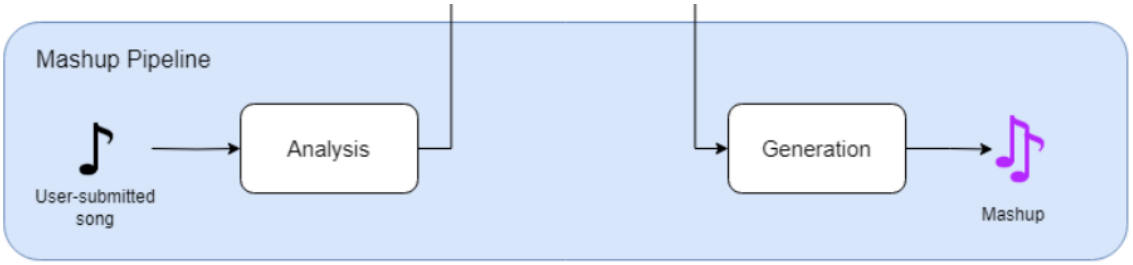


Figure 14. A high-level overview of the mashup pipeline

The goal of the mashup pipeline is to take a song submitted by the user, and by utilizing the analysis results in the database, find suitable material to generate a coherent mashup. This section will go into detail about how we analyzed each song, queried the database, and generated the final mashup result.

3.2 Analysis

3.2.1 Audio separation

Nowadays, most songs contain the following 4 tracks: drums, bass, vocals, and instrumentals. The goal of audio separation is to separate any given song into the four tracks mentioned above, so that we can analyze and manipulate the different tracks separately. Using *Demucs*[22], we were above to achieve this.

The first step of the mashup is thus to put the user-submitted audio through Demucs to get the separated audio components. This will facilitate the subsequent analysis steps and mashup creation in the end.

[22] The Demucs project is a transformer-based deep learning model developed by Defossez [8] which employs a hybrid separation technique using both waveform and spectrogram analysis. The goal of Demucs is to take in a song, and return the following four separate audio tracks: drums, bass, vocals, and instrumentals.

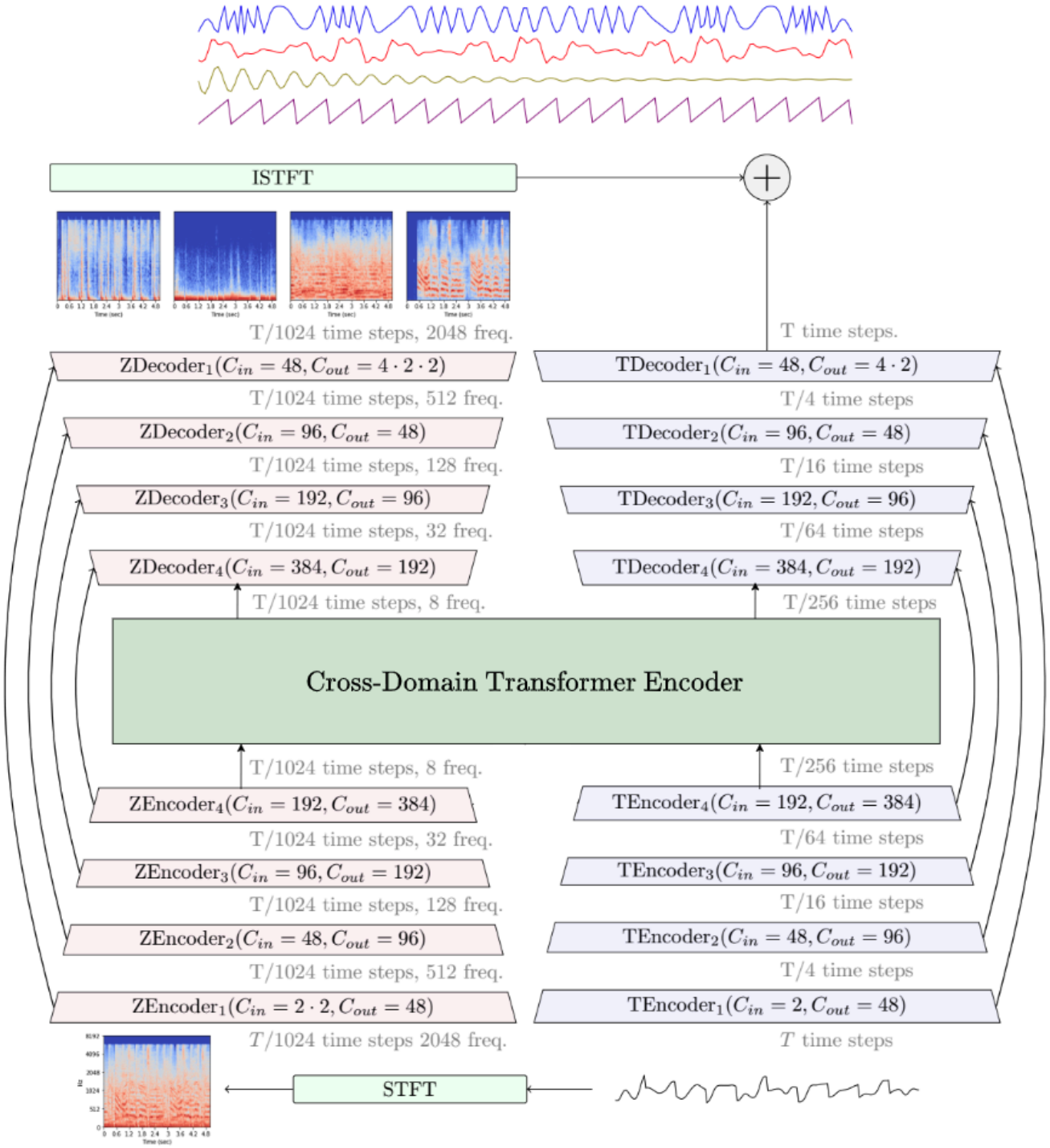


Figure 15: The architecture of Demucs, a wave-spectrogram hybrid transformer for audio source separation.

Model	Domain	Extra data?	Overall SDR	MOS Quality	MOS Contamination
Wave-U-Net	waveform	no	3.2	-	-
Open-Unmix	spectrogram	no	5.3	-	-
D3Net	spectrogram	no	6.0	-	-
Conv-Tasnet	waveform	no	5.7	-	
Demucs (v2)	waveform	no	6.3	2.37	2.36
ResUNetDecouple+	spectrogram	no	6.7	-	-
KUIELAB-MDX-Net	hybrid	no	7.5	**2.86**	2.55
Band-Spit RNN	spectrogram	no	**8.2**	-	-
Hybrid Demucs (v3)	hybrid	no	7.7	**2.83**	**3.04**
MMDenseLSTM	spectrogram	804 songs	6.0	-	-
D3Net	spectrogram	1.5k songs	6.7	-	-
Spleeter	spectrogram	25k songs	5.9	-	-
Band-Spit RNN	spectrogram	1.7k (mixes only)	**9.0**	-	-
HT Demucs f.t. (v4)	hybrid	800 songs	**9.0**	-	-

Figure 16: Demucs shows state-of-the-art results compared to popular models such as Spleeter

3.2.2 Temporal segmentation

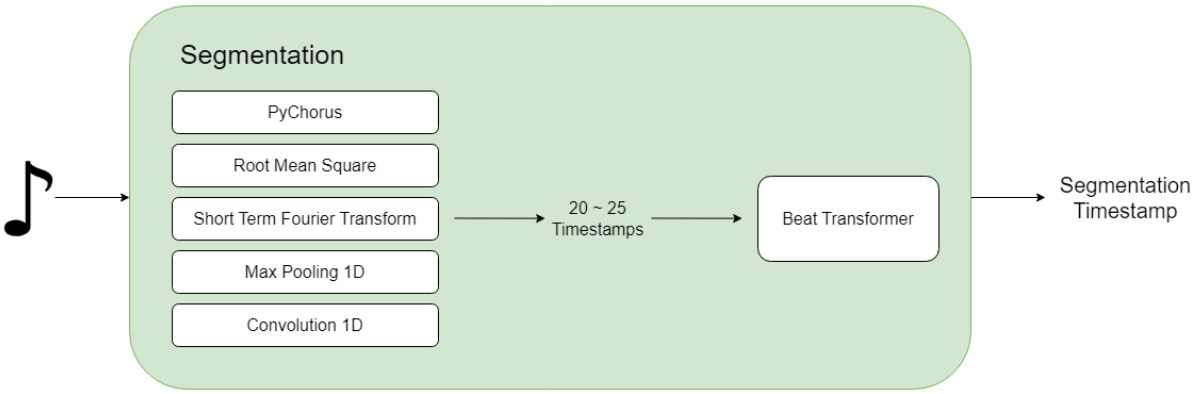


Figure 17: The workflow of segmenting a song

For any given song, the goal of time segmentation is to give an approximation of where the chorus of the song is. We do this by first (1) finding the starting time of the chorus in the song, then (2) determining the length of the chorus by considering the song's tempo under an algorithmic rule with respect to the song's spectrogram.

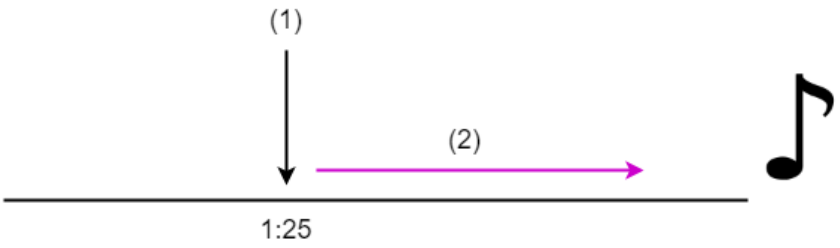


Figure 18: Segmenting a song by (1) finding the starting time of the chorus, and (2) determining the length

Based on our testing results, no single method accurately returns the starting time of the chorus, with reasons to be specified below. Thus we have chosen to use an ensemble method to determine the starting time. The strength of using an ensemble method is that the algorithms can make up for each others' weaknesses. After testing, we have settled on using the following 5 different algorithms, then aggregating the results of each algorithm into one final chorus entry timestamp.

- **PyChorus self-similarity test**

  The motivation behind PyChorus's implementation is that the chorus should repeat a few times throughout the song. Thus if we find that a certain section of the song appears repeatedly throughout the song, then it is most likely the chorus.

  The exact technique to determine which verses appear repeatedly is proposed by M. Goto in the paper "*A detection method for musical audio signals and its application to a music listening station*" [9]. Using the spectrogram and approximated tempo of the song (obtained from the beat analysis part), the algorithm compares the chroma features of the song throughout the entire song, then assigns an *overlapping score*[23] to each 8-bar segment using the Support Vector Machine algorithm. The higher the segment's score, the more likely the segment is a chorus. PyChorus will then output the timestamps of the beginning of all predicted chorus segments that do not coincide with one another. In essence, PyChorus is a self-similarity test.

  The downside to this approach is for some songs, it has a chance to split the song in the middle of a sentence. Furthermore, some songs with repeating verses that are not choruses are returned by PyChorus. For example, when running PyChorus on the song *Never Gonna Give You Up* by Rick Astley, we obtained the following output: (Chorus highlighted in red)

[23] In implementation, the *overlapping score* is equivalent to indicating how many segments of the song share the identical chroma features up to a soft margin.

I just wanna tell you how I'm feeling

Gotta make you understand

Never gonna give you up

...

We've known each other for so long

Your heart's been aching, but you're too shy to say it (to say it)

Inside, we both know what's been going on (going on)

We know the game and we're gonna play it

This motivated us to experiment with other methods.

- **Root-Mean-Square (RMS) volume test**

  The motivation behind this method came from the idea that during transitioning into the chorus of a song, the volume of the song should increase. Thus if we can determine where the changes in "loudness" occurs, then we would have found the chorus.

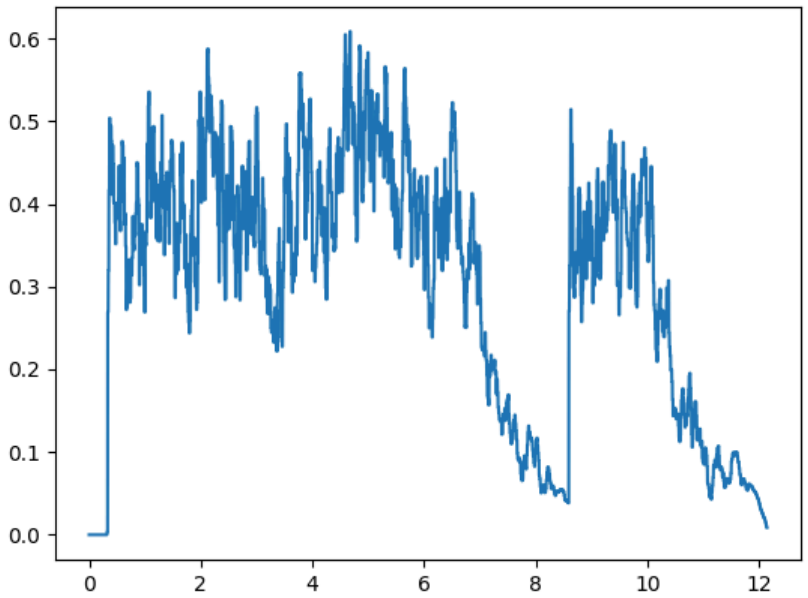


Figure 19: There is a sudden increase in volume from around *t=8.5*, indicating a possible entry to chorus

The first hurdle is that we, as humans, detect an entry into the chorus by an increase in the loudness of the vocals but not the instrumentals. We would first need to isolate the vocals of the song. This is done using the Demucs library from the previous section. After isolating the vocals, we transform the audio waveform to a measure of "loudness" by computing the root-mean-square (RMS) of the original audio waveform, using a hop length of *H=4096*. The resulting

waveform would then be non-negative. After that, we compute the successive differences between the loudnesses at times *t* and *t-1*, and find the top few timestamps with the largest differences.

Mathematically, if ω denotes the original audio waveform with respect to the sampling rate, then the root-mean-square of ω with a hop length of *H=4096* is found by

$$\Gamma(s) = \sum_{t=1}^{D} \left(\frac{1}{H} \int_{[(t-1)H, tH]} \omega^2 \, d\mu \right)^{\frac{1}{2}} \chi_{[(t-1)H, tH]}(s)$$

The required timestamps can then be found by the timestamps which maximizes the derivative of gamma:

$$\operatorname{argmax} \left\{ \frac{d}{ds} \Gamma(s) : s \in [0, N] \right\}$$

For most songs, this algorithm produces a reasonably accurate result. However, there are some songs that do not increase their loudness immediately upon entering the chorus. For example, running this algorithm on *Lemon* by Kenshi Yonezu gives the following segmentations:

ありはしないとわかっている
あの日の悲しみさえ
あの日の苦しみさえ

Running the algorithm on *Never Gonna Give You Up* by Rick Astley gives the following:

Gotta make you understand
Never gonna give you up
Never gonna let you down

We worked around this problem by fading into the chorus 2 seconds before the timestamp returned by this algorithm.

- **Convolution 1D edge detection test**

  The motivation behind this test is similar to the previous test, in which we want to find the most drastic increase in volume throughout the song. However, unlike the previous method which uses the successive differences between the timestamps to determine the changes in volume, here we pass an edge detection convolution filter to detect such changes in volume.

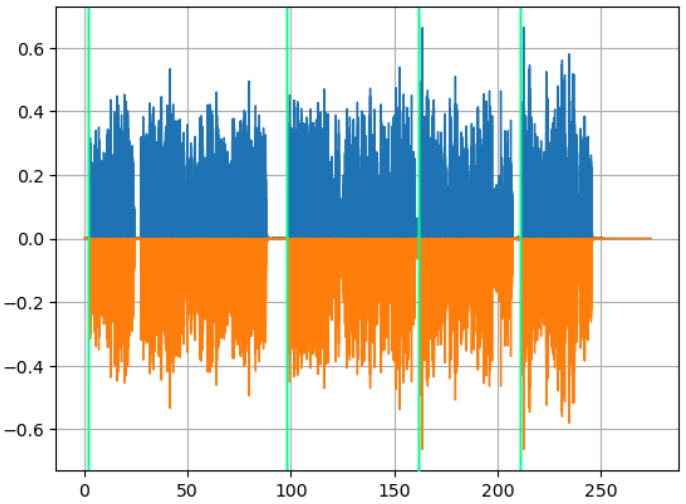


Figure 20: Conv1D left edge detection applied to *Lemon*, by Kenshi Yonezu. Green bars: edges detected

Given the original audio signal, we first extracted the vocals of the song, took the absolute value of the vocals, then passed an edge detection convolution kernel through the audio to determine all edges. After that, we filtered out all edges corresponding to a decrease in volume, which results in the left edges remaining. The audio signal above shows the original song's audio waveform, together with the edges detected, shown in green.

- **Max-Pooling 1D energy test**

  The idea behind the max-pooling energy test is that the chorus of a song should be the "loudest" parts of the song. Thus by finding verses of the song which are in general louder than its neighbors, we are able to determine the chorus.

  We first extract the vocals of the song, then take the absolute value of the vocals, and run a max pooling kernel through the audio signal. Based on the returned signal, we compute the successive differences between timestamps *t-1* and *t* to find the largest increases.

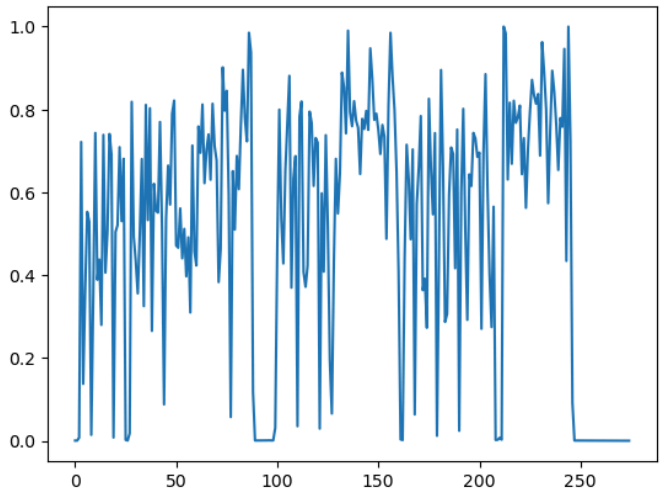


Figure 21: Max-Pooling 1D applied to Lemon, by Kenshi Yonezu

The advantage of this method is that with a large enough kernel size, max pooling can ignore the small “ups and downs” in the audio signal, so that only the verses with a consistent large energy level are detected.

- **Short Time Fourier Transform energy test**

  The idea of the Short Time Fourier Transform (STFT) energy test is similar to the previous algorithm, in that we want to find the verses of the song with the strongest energy. The difference is that instead of using max pooling, we use the spectrogram of the song. STFT produces this spectrogram. This allows for a more fine-grained analysis about the energy levels of the music, giving a more accurate analysis.

  For example, the *spectrogram* of the song *Jane Birkin* by MIKA is shown below.

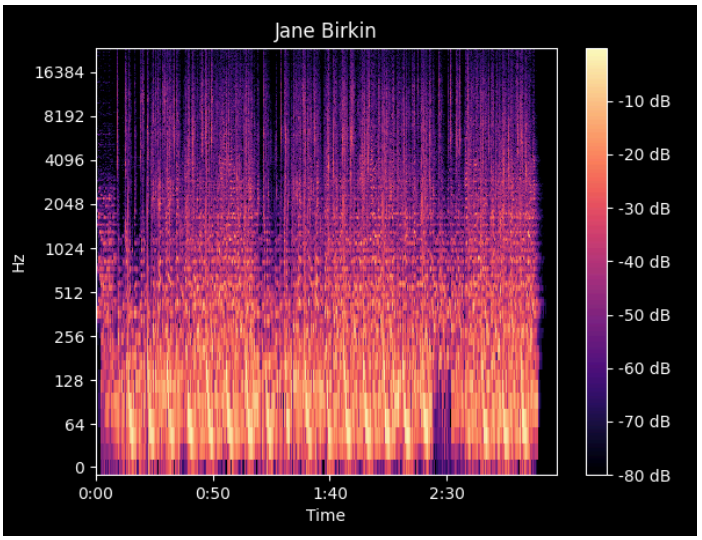


Figure 22: Spectrogram of *Jane Birkin*, by MIKA.

After retrieving all the timestamps returned by the 5 algorithms above, we align each of the timestamps to the closest downbeat using the beat detection result in Section 3.2.1. This will give us around 20~25 downbeats in total (roughly 4-5 timestamps returned by each algorithm). The downbeat that occurs with the highest frequency will then be the timestamp representing a transition into the chorus.

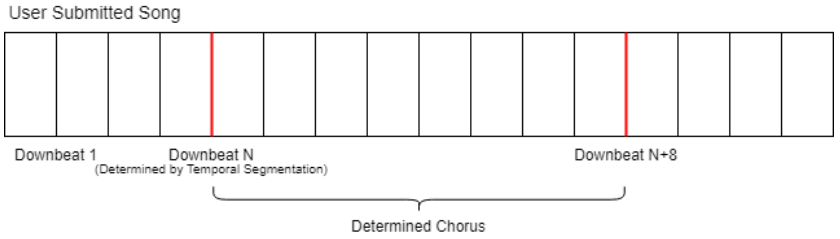


Figure 23: The final temporal segmentation result

3.3 Querying database

In the database query step, we take the chorus of the user-submitted song and find the best segment in the database to mashup. Recall that in section 1 we have explained how the harmony and rhythm of the song serves as critical metrics for determining the similarity of two songs; we will use the chord analysis results and beat analysis results from section 2.2.1 and 2.2.2 to perform a similarity analysis test. The core algorithm of this pipeline is thus a way to calculate the similarity score of two songs given their chords and beats.

It is helpful to view the chord analysis results, beat analysis results, and the audio itself as a time series - an object with a duration D that defines a value at every point in time between $t=0$ and $t=D$. For example, the audio can be viewed as a time series where at time t the audio takes the value as the amplitude of the underlying sound wave. Similarly, the downbeat results can be regarded as a time series, where at time t it takes the value as the bar number of the audio, or in other words the number of downbeats that occurred on or before time t. The chord analysis results can be regarded as a time series where at time t the value is one of the 170 chord labels above based on the harmony of the song at that time. Then, it makes sense to speed up/slow down, splice, and join beat and chord results in the same way we performed these operations on their corresponding audio.

A simple example would be as follows: Suppose a song plays the “C” chord from $t=0$ to $t=1$ seconds, and “D” from $t=1$ to $t=3$ seconds. If we speed up this chord analysis result by 2 times, it would be playing the “C” chord from $t=0$ to $t=0.5$ second, and “D” from $t=0.5$ to $t=1.5$ seconds.

The intuition for the similarity metric is as follows: Suppose we have extracted 8 bars of music using the chorus analysis result and beat analysis result. For every 8-bar song

segment of every song in the database, we can make speed adjustments to every bar of the sample from the database to align its rhythm with the user's submitted song segment. They will now have the same duration and thus can be compared at every point of time. We make these speed adjustments on the sample song segment's corresponding chord result, and compute the distance of two chords (the smaller the distance, the closer the two chords, and hence the more harmonious they should sound together) at every point multiplied by the duration. This gives a similarity score between two song segments, where its physical meaning is the weighted sum of the duration of the inharmonious portions if we were to mash these two songs together.

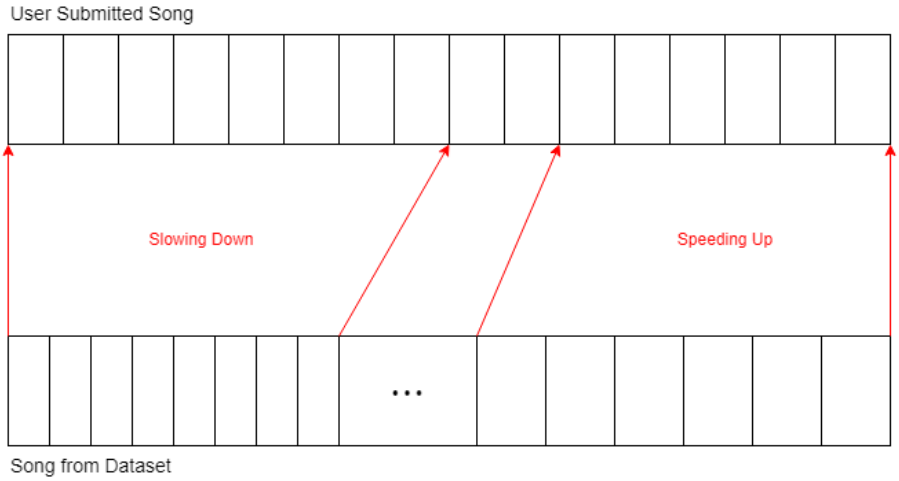


Figure 24: A visual demonstration of the speed adjustment step applied to the songs in the dataset

To be precise, we define the distance between two chords to be the number of notes that they are different, or more accurately the symmetric difference of the set of notes that comprises the chords. The chords "C Major" (with notes "C", "E", and "G") and "E minor" (with notes "E", "G", and "B"), for instance, would have a distance of 2, because the notes "C" and "B" were in one of the chords but not the other.

Mathematically, we define the distance formula as follows: for the chord results $C_1(t)$ consisting of a set of notes at every point of time, and the speed-adjusted chord results $C_2(t)$, where $C_1(t)$ and $C_2(t)$ have a common duration D, they would have a distance given by the following formula:

$$\operatorname{dist}\big(C_1(t), C_2(t)\big) = \int_{[0,D]} |C_1 \triangle C_2| \, d\mu$$

Finally, the mashability score between two songs would then be defined as

$$\operatorname{Mashability}\big(C_1(t), C_2(t)\big) = 100e^{-0.05 \cdot \operatorname{dist}\left(C_1(t), C_2(t)\right)}$$

Design justification - Why did we come up with a mashability scoring function?

Interpreting the similarity metric is not easy, as to do so one would need to understand all the implementation details. We wanted to come up with an intuitive measure of how mashable the two songs are. To achieve this, we decided to curve the score returned by the similarity metric so that we have an output in the range 0 to 100, where 100 represents a perfect match. In practice, to have a mashability score of 80% or above, the distance metric would need to return a value lower than 4.5. Thus having a 80% match is already quite hard to come by!

Based on our testing feedback, this similarity metric had two areas that could be improved on.

The metric might find songs with sparse harmony, for example rap music or the silent portions during the end credits of a music video, where a substantial amount of the song has the "No chord" label.

As this would not produce a good mashup result, we filtered out segments with more than 20% of its duration having no-chord segments, and made some further adjustments on top of the core search algorithm.

The metric might return matches that sometimes do not mash well. This is due to us speeding up and slowing down the audio of some certain songs too much, resulting in false matches.

To solve this, we restricted our search to songs where the beat alignment process does not speed up the audio more than 125% or slow down the audio less than 80%. Doing so avoided introducing undesired noises in the final mashup result. On the other hand, we expand our search to songs in the database that are pitched up or down 3 semitones so that there is more flexibility with the final mashup.

3.4 Generation

The first part of the pipeline - the database query algorithm - returns a list of song segments that are similar to a specific segment of the user-submitted song (determined by the chorus extraction). We will document the second part of the pipeline here, which involves creating the necessary auditory components for mashup, and using various audio processing techniques to combine and master a decent-sounding mashup result.

3.4.1 Component creation

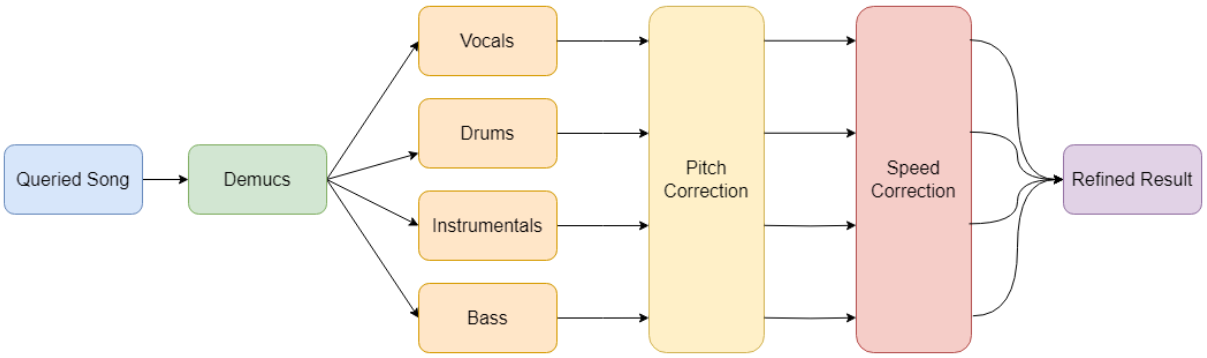


Figure 25: A broad overview of mashup algorithms after the database query step

This section of the pipeline creates the components for the final mashup, which includes the separated tracks (vocals, drums, bass, and instrumentals) of both the user-submitted song and the queried song from the database. Also, recall in section 3.3, we implemented some tricks to expand our search radius to songs that could vary both in speed and pitch. Thus we need to make speed and pitch adjustments to the queried song to make the result sound coherent.

In the current pipeline, the user is prompted to choose their desired piece to proceed with the mashup. When the piece is chosen, its corresponding URL will be used to first fetch the song from YouTube. The audio will then be separated into vocals and other tracks using the Demucs algorithm, as documented in section 3.2.1. Using the corresponding beat analysis information from the database, we will calculate the factors to speed up or slow down different segments of each of the audio tracks. Finally, we will alter the pitch of the separated audio tracks to match that of the user-submitted song. Notice we did not apply Demucs after the speed and pitch alteration, but instead chose the resource-heavy way of repeating the speed and pitch changes on each of the

separated audio tracks; we found these alterations sometimes affected the output quality of Demucs negatively.

These components can be used in subsequent parts of the pipeline, where we can implement different types of mashup algorithms.

3.4.2 Mashup algorithms

We came up with 3 algorithms for mashing up songs.

1. **These two songs have the same key**

   The most basic algorithm we decided to focus on is a simple vocal swap, nicknamed “These two songs have the same key” algorithm, as demonstrated by a simple type of mashup on song segments on YouTube that aimed to demonstrate the similarity of two songs.

Figure 26: An example of “These two song have the same key”: Polyphia - ABC and Renai Circulation

These two songs have the same key

   The output of this algorithm will be a 16-bar song segment. The first 8 bars will be the original (user-submitted) song, and the second 8 bars will be a mashup between the backing track (drums + bass + instrumentals) of the user-submitted song and the vocals of the queried song.

   With the mashup components created in 3.4.1, let’s suppose the segment used to query the database is bars *N* to *N+8* of the user’s submitted song. First, we use a high-pass filter on the vocals of the queried song - we found that it improves the quality of the audio and removes a low-frequency “muffle” artifact that was present in many of the outputs by Demucs. Then, we adjust the stereo

pan of the audio components. This makes the final product sound more spacious and the audio tracks less clashing. Based on our testing, we found the following values to work well on many mashups.

Component	Stereo Pan left
Vocals	0.15
Bass	-0.15
Drums	0.3
Instrumentals	-0.3

We then adjust the volume of each component. To detect the average volume, we reused the RMS algorithm in temporal segmentation as mentioned in section 3.2.3. We decided to adjust the volume of the vocals to be 1.1x the other components. Finally, we take the separated audio tracks of the user submitted song, extracting bars *N-8* to *N+1*, making the same volume and stereo pan adjustments, and join these two audios together. The audio between bars *N* to *N+1* is used to introduce a cross-fade effect to enhance the “wow factor” of the mashup.

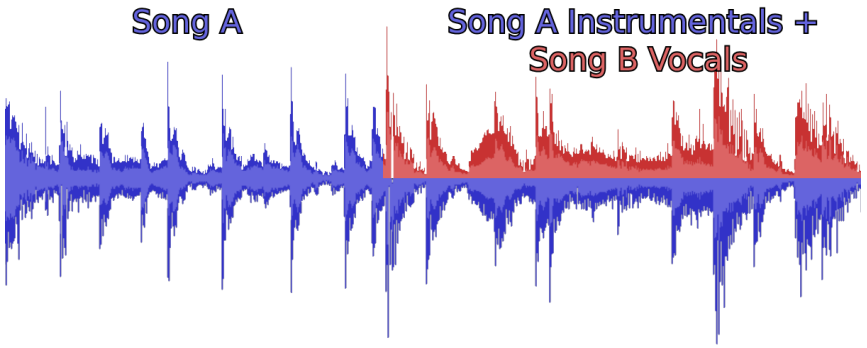


Figure 27: Sound wave of song after vocal swapping

2. Full-song max-flow mashup

This idea attempts to extend the work we have done into a full-song mashup. The intuition is to take every possible way that we can mash the user-submitted song (song A) and the song queried from the database (song B), determine the similarity score between each of them, and then find the optimal positions to perform the mashups that maximizes the accumulated similarity scores. Since the optimization can be solved with an application of the Ford-Fulkerson

Maximum Flow algorithm, we decided to name this algorithm the “max-flow mashup”.

The algorithm will first consider all possible 4-bar combinations between songs A and B. This will generate a table *T* where *T[i][j]* denotes the similarity score of bars *i* to *i+4* of song A and bars *j* to *j+4* of song B. This table of scores can be used to determine the mashup between song A and song B that maximizes the similarity scores. We decided to first apply a high-pass filter to the vocal tracks with the same reason as the previous algorithm. Then using song A, at every point where a mashup should occur (as determined by the algorithm), we will randomly swap out one of the 4 tracks of song A with the corresponding part in song B. We introduce cross-fade effects between the mashup boundaries too. Finally, we normalize and readjust the volume of the swapped segments to match the volume of the original segment. This achieves a randomized but coherent full-song mashup result.

3. Best-match mashup

Our experimental result for the *Ford-Fulkerson* Mashup shows that the initial search from the database might not always return the best matching segment of two songs - it only gives the best result for the segment extracted using time segmentation, but neglects the other segments. The intuition for this algorithm is thus to refine the search on every segment of song A and song B. The best match across all comparisons will theoretically give us a better matching segment between song A and song B than the one we found, which can be used to create a “These two songs have the same key” mashup.

We have conducted our experiments on a test set of 200 songs. For each song, we used the first stage of the pipeline (section 3.3) to select a segment (segment A from song A) and get the closest matching segment (segment B from song B). Then, we conduct a tabular search with song A and song B, comparing each 8 bar segments of song A with that of song B, and record the new score. On average, we found the new score to be an 8% improvement over the old score. This suggests that this algorithm could be an improvement over the naive “These two songs have the same key” algorithm.

Section 4: Frontend Development and User Feedback

4.1 Frontend development

Our graphical user interface serves as a bridge between the user and the backend system. Upon launching the GUI, the user will be brought through the mashup pipeline (section 3) step by step. To create our GUI, we used JavaFX.

Design justification - What is JavaFX, and why use JavaFX?

JavaFX is a Java library for creating rich GUIs with visually appealing user interfaces. It has support for multimedia and offers a high level of customizability using CSS.

We chose to use JavaFX for GUI development due to its simplicity in creating powerful graphical user interfaces. Since our backend was written in Python, we needed a GUI library capable of passing commands to our Python backend system. Using Java's ProcessBuilder, we were able to call our Python backend and communicate data with it.

Another reason why we chose to use JavaFX is its well-developed integration between asynchronous function calls and the GUI interface. As our backend system involves many large machine learning models, running them would take some time. We would not want to block the main GUI thread while running the models, but at the same time we want to keep the GUI interactive, and show the user our progress in running the models. JavaFX's Task and Service classes provides exactly what we needed.

To make the GUI look refreshing, and to reflect the vibrant, artistic nature of our project, we chose to use a dark theme with bright, glowing colors for texts, buttons, and logos. The flow of our GUI is as follows:

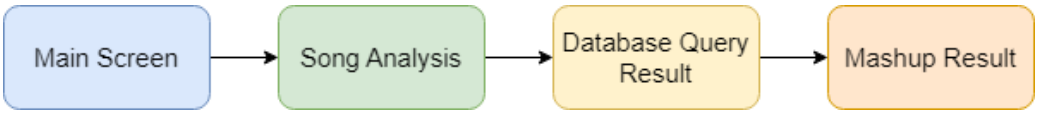


Figure 28: User flow of the AutoMashup GUI

Main Screen

Upon launch of the program, the user will be greeted with the following screen.

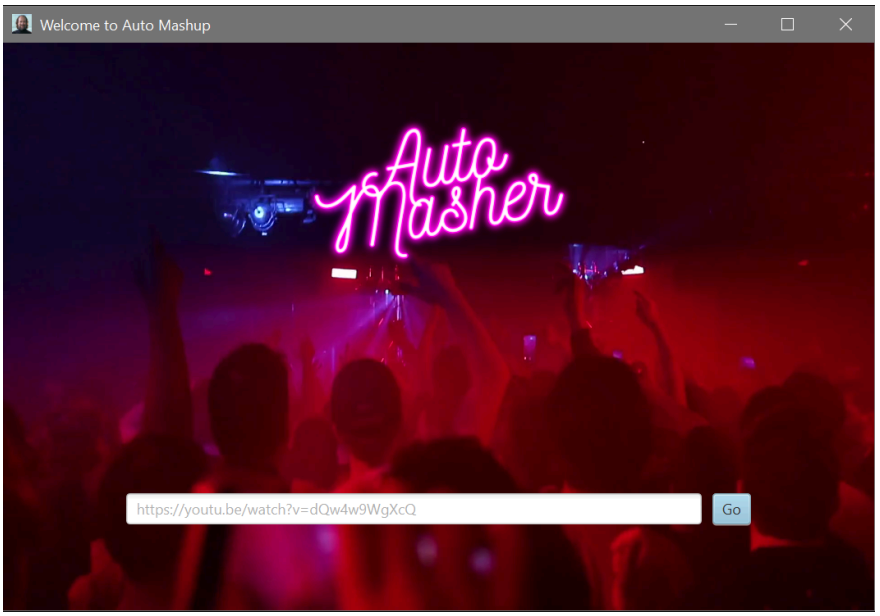


Figure 29: The main screen upon launch of our GUI

After startup of the GUI, the Python backend system is started and awaits commands sent by the GUI. On this screen, the user will be able to enter the link of the song he/she wishes to form a mashup with. Validation checks will be made to ensure that the inputted link is indeed a YouTube video. The GUI will then load the video's YouTube thumbnail.

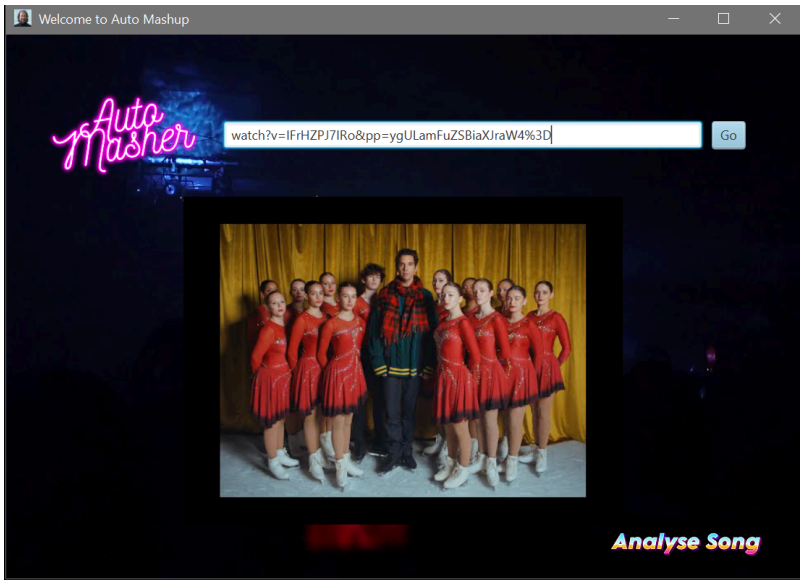


Figure 30: After the user confirms that the thumbnail is correct, he/she can advance to the next step

After the user sees the thumbnail and confirms that the YouTube link is correct, the user may click on “Analyze Song” to advance to the next step. A command will then be sent to the Python backend to run a quick analysis on the song.

Song Analysis Screen

The song analysis screen is where we show the user the analyzed information of the song he/she inputted. We display the spectrogram of the song, as shown in the following screenshot.

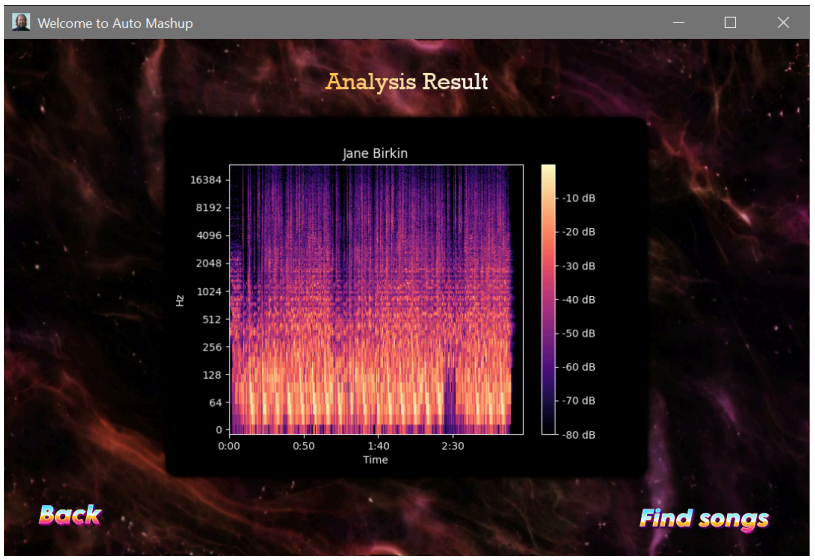


Figure 31: The spectrogram of the song is displayed

Implementation detail - Why did we choose to display the spectrogram?

The spectrogram is a visual way of understanding the song inputted. It displays the intensity of the different frequencies used in the song at each moment of time. We decided to display the spectrogram over other information because as well as being visually appealing, the spectrogram of the song can be generated quickly using the Short-Time Fourier Transform (STFT) algorithm implemented in Librosa. Other information such as the harmony and rhythm analysis requires running a large deep-learning model, so it would take some time.

After confirming the graph, the user can click on “Find Songs” to query the database with the closest song found. This is a time intensive process as it requires running the audio analysis models on the song. This includes Demucs for audio separation, the chord and beat analysis models, as well as PyChorus for temporal segmentation. In the final step, the database is queried.

Implementation detail - Keeping the GUI responsive

Passing commands to the Python backend and awaiting the outputs from the models is time intensive and will result in blocking the JavaFX main thread, making the GUI unresponsive to any clicks from the user. To avoid this issue, asynchronous function calls are used.

A progress bar with text below it shows the analysis progress. This is done to keep the GUI as interactive and responsive as possible, and inform the user about the current progress.

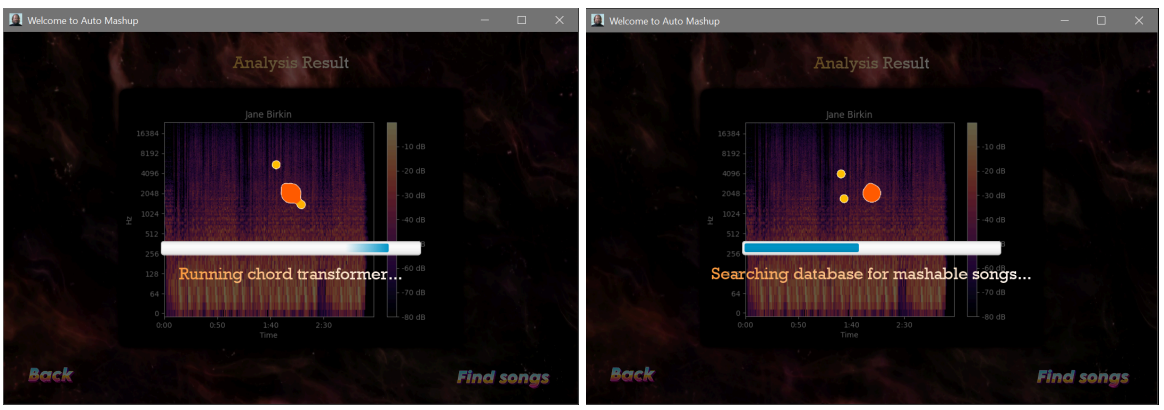


Figure 32: Loading text displayed for running the chord transformer and searching the database

Database query result screen

After the backend system finds the best matches, the user is then presented with three songs to choose from. A media player for the three songs is provided for the user to listen to the three matched songs. Sliders are provided for the user to fast forward to certain moments in the songs.

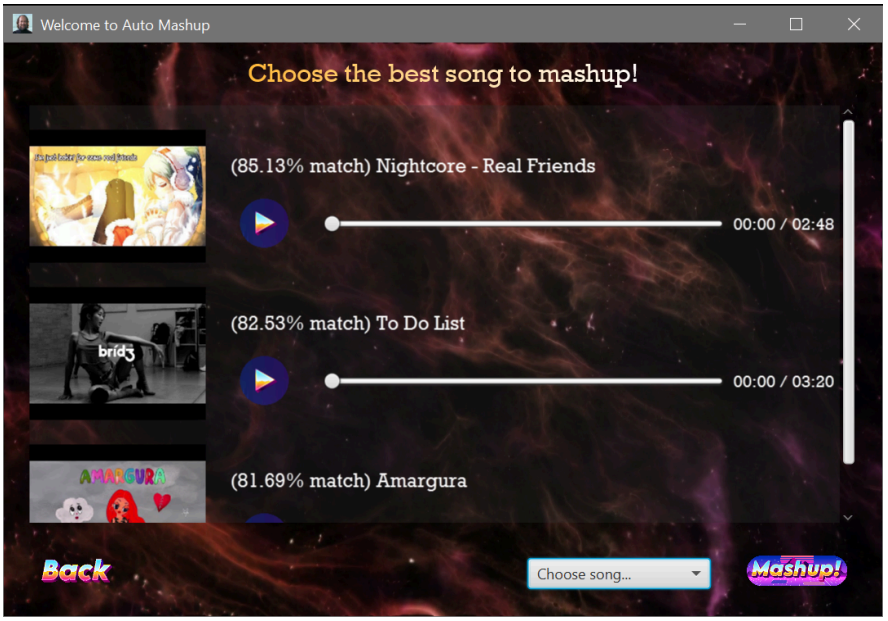


Figure 33: The user is presented with the best 3 matches. The user can select one out of the three songs for mashup

After the user selects a song, a command is sent to the backend system to conduct the mashup procedure.

Implementation detail - Why do we require the user to select a song, instead of just matching the top one?

Since our compiled list of songs in the database contains remixes, for popular songs there is a chance that a song is matched with its remixed version. To avoid this issue, we allow the user to choose from the top 3 matches. This greatly reduces the chance that a remixed version of the song is mashed up with the original version.

Mashup Result Screen

After the backend system mashes the two songs, the audio file is then sent to the frontend system. The user can then listen to the mashup result. A slider is provided for the user to fast forward to certain moments in the mashup result.

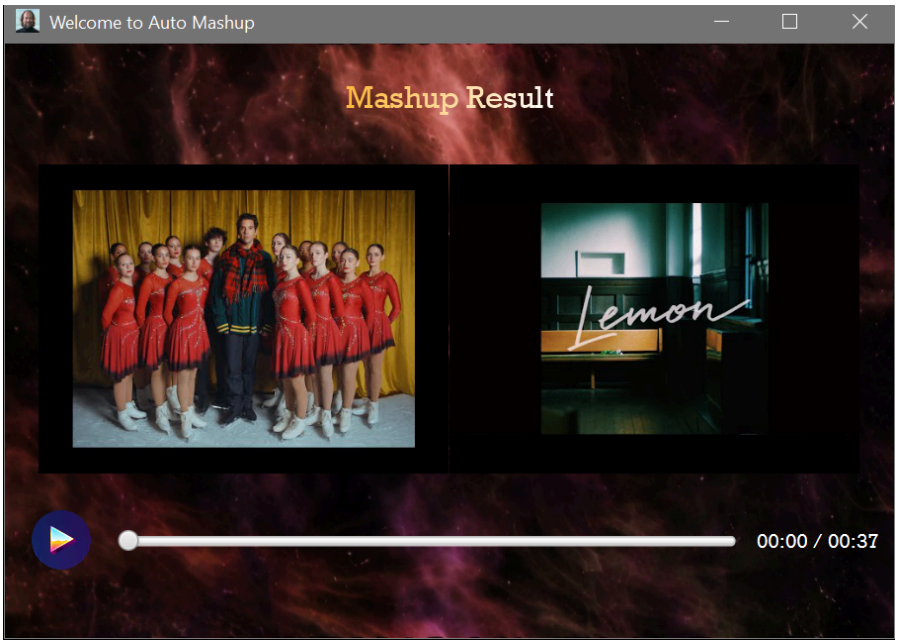


Figure 34: The user can play the mashup result

4.2 User feedback

We have collected 60 data from user feedback testing to evaluate the effectiveness of our system's performance. These results were collected via anonymous surveys online, where users are presented with song samples selected and mashed using the Global Best algorithm, and they will rate, on a scale of 1-5, the effectiveness of the mashup. They will also state whether they agree with the similarity score presented by our system.

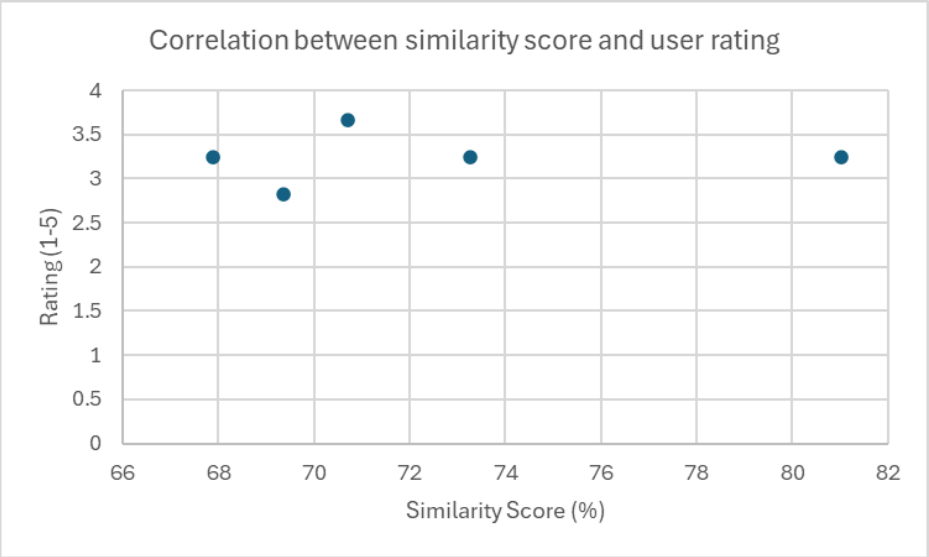


Figure 35: The correlation between similarity score and user rating

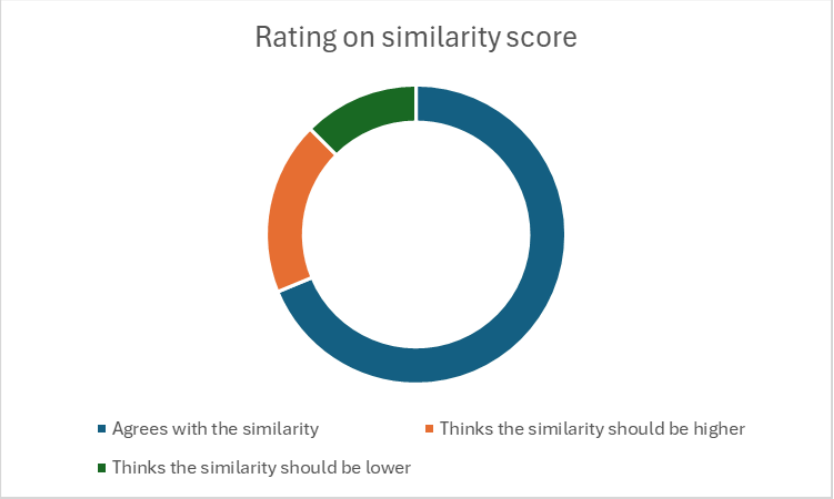


Figure 36: User feedback on the similarity score

These figures demonstrate the correlation between similarity score and the effectiveness of the mashup. We found that the user mostly agrees with the similarity score, but there is little correlation between the similarity score and the effectiveness of the mashup. This suggests that there are likely some other factors that influence the effectiveness of the mashup. In particular, our supervisor suggested that a basic mashup over the chorus might be better than performing a refined global search over two songs, which might only skew the similarity score without enhancing the overall mashup. Our supervisor also suggested that we can investigate in using autotune, because the temperament [24]of two songs might not match. This shows that the

[24] The temperament of a song describes the sub-semitone tuning precision of a song. While intervals and harmony describe the ratio of frequencies between notes, they do not describe the absolute frequency of the note. The modern convention is to produce music relative to A=440Hz but some songs, in particular older ones, might deviate from it.

similarity score reveals the quality of the mashup only to a certain degree - experimentally it is a suitable metric to perform a database search, but it does not accurately reflect the quality of the mashup in every case.

What we learnt from the survey result

We can improve our tuning analysis and correction. For example, by implementing temperament analysis and correction mechanisms, we can improve the quality of the final mashup.

We may also try alternative mashup algorithms. The survey result was only based on the global max mashup algorithm (section 3.4.2). Our survey results shows that users generally agree with the effectiveness of using a similarity score to query the database. However, other factors, such as the output quality of the audio separation model, the chorus detection algorithms, and the mashup algorithms, must be improved to enhance the quality of the mashups. We may try out other mashup algorithms.

Section 5: Discussion

5.1 Project review

We succeeded in designing and implementing an automated system to create pleasing mashups given a user-provided song. Here, we restate our two major objectives aimed at solving the issues mashup artists face when creating mashups, and how our system solves them.

Objective 1 - Easing the challenge of finding harmonious songs for mashup

Creation of a powerful and informative database of pop songs

We created a comprehensive database containing over 16,000 songs of varying genres. We pre-analyzed the songs to increase efficiency, and made it possible to query the system with our defined similarity metric that provides great matching songs with low latency - it only takes around 45 seconds to search over 1000 hours of audio! Plus, the database is easily expandable to cover even more genres of music in an even larger quantity, as expanding the database can be done as easily as running the harmony and rhythm analysis on the new song.

Objective 2 - Reducing the reliance on music theory for mashup creation

Providing accurate and reliable analysis of the song

By integrating multiple successful researches such as the BTC-ISMIR rhythm analysis model, the audio separation model Demucs, as well as the beat transformer, and together with the help of various audio and machine learning libraries such as Librosa, PyTorch, Tensorflow and TorchAudio, we successfully compiled a set of audio analysis functions that we can easily utilize, and constructed wrappers around them so that they can be tightly integrated into the analysis and generation stages of the mashup pipeline. The combined analyzed information of the models mentioned above provides us with an accurate analysis of the pop song, so that a user can successfully create harmonious mashups without the need for music theory.

Generating great mashups between two songs

Utilizing our analysis tools and database, we successfully created a harmonious mashup using various algorithmic approaches such as adjusting the pitch and BPM of the song retrieved from the database, and interweaving the choruses of both songs. Finally, we were able to generate a mashup given a user-inputted song, all without the need for the user to know music theory.

5.2 Limitations

Initially, we were quite skeptical about the idea of algorithms being able to generate pleasing mashups. However, our project demonstrates that even for something as artistic as musical mashups, algorithms and AI models are still more than capable of generating promising results. However, our system is not without its limitations and areas we could improve on. This section will document some assumptions we have made and limitations we could not overcome.

Limitations on vocal separation's quality

Within our mashup pipeline, we used vocal separation to perform vocal swapping during generation (section 3.4). However, the audio separation AI model we use, Demucs, does not always produce good vocal separation. For some songs, the resulting separated vocals may contain some noise and artifacts. Although we applied a high-pass filter to improve the sound quality, the result could still be improved.

Limitations on database query returning remixed versions of the same songs

During the query stage for the database, for popular songs there is a chance that the database would fetch a remixed version of the original song as the closest match. This will pose an issue during the generation process (section 3.4), as in some cases both the remixed and original versions of the song uses the same vocals, so conducting a vocal swap is meaningless. As of now, we have not yet found a reliable method to automatically filter out remixes of the same song.

Limitations on the size of the dataset

Our current implementation only used a database containing 16,082 songs, which is considerably less than the number of songs that were uploaded to audio streaming platforms such as Spotify [10] (100 million tracks) and Soundcloud [11](135 million tracks). This means we may inevitably miss some possible great matches, although in theory this limitation can be solved by simply increasing the size of our database.

Limitations on the accuracy of machine learning models for edge cases

The chord transformer and beat transformer models are deep learning-based and therefore, not fully accurate. These inaccuracies, when left unchecked, leak into our database, directly affecting the quality of our mashup and therefore the final result.

The beat transformer assumes a constant meter of either 3/4 or 4/4 throughout the whole song. Thus, the model fails to produce accurate results on songs that change meter. For example, the song *Knight of Firmament* changes the meter from 4/4 to 3/4 in a short interlude section at around 1:15. It also fails on songs that are not counted in groups of 3 or 4. As another example, the Mission Impossible theme is counted in groups of 5. To address this, we chose to focus on pop songs, as this genre tends to have songs that are counted in either 3 or 4. Empirically, we found that songs that are counted in 2 will automatically be counted in groups of 4, and does not greatly affect the final quality of the database query step.

The BTC-ISMIR model analyzes only the spectrogram of the song to generate the chord results of the song, and thus is limited to the quality of the spectrogram. For the database generation, we have used the HPSS[25][12,13] algorithm to first separate the audio into its harmonic and percussive components. The BTC-ISMIR model is only applied on the harmonic component; the HPSS step is skipped during the database query. This seemed to improve the quality of the database query step, but a drawback is even segments from identical sources does not result in a similarity score of 100%.

We also assumed that we will be working in the 12-tone equal temperament system with only the aforementioned 170 chords. Essentially, the frequency space of the audio is a continuous space, making harmony analysis almost impossible. This assumption restricts us to a finite set of harmonies to work with, which permits a detailed automatic harmony analysis pipeline. This is another reason we focused on pop songs: the harmony is usually more restricted and easier to detect. We also chose to circumvent this problem by heavily penalizing the “Unknown chord” label which contemporary-styled harmony usually falls into.

[25] HPSS, or harmonic percussive source separation, is a median-filtering algorithm which decomposes an audio time series into its harmonic and percussive components.

Section 6: Conclusion

6.1 Summary

Our group has successfully constructed an automated system that can produce a mashup when given a song from a user. We achieved this by first cataloging the auditory information of a large set of contemporary music, then organizing it into a database. We then query this database with a distance heuristic to retrieve musically compatible songs. Lastly, we reorganize and combine the components of the given song and the retrieved song to create a complete mashup.

6.2 Future work

Our project proved that algorithmic approaches together with the help of AI models can successfully generate pleasant transitions. Since our system is highly modular, if we could further improve and implement the following aspects, we believe the result would be even better.

Producing an even better result for a full-song mashup

Our system is currently capable of producing a pleasant mashup result of around 30-40 seconds. We have attempted to extend the algorithm to a whole song, but depending on the song chosen, the result was sometimes not as impressive as the 30-40 seconds segment. If we were able to mashup a complete song while maintaining the same quality, we believe the system would be even more valuable.

Employing generative AI models to smoothen the transitions

The involvement of generative AI for creating transitions between the two songs was one of our original proposed ideas, but we later scrapped the idea as we found that the currently accessible generative AI models do not produce the results we expected. If we could employ generative AI models to smoothen the transitions between the two songs, we believe the mashup result would sound even more natural.

Expanding the representative power of our database to genres other than pop songs

To further improve the possible matches, we suggest expanding the database to contain a much larger number of pop songs. This will further improve the variety in our database, and find possibly even better matches. Furthermore, if we were able to extend our algorithms to analyze songs other than pop songs, we believe the system would be able to create even more exciting mashups.

Developing an even more accurate temporal segmentation algorithm

Our current algorithmic approach for temporal segmentation produces accurate results most of the time. We could improve on the accuracy of temporal segmentation by training a machine learning model to detect the chorus. We believe with an increased accuracy in chorus detection, the chances that a database query returns a false result would be lower, and conducting a database query would be even more meaningful.

References

[1] oneboredjeu (2016). X Gon Give It To Ya Maybe - Carly Rae Jepsen vs. DMX (Mashup)

X Gon Give It To Ya Maybe - Carly Rae Jepsen vs. DMX (Mashup)

[2] Horner, A. and Wu, X. (2023). An automated pop song mashup system, app, and evaluation.

[3] Davies, M. E., Hamel, P., Yoshii, K., and Goto, M. (2014). Automashupper: Automatic creation of multi-song music mashups. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 22(12):1726–1737.

[4] Xing, B., Zhang, X., Zhang, K. et al. PopMash: an automatic musical-mashup system using computation of musical and lyrical agreement for transitions. Multimed Tools Appl 79, 21841–21871 (2020). https://doi.org/10.1007/s11042-020-08934-2

[5] Wu, X., & Horner, A. (2023c). An automated pop song mashup system using drum swapping. The Journal of the Acoustical Society of America, 154(4_supplement), A288–A289. https://doi.org/10.1121/10.0023554

[6] Park, J., Choi, K., Jeon, S., Kim, D., & Park, J. (2019, July 5). A bi-directional transformer for musical chord recognition. arXiv.org. https://arxiv.org/abs/1907.02698

[7] Zhao, J., Xia, G., & Wang, Y. (2022, September 15). Beat Transformer: Demixed Beat and Downbeat Tracking with Dilated Self-Attention. arXiv.org. https://arxiv.org/abs/2209.07140

[8] Défossez, A. (2022). Hybrid spectrogram and waveform source separation.

[9] A chorus section detection method for musical audio signals and its application to a music listening station. (2006, September 1). IEEE Journals & Magazine | IEEE Xplore. https://ieeexplore.ieee.org/document/1677997

[10] Spotify (2024). Spotify - About Spotify https://newsroom.spotify.com/company-info/

[11] SoundCloud (2024). SoundCloud Go+ Tracks - SoundCloud Help Center https://help.soundcloud.com/hc/en-us/articles/360052599653-SoundCloud-Go-tracks

[12] Fitzgerald, Derry. “Harmonic/percussive separation using median filtering.” 13th International Conference on Digital Audio Effects (DAFX10), Graz, Austria, 2010.

[13] Driedger, Müller, Disch. “Extending harmonic-percussive separation of audio.” 15th International Society for Music Information Retrieval Conference (ISMIR 2014), Taipei, Taiwan, 2014.

Appendix A - Project Planning

This section contains the updated Gantt chart and division of labor.

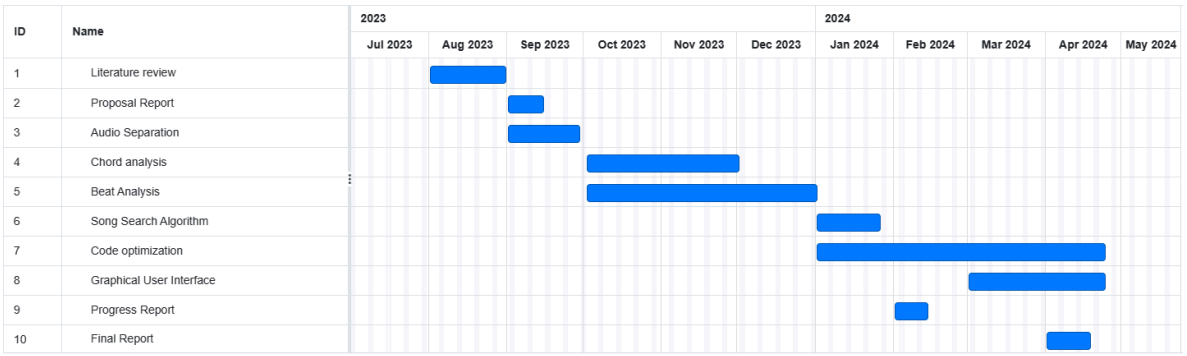


Figure 37: Gantt Chart

Task	Harris	Darin	Anthony	Joshua
Audio Analysis		•	•	
Music Database		•		
Mashup Pipeline		•		•
Experimentations	•	•	•	•
Graphical User Interface				•
Frontend - Backend communication		•		•
FYP Video	•			
FYP Reports	•	•	•	•
FYP Presentation	•	•	•	•

Figure 38: Division of Labour

Appendix B -

Required Hardware and Software

This section outlines some specific hardware and software used for development and those which may be used in the future.

B.1. Hardware Requirements

- Graphics Processing Unit (GPU):

While not mandatory, a GPU with CUDA support can accelerate certain deep learning tasks, such as those involved in the BTC-ISMIR and beat transformer models. This would enhance the efficiency of the harmony and beat analysis processes. A minimum of 4GB VRAM is required to run the beat transformer model.

B.2. Software Requirements

- Programming Languages:

The application involves complex algorithms and machine learning models, requiring a programming language suitable for scientific computing and deep learning. Python is used along with its vast ecosystem of libraries such as PyTorch, Librosa, TorchAudio and Hugging Face. For the GUI, we used Java with its JavaFX GUI library.

- Version Control System:

Git is used to manage code versions and collaborate effectively with team members. This ensures efficient tracking of changes and facilitates collaboration in a development environment.

Appendix C - Meeting Minutes

C.1. 1st Meeting Minutes

Date: 27/07/2023
Time: 18:00
Location: Library
Members present: Anthony, Joshua, Harris, Darin
Members absent: None
Recorder: Harris

- Approval of last meeting Minutes
  N/A

- Discussion Items
  - Members discussed their expectations of the project and what they want to contribute.
  - Expectations on the project outline were aligned as follows:
  - The goal is to construct is a Automatic Mashup System which takes in audio files and produces a desirable mashup with elements from the input audio
  - There will be separate components of the system, including:
    - Audio separation (Separate a song into vocals, instrumentals, drums, etc. Separate audio into clips)
    - Audio adjustment (Adjust pitch and tempo to match different clips, use AI techniques to influence the style of the song)
    - Audio generation (Use generative AI to create more clips to use in combination phase)
    - Audio combination (Use existing or design algorithms to combine clips into a cohesive song)
  - These components may use existing open-source code or techniques, or we may create our own implementations as well.
  - Members were assigned readings to go over before the next meeting.
- Meeting Adjournment and Next Meeting
  - The meeting is adjourned at 6:45pm
  - A meeting with Prof. Horner will be scheduled to discuss progress.

C.2. 2nd Meeting Minutes

Date: 05/9/2023
Time: 15:30
Location: Prof. Horner's Office
Members present: Anthony, Joshua, Darin
Members absent: Harris
Recorder: Anthony

The Project Supervisor, Prof. Horner, was present at the meeting.

- Absentee(s)
  - Harris was excused from the meeting due to health issues, which was aforementioned to the remaining members but not to the supervisor.

- Approval of last meeting Minutes
  - The minutes of the last meeting were approved without amendment.

- Discussion Items
  - Members summed up and reported the discussion results regarding the project direction from the last meeting, and the current progress to the supervisor, and comments and advice were given by the supervisor.
    - The project direction was determined to be Music Mashups involving the usage of AI.
    - The Supervisor suggested seeking assistance from one of the research postgraduates he is supervising, **Wu Xinyang(xwuch@connect.ust.hk)**, who has been working in the similar direction, would also be a viable option.
    - The supervisor commented that it might be too much to develop a fully functional online application for the integrated AI Music Mashup system we proposed in the 1st meeting. An offline prototype would be sufficient for the project.
    - ***An Assumption of the audio tracks we would use are Pop music was made by the Supervisor.***
    - For the audio separation module, the supervisor suggests we make use of the existing Moose.AI tools to perform audio layer extraction.
      - Member Darin reported that he has been working with the AI provided by Meta, but encountered certain difficulties which led him to re-implement the modules on his own.
      - Supervisor commented that as long as the performance is better, subject to our interpretations, than that can be

provided using Moose.AI, and the progress allows, we can continue on our walk.

- Member Darin mentioned that the reason we opted for Meta's model was apparently, Moose.AI is a commercial AI tool for audio separation.
- Supervisor replied that if we do attempt to make use of Moose.AI's for the sake of the project, the trial version would be sufficient.
- Supervisor made several remarks on previous results on the usage of Moose.AI on separating layers from given soundtracks.
    - The vocals, instrumental parts extracted were good enough to be used.
    - The drum lines extracted were clear
    - The bass lines extracted were very thin
    - In general, each layer alone is perceptually weak to the supervisor, but it would be fine if they are combined together.

- For the audio adjustment module, the supervisor commented that it might not be necessary for us to incorporate advanced techniques from the reading material about audio style transfer into the project.
    - The supervisor mentioned that often, in pop music, the style or genre of a song is predetermined by the bass line and the drum line. Tinkering with these parts would be a way to achieve style transfer.
    - Experimenting with advanced techniques is encouraged as it is also a method to achieve the goal.
    - Member Anthony asked about whether introducing a spectrum-like system to alter the generated mashup would be feasible for the project. The supervisor replied that it would be workable, but it would not be the first few steps we are going to take.
    - The module might be required to be able to transpose source audio to the required (diatonic) tonality(what kind of tonal system the pitches of the song is following), or to alter the tempo(or speed) to aid the production of the mashups as preprocessing.
    - The module might be required to be able to modify the amplitude of the audio tracks due to some extracted tracks being possibly too weak to be useful.

- For the audio generation module, the supervisor commented that we should set our priority to generating music mashups with existing music first, this can be done but only after the above item has made certain progress.
- For the audio combination module, the supervisor mentioned the approach on how to combine the audio tracks can be a rather artistical item to consider. He recommended the simplest method, to combine the layers on top of each other, for our early steps.

- The supervisor has provided some guidelines on the approach to do the mashups, and some details we need to be aware of.
  - The supervisor suggested the **first step we should take is to swap the drum lines between the two songs** as it is pitchless.
    - The members were sent an email related to this approach back in June by **Wu**, which contains a qualtrics link to be a survey to our aesthetic preference to a given collection of original soundtracks with altered drumlines.
  - The supervisor suggested the **second step we should take is to try to swap the bass lines between the two songs.** He mentioned that there are differences between the effects on swapping the bass lines along with the drum lines and without it.
    - The supervisor commented that considering the bass line and the drum line of the same song as one part is more logical in a musical sense.
    - The supervisor emphasized that the bass line extracted by Moose.ai can be very hard to be audible. It was verified by the members by listening to some of the extracted bass lines played by the supervisor.
    - The supervisor made a remark that often matching the tempo (speed of the song) of both songs is not a trivial step to take. If certain preprocessing steps were not involved, it would induce subjective awkwardness to the listeners. The rhythmic pattern differences across the songs are needed to be aware of.
  - The supervisor mentioned that often dissonance (pitch clashes which induce subjective unpleasant feelings on listeners) would appear if the tonality of the two source soundtracks does not agree on each other. Transposition might be required before producing the mashups.
    - The supervisor suggested that it might be helpful to obtain information like tempo, tonality, meter (in 2 or in 3), etc., of the original soundtracks online to aid any preprocessing steps we might be going to take.

- Member Joshua mentioned doing audio style transfer in the project. The supervisor replied that might be a rather low-level(in terms of user interactivity) way to produce mashups. The supervisor suggested **we should start working on the project by working on a high-level approach, then move on to experimenting with certain low-level techniques.**
    - The supervisor mentioned the following items could be some of our directions to do experiments with in the later steps:
        - AI Music Generation
        - Audio separation
        - Audio style transfer
        - Creating mashups involving generated ai music audio
        - Mashups involving multiple tracks or mashups
        - Methods on combining audio tracks

- Overall, the supervisor did not show major disagreements on our direction regarding the project.

- The supervisor advised the group should finish the project proposal by the end of this week for inspection and follow-up.
- The length of the mashup made was not discussed in this meeting.

- Meeting Adjournment and Next Meeting
    - The meeting was adjourned at 16:03.
    - The details of the next meeting are to be discussed.
    - A first draft of the project proposal is required to be submitted to the supervisor for inspection and follow-up before 11/9/2023.
    - The project proposal is due 15/9/2023, ten days later.

C.3. 3rd Meeting Minutes

Date: 22/9/2023
Time: 15:30
Location: Room 2466
Members present: Anthony, Joshua, Darin, Harris
Recorder: Anthony

- Approval of last meeting Minutes
    - The minutes of the last meeting were approved without amendment.

- Discussion Items
    - Members summed up and reported the discussion results regarding the project direction from the last meeting.
        - Member Joshua asked about how exactly we should deal with the Copyright of the tracks we were using.
        - Member Darin comments that we can check the usage of the Creative Commons, and Fair use declaration
        - Member Darin stated that Audio Separation module, BPM detection, key detection. tempo/key adjustment, fade and reverb for the system are done.
        - Member Anthony mentioned that the current way we combine the audio is just simply overriding operator + in Python
        - Members of the Group decided the Audio Manipulation(Creation) module is needed to be worked on. The members wants to develop a functionality that identifies every chord progression in a given tracks, then by some method to generate audios with a GAN model, which will be done later.
        - Member Harris mentioned that we can’t find a suitable database
        - Member Darin and Anthony stated that we could have asked the Supervisor about the database he and his RPG is using.
        - Members of the Group proposed an idea that we can take the head and tail of the music, and then pasted it into a generative AI model to generate the middle section of a song.
        - Members of the group mentioned that we have to decide when to call the audio adjustment module in the creation module
        - Members of the group have to decide whether to make the input audio bass boosted or normalized across the source to fix the bass and drum beats.
- Meeting Adjournment and Next Meeting

- The meeting was adjourned at 16:00.
- The details of the next meeting are to be discussed.

C.4. 4th Meeting Minutes

Date: 15/11/2023
Time: 16:30
Location: Prof. Horner's Office
Members present: Anthony, Joshua, Darin, Harris
Recorder: Darin

The Project Supervisor, Prof. Horner, was present at the meeting.

- Approval of last meeting Minutes
    - The minutes of the last meeting were approved without amendment.

- Discussion Items
    - Beat Analysis Algorithm Difficulties:

      Discussed challenges encountered in implementing the beat analysis algorithm using librosa. Noted the lack of accuracy in the current implementation and the impact on beat tracking. Professor Horner shared insights, mentioning that his Ph.D. student faced a similar issue and will provide an alternative method to improve accuracy.

    - Mixing, Mastering, and Balancing:

      Discussed the importance of mixing, mastering, and balancing in the final audio output. Recognized the need for a cohesive and well-balanced sound in the mashup compositions. Agreed to explore techniques and tools, including audio editing software, for effective mixing and mastering.

    - Next Steps:

      Team members assigned tasks related to integrating the upcoming beat analysis method and testing its impact on accuracy. Research and experimentation planned for implementing mixing and mastering techniques. Set a follow-up meeting to discuss any potential issues.

- Meeting Adjournment and Next Meeting
    - The meeting was adjourned at 17:05.
    - The details of the next meeting are to be discussed.

Appendix D –

Musical Terminology Glossary

This section contains a glossary of musical terms used in this report to aid its understanding.

- **Note**
  A sound with a specific frequency played by an instrument. It is what we hear when we press one piano key or pluck one guitar string.

- **Chord**
  A chord is a sequence of notes played in parallel. Often, words like Major, Minor, Augmented, Diminished, Seventh and Suspended are used to describe the tone quality of the chord.

- **Harmony**
  The harmony describes the sound quality of the chord. If it sounds “good”, then the chord is consonant, otherwise it is dissonant. Classical music theory has rigorous definitions for the consonance or dissonance of chords, but it is not necessary to understand this report.

- **Key**
  The key describes the tonal center of a song. The key can be “Major” or “Minor”, along with 12 different keys on the keyboard, for example “C Major” or “D minor”.

- **Beat**
  A unit of time in the context of a song, particularly the song’s rhythm.

- **Downbeat**
  The beat that indicates the beginning of a measure.

- **Tempo**
  Describes the speed of the song. If a beat is short, then the tempo is fast, and vice versa.

- **12-tone Equal Temperament**
  The music system commonly used by the west since the Classical Period (c.a. 18th Century). An octave consists of 12 tones, and each pair of consecutive notes maintains a ratio of 12th root of 2.

- **Intervals**
  The “vertical” distance between two notes, usually used to describe a leap consecutive notes or distance between notes in a chord.
    - A simple interval means that the fundamental frequency ratio between the two notes can be approximated by a rational number with a small integer denominator. Notes with simple intervals to one another produce a consonant sound.
    - A complex interval means it is not simple, and notes with complex intervals to one another produce a dissonant sound.
- **Meter**
  The underlying structure of a measure of a song.
    - Simple meter: 2/4, 3/4, 4/4
    - Compounded meter: 6/8, 9/8, 12/8

Appendix E – Survey Questions

This section contains the google form we used in our anonymous survey on the effectiveness of an automatic mashup system designed by our group.

E.1. Part I

This question is to inform all participants that the survey will be conducted in a completely anonymous manner, as well as asking for their consent in participating in the survey.

This survey will be completely anonymous, and it will take (probably) less than 10 minutes to complete. Please let us know when you are ready.

- ◯ I agree to proceed :D

E.2. Part II

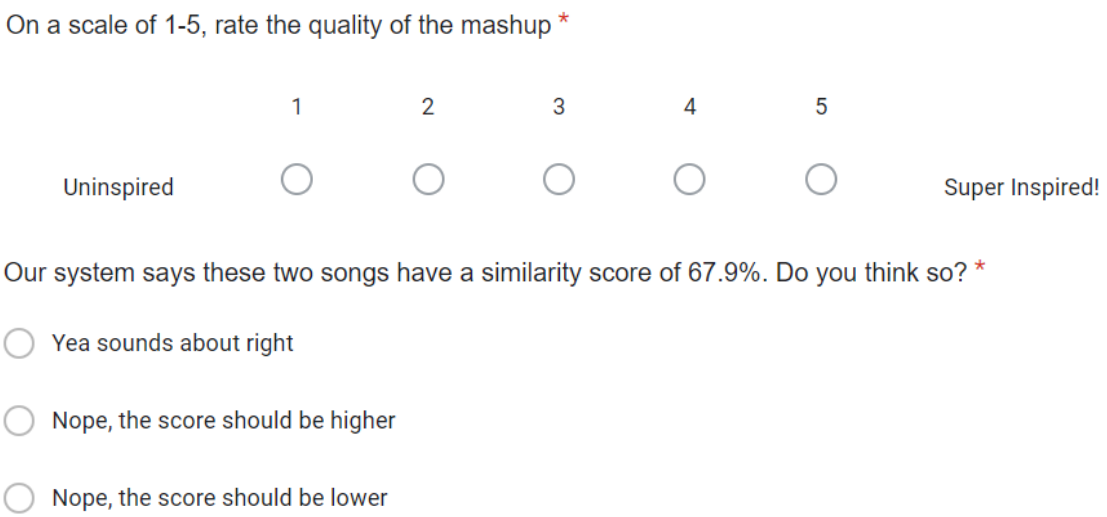
For each Mashup sample we have provided in this survey, we have asked the following questions:

Are you familiar with the input song? (If you know the name of the song, you can enter it in the "Others" option) *

- ◯ Nope never heard of it
- ◯ Sounds somewhat familiar...
- ◯ Yep, heard of it before, but couldn't recall its name
- ◯ 其他...

Are you familiar with the song from the database? (If you know the name of the song, you can enter it in the "Others" option) *

- ◯ Nope never heard of it
- ◯ Sounds somewhat familiar...
- ◯ Yep, heard of it before, but couldn't recall its name
- ◯ 其他...



We have provided the following mashups, with the segments from the original songs listed in the following table:

Name of the Input song	URL	The song found in our database	URL
Ed Sheeran - Bad Habits	Ed S...	Lady Gaga - Poker Face	Lady ...
Coldplay - Hymn For The Weekend	Solt...	Soltera Remix - Lunay X Daddy Yankee X Bad Bunny	Solter...
YOASOBI - 夜に駆ける	YOA...	Da-iCE - スターマイン	Da-iC...
TWICE - MOONLIGHT	MO...	MAMAMOO - Starry Night	[MV] ...
Raon - ENDROLL	Rao...	Alphaville - Big In Japan	Alpha...