School of Computing
FACULTY OF ENGINEERING AND
PHYSICAL SCIENCES

Final Report

Evaluating Audio Source Separation Models Through Objective Metrics and Perceptual Evaluation

Manil Kohombage

Submitted in accordance with the requirements for the degree of
BSc Computer Science

2023/24

COMP3931 Individual Project

The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Final Report	PDF file	Uploaded to Minerva (08/05/24)
Digital participant consent forms	PDF file / file archive	Uploaded to Minerva (08/05/24)
Link to online code repository	URL	Sent to supervisor and assessor (08/05/24)
User manuals	PDF file	Sent to client and supervisor (08/05/24)
Link to video demo	URL	Sent to client and supervisor (08/05/24)

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of Student) 

Summary

Audio source separation is a technology that is able to separate individual sound sources (e.g., drums, bass, vocals, and other instruments) from a mixed audio signal. Currently, the performance of source separation models is primarily bench-marked using objective measures such as the Signal-to-Distortion Ratio. While these metrics provide a quantitative measure of separation quality, their correlation with human perception and subjective evaluation remains an open question.
This project investigated the relationship between objective measures and perceptual evaluation of audio source separation models. Specifically, it aimed to determine whether or not the widely used Signal-to-Distortion Ratio (SDR), Signal-to-Interference Ratio (SIR), Source to Spatial Distortion Image (ISR), and Signal-to-Artifacts Ratio (SAR) metrics accurately reflect the perceived quality of separated sources by human listeners.

Results for this study were obtained by a developed web application that facilitates the practice and evaluation of two prominent source separation models, Spleeter and Demucs. The application allowed users to assess the performance of these models on various songs by providing the quantitative metrics (SDR, ISR, and SAR) for each separated source. Users would listen to the original clean sources and directly compare them with the separated outputs from Spleeter and Demucs, enabling a perceptual evaluation.

Through user testing and evaluation, the project revealed significant discrepancies between the objective measures and the perceived quality of separated sources. In numerous cases, the perceptual evaluation by users contradicted the quantitative metrics, highlighting the limitations of relying solely on mathematical equations to determine audio quality.

The project achieved an outcome that recognises the importance of using a more holistic approach, incorporating both quantitative and qualitative methods to form a comprehensive evaluation. As the field of audio source separation continues to evolve, by combining quantitative metrics with perceptual evaluation, researchers can create models that not only excel in numerical performance but also deliver a listening experience more tailored to human perception.

Acknowledgements

I would like to thank my supervisor, Arash Rabbani, for his guidance on the the project and teachings on the theory behind both model's architecture. Understanding the inner workings of the system would have been much more difficult without Arash's explanations.
In addition, thank you to Julian Brooks for his advice and suggestions through the early early stages of the project.
Finally, thank you to my assessor Dibyayan Chakraborty for relaying feedback on the projects progress.

Thank you to my family for continuously supporting me through this academic journey, as well as my friends who took part and interest in my user testing.

Contents

Chapter 1

Introduction and Background Research

1.1 Introduction

When listening to music, humans can innately pick out instruments within a complex mix of sounds. For example, when listening to music, it is trivial to pick out the drums, guitar, bass, and vocals. However, there is no way to pull out these isolated instruments that our mind separates from the music. Following the trends of most technological advancements, if our brains can process music in this manner, there must be a possibility to develop algorithms that can do the same.

Over the last century, there has been substantial demand for this technology, commonly named "de-mixing." The notion of seamlessly separating audio back into its individual components has been nothing more than a theoretical concept. However, thanks to recent technological advancements in deep learning, this technology now exists.

Introducing Spleeter, a Python library that utilises TensorFlow (a machine learning and artificial intelligence library). Spleeter is a source separation library with pre-trained models that allows users to separate audio files directly through the command line. (Hennequin et al., 2020)
Currently, Spleeter can perform stem splits of 2, 4, or 5 isolated tracks. These include:

Vocals / Instrumental (Other)	2 Stems
Vocals / Drums / Bass / Other	4 Stems
Vocals / Drums / Bass / Piano / Other	5 Stems

Demucs is another source separation model, with relatively the same option of outputs. However, its architecture is completely different; Demucs utilises PyTorch, another deep learning framework. (Rouard, Massa and Défossez, 2023)

Audio source separation models are usually compared against each other using an evaluation metric called Source-to-Distortion Ratio (SDR). Grais et al. (2019) explains this value as a combination of three energy ratios: Source to Spatial Distortion Image (ISR), Source-to-Interference Ratio (SIR), and Source-to-Artifact Ratio (SAR). These metrics are further discussed in 1.3.3.

This project aims to compare the two source separation models by creating a user-friendly interface to allow users to de-mix songs and discover which model is more accurate. This will be based on their perceptual evaluation by comparing model outputs against the original signal. User data will then be analysed alongside evaluation metrics (SDR, ISR, SAR, and SIR) to determine whether or not they provide a valid justification for a model's performance, or if there are limitations to relying on these objective measures. As a result, a comprehensive assessment can be made by combining user feedback and quantitative evaluation, offering insights into the strengths and weaknesses of the models under comparison.

1.2 Analysis of Previous Similar Works

In order to create a user friendly interface, an analysis of previous similar works has been conducted.

1.2.1 DeMiXER

"DeMiXER", is a web application aimed at music professionals to separate tracks into different stems. According to audiosourcere (2024), DeMiXER's technology is powered by AudioSourceRE's products, which are cloud-powered AI with audio source separation algorithms.
Its source separation model makes neither use of Spleeter nor Demucs and has not been made open-source or publicly available. Only a 30 second preview of the isolated tracks are available too; the full-length downloads are hidden by a paywall. However, from an initial analysis, separated results are of great quality.

Figure 1.1 is an arrangement view after a song is processed on DeMiXER's servers. It follows important elements to provide a visual design for good usability. According to Kelly (2019), visual design for good usability is achieved through the efficient use of grouping and typography. This involves the intestinal choice of spacing, colour, and texture between neighbouring connected components and suitable typography to separate groups.
Helping users understand and seamlessly de-mix audio files is achieved by providing audio visualisation through waveforms, with a click-to-play feature.

Figure 1.1: DeMiXER GUI (audiosourcere, 2024)

1.2.2 Free Music Demixer

"Free Music Demixer", is another web application that produces similar results, yet is completely free. This is due to audio processing being performed on the user's local device, as well as the system's use of Demucs' library. As a result, processing time is completely dependent on the user's machine. (Hanssian, 2023)
Although the system is free and produces accurate results, it lacks a good user-interface; there

is little to no beauty or emotional content in its visual design (figure 1.2).

Upon completion of audio processing (figure 1.3), there is little indication of completion. Users may believe that processing is still occurring as there is no 'Completed' alert. Instead, ambiguous download links appear which could easily be missed or interpreted differently.

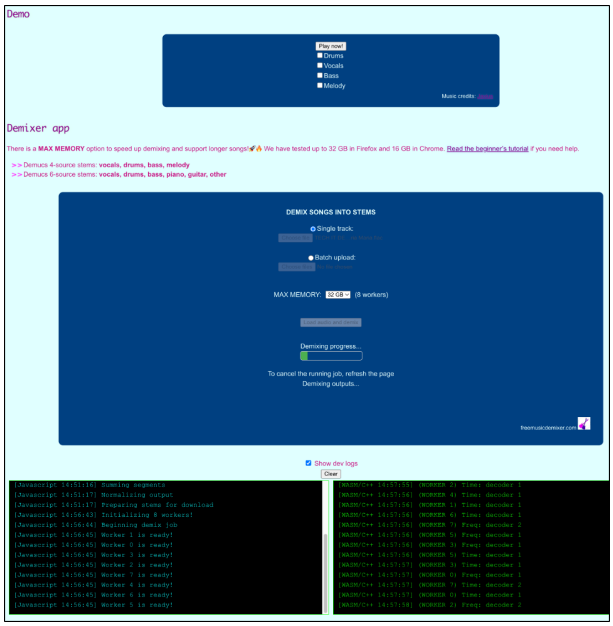

Figure 1.2: Free Music Demixer GUI (Hanssian, 2023)

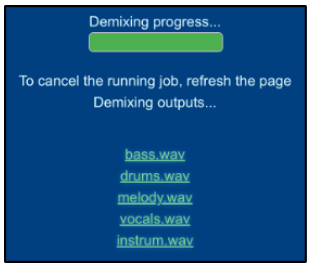

Figure 1.3: Free Music Demixer [Processing Completed] (Hanssian, 2023)

1.2.3 SpleeterGUI

"SpleeterGUI", is a local application that uses Spleeter. It has many dependencies on the user's device, including its operating system (e.g. Windows 10 only) and specific hardware requirements. (Mitchell, 2022)

Even though its use base is limited, it provides a simplistic yet effective approach to utilising Spleeter's audio separation library. Spleeter's command line options are visualised and represented via buttons and check-boxes. For example, the application informs the user of the number of available sources Spleeter is able to separate, further allowing them to select whichever they desire. Giving the user the freedom to choose the amount of sources to separate is advantageous for a few reasons. Some users may only require vocal separation without any interest in the other instruments. This does not just cater to the user's specific needs but also results in a quicker

separation process compared to separating all sources.

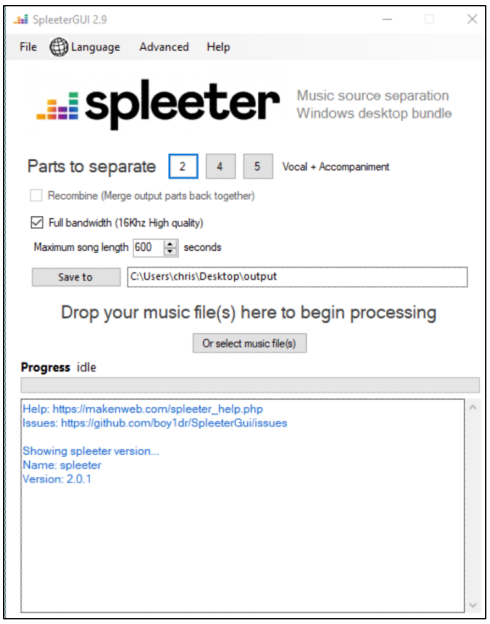

Figure 1.4: SpleeterGUI (Mitchell, 2022)

1.2.4 Analysis Overview

All applications found online that utilise either Spleeter or Demucs did not provide an on-demand evaluation tool, largely due to the main use case to solely separate audio sources without having access to clean signals. Thus, it is not possible to compare artificially created separations against the non- existing original ones. Therefore, research indicates that there are no viable existing all-in-one solutions for this problem.
There were no findings of applications that allowed a user to choose which separation model to use. If a user wishes to compare models, they need to use separate online tools, or install each model's libraries. They would then have the option to A/B test each isolation against each other, or calculate their evaluation metrics, the latter solution being too advanced for the general user.

Making a web application would cater to more users, as local applications have too many dependencies. Ideally, utilising cloud servers would be beneficial as the dependency of the user's hardware would be mitigated. However, as deployment was not a focus on this project, the web application will be local.

1.3 Background Research

1.3.1 Spleeter's Architecture and How It Works

Hennequin et al. (2020) states that Spleeter comes with a set of pre-trained models that are U-nets, a conventional neural network architecture with skip connections. U-nets are broken down into encoders and decoders; encoders extract important features by reading and understanding the input audio, whereas decoders pull out their individual components.
Figure 1.5 shows how Spleeter uses a 12-layer model, equally divided into six encoders and decoders. Skip connections connect the encoders output at each down sampling step, directly

to the corresponding layer in the decoder. This allows the system to estimate a soft mask for each source of the audio file, which is the relative amount or weight of that stem (Drums, Bass. Etc) at every point in the song; in other words, it calculates how much of a specific instrument is present at that point.

The paper (Hennequin et al., 2020) mentions that the model separates sources using the calculated soft mask or by using multi-channel Wiener filtering. Wiener filtering is a signal processing technique utilised for noise reduction. It leverages spatial information by processing signals from multiple channels (left and right sources) to reduce noise in certain areas while preserving the desired signal. To do this, Spleeter estimates the spectrograms (frequency-time representations, visualised in figure 1.7) of the individual source signals to create an optimal Wiener filter for each source; essentially, to filter out the unwanted parts of the signal.

Spectrograms are generated by using a common practice signal augmentation technique named Short-Time Fourier Transform (STFT). Zhang and Zheng (2023) explains how STFT is able to transform signals into the time-frequency domain, or spectrogram. It does this by segmenting the incoming signal into a sequence of concise temporal windows, which are then each processed by the Fourier transform, yielding a representation within the frequency domain. Finally, to transform the signal into the time domain, the short-time Fourier transform is applied.

Skip connections are advantageous here, as the direct connections better retain important details that would otherwise get lost in compression.
A "bottleneck architecture" is an alternative to the use of skip connections (figure 1.6). According to Lerch and Zhou (2015), input information is compressed all the way down to a small "bottleneck" layer through a gradual decrease in the number of neurons per layer. This is then passed to the decoder to gradually expand back out. Because of this extensive compression and the absence of skip connections, the final model would have lost important fine-grained details. Thus making separation less accurate, especially in the higher frequencies of the audio and the preservation of transients. This is due to higher frequencies being more fragile to distortion when undergoing heavy compression, which is a critical region for instrument clarity and transient.

Spleeter's models were trained on Deezer's internal Bean dataset, which is not publicly available due to obvious copyright reasons. Prétet et al. (2019) mentions that the bean dataset contains *24,097* songs, *21,597* of which were used for training, *2,000* for validation, and *500* for testing. This large volume allows training on a broader range of music genres, mixing styles, and audio quality, potentially resulting in Spleeter's accuracy beating its competitors.
It is worth noting that the released models were trained on spectrograms up to 11kHz, meaning all outputted separation files will have discarded frequencies above 11kHz.

Training loss, referred to as L1-norm, is the measure of accuracy between Spleeter's produced spectrograms and the target ones (from the training dataset). It is calculated by masking the input mix spectrograms (representations of the original audio, containing all its sources) and the target source spectrograms. During training, the goal is for the model to generate output spectrograms that are close to the target ones. (Hennequin et al., 2020)
Hennequin et al. (ibid.) also claims that Spleeter separated the entire musdb18 dataset, around 3.5 hours of audio, in under 4 minutes. Meaning it is able to separate 100 seconds of stereo audio

into 4 stems in under 1 second, proving its efficiency in processing large datasets.

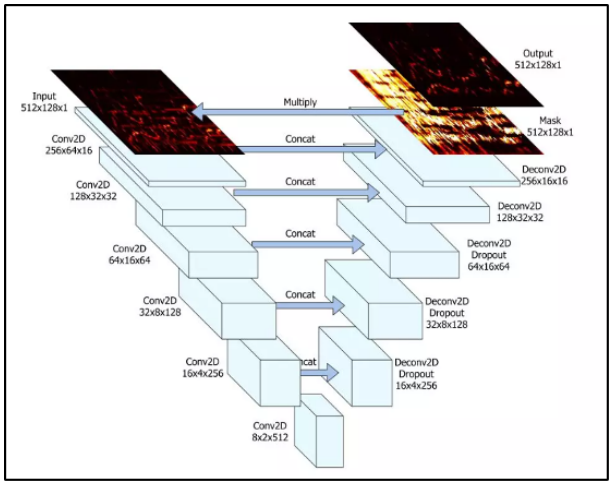

Figure 1.5: U-net architecture (Young, 2022)

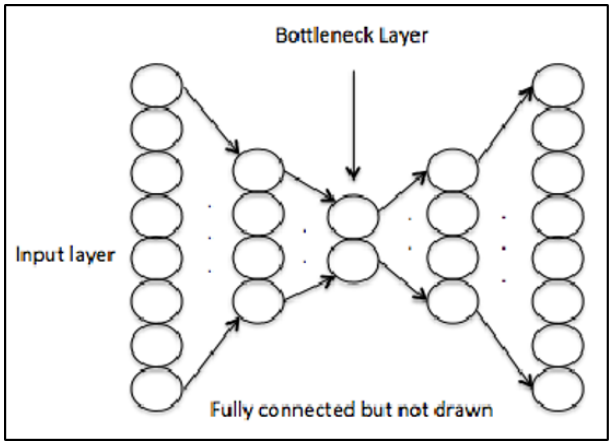

Figure 1.6: Bottleneck architecture (Lerch and Zhou, 2015)

1.3.2 Demucs Hybrid Architecture

Spleeter only focuses on spectrograms, whereas Demucs also takes waveforms into account. Rouard, Massa and Défossez (2023) says its hybrid source separation model decides which domain (spectrogram or waveform) is best suited for use while allowing a combination of the two.

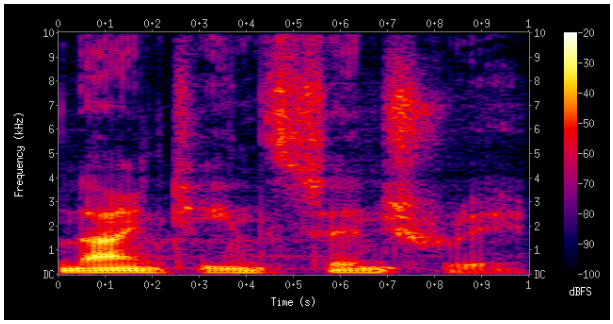

Figure 1.7: Spectrogram example (Wikipedia, 2024)

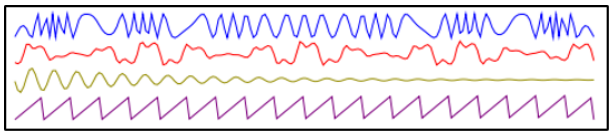

Figure 1.8: Waveform example (Rouard, Massa and Défossez, 2023)

Hybrid Demucs' multi-domain analysis mode is composed of a temporal branch, a spectral branch, and shared layers; it is a dual U-net structure with both branches having their respective skip connections.
The temporal branch takes the input waveform and contains 5 layers, while the spectral branch takes the spectrogram. After the 5th encoder layer, both branches have the same shape, which allows the sum of two aligned representations to be sent to the shared encoder/decoder layer, where its output becomes the input of the temporal and spectral decoder.

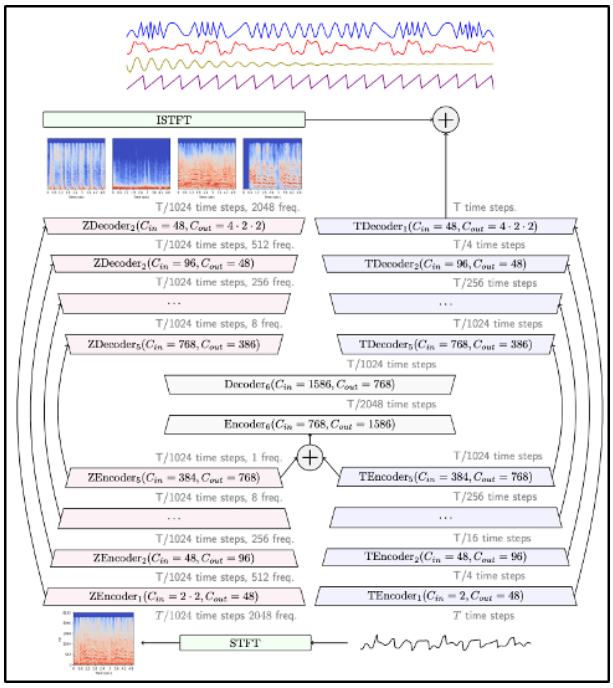

Figure 1.9: Hybrid Demucs Architecture (Rouard, Massa and Défossez, 2023)

Demucs performs poorly when trained solely on the musdb18-hq dataset, due to transformers being data hungry, so an internal dataset composed of 800 songs was trained on top. It is worth noting that training on Demucs used a pre-processed musdb18 dataset that adhered to certain rules. For example, songs were only kept where all four sources were non-silent for at least 30% of the time. For a 1-second signal, the signal is classified as silent when the volume is less than -40dB. (Rouard, Massa and Défossez, 2023)

1.3.3 Evaluation Metrics

Most source separation tools are compared and bench-marked on the musdb18 dataset, which contains *150* full length music tracks (all stereophonic, and encoded at 44.1kHz) of different genres along with their isolated drums, bass, vocals, and other stems. Its training set is composed of *87* songs, all encoded in the Native Instruments stems format (.mp4), which is a multi-track

format containing 5 stereo streams (Rafii, Liutkus, Fabian-Robert et al., 2017):

1. Full Mixture (Combination of all below signals)
2. Drums
3. Bass
4. The rest of the accompaniment
5. Vocals

Using a standard dataset for audio source separation models allows an equal and accurate comparison where reproducibility is then possible too, so other research can verify results.

The most commonly used evaluation metric is the Signal-to-Distortion ratio (SDR), a quantitative measure of how well an algorithm has separated the desired source from unwanted distortion, interference, and artefacts. Thus, evaluating the overall sound quality of the separation process. (Bhattarai, Pandeya and Lee, 2020)
SDR a unification of three other critical metrics:

- Signal-to-Interference Ratio (SIR) is the measure of how well the target source is isolated from interference or leakage from other sources. For example, when isolating vocals from a mixture, leakage of other instruments that cover the same frequency range are often present. A high SIR indicates that the isolated source contains little unwanted signals from other sources, whereas a low or negative SIR indicates that the level of interference from other sources is equal to or greater than the level of the target source in the separated output.
- Source Image-to-Spatial Distortion Ratio (ISR) measures how well the separated sources preserved the spatial characteristics and stereo positioning of the original signals. Spatial information refers to the positioning and localisation cues that allows humans to perceive which location and which direction a sound is coming from. This is created by differences in timing, level (loudness), and phase between the audio signals, captured by multiple microphones or channels. A high ISR indicates that the separated audio maintains the original spatial positioning and ambience of the original source, whereas a low ISR indicates that it has been lost or significantly distorted.
- Signal-to-Artefact Ratio (SAR) refers to the level of artefacts introduced by the source separation algorithm itself. A high SAR means that the separated isolation has very few audible artefacts or distortion, whereas a negative SAR indicates that the level of artefacts introduced by the separation process is greater than the level of the original source signal.

SDR is calculated using the formula in figure 1.10; a higher value means a less distorted signal, suggesting better audio quality.
Usually, an overall SDR is an equal average of all instruments. However, in some cases, certain instruments are of more importance and therefore should be given an increased weight amongst others.

$$SDR_{instr} = 10\log_{10}\frac{\sum_n \left(s_{instr,left\ channel}(n)\right)^2 + \sum_n \left(s_{instr,right\ channel}(n)\right)^2}{\sum_n \left(s_{instr,left\ channel}(n) - \hat{s}_{instr,left\ channel}(n)\right)^2 + \sum_n \left(s_{instr,right\ channel}(n) - \hat{s}_{instr,right\ channel}(n)\right)^2}$$

$$SDR_{song} = \frac{1}{4}(SDR_{bass} + SDR_{drums} + SDR_{vocals} + SDR_{other})$$

Figure 1.10: Signal-to-Distortion ratio Equation (Mitsufuji et al., 2021)

1.3.4 Existing Benchmark Tests

Sony held a Music De-mixing Challenge (Mitsufuji et al., 2021) where many source separation models were bench-marked against the musdb18-hq dataset, an uncompressed version of musdb18. (Rafii, Liutkus, Stöter et al., 2019)

Table 1.1 shows Demucs v4 (hybrid transformer model) clearly beats Spleeter in SDR, suggesting it produces more accurate results according to these mathematical equations. However, there is no consideration of perceptual evaluation or the performance and efficiency of models. For many use cases, computation time is just as important.

Model	SDR (Song)	SDR (Bass)	SDR (Drums)	SDR (Other)	SDR (Vocals)	Songs Trained
Spleeter	5.9	5.5	6.7	4.6	6.9	25,000
Demucs v4	8.8	9.8	10.0	6.4	8.9	800

Table 1.1: Comparing SDR values of Spleeter and Demucs v4 (Audioshake, 2023)

1.3.5 Real World Application

1. ***Mixing and Mastering***: Audio source separation technology has become increasingly accurate in recent years. It has allowed artists and major record labels to create stems of tracks that were missing or even from live recordings that require mixing and mastering for audio enhancement. For example, 'The Libertines', a popular English rock band formed in 1997, had a remaster of their albums. Rough Trade Records wanted to release a remaster of their albums but no longer had any project files or stems. Instead, they used AudioShake to artificially create them. (Audioshake, 2024)

2. ***Localisation***: Background noise and music can interfere with Audio Speech Recognition (ASR) transcription, which are components in many larger systems. Audio source separation technology could extract a cleaner dialogue stem from the background music before passing it on to the ASR system, resulting in an increase of the ASR's transcription accuracy.

3. ***Musical Instrumental Detection***: Several applications require musical instrumental detection. For example, Digital Service Providers (DSPs) like Spotify, manually ask artists what instruments have been used in their song. For their specific use case, this is to improve playlist curation; Spotify can find songs with only guitar and vocals to consider their placement in an acoustic playlist. Future updates and new systems can utilise a more automated approach to improve customer experience and workflow by utilising audio source separation.

Chapter 2

Methods

2.1 Overall System Aims

The overall system aim is to provide the user with a simplistic system to separate sources of a music piece using Spleeter and Demucs and be given evaluation data on how accurate each source separation was. It is then up-to the user to decide whether these widely used measures provide a good justification to a model's accuracy. This is achieved by a front-end user interface that visualises the data through radar charts and tables, as well as an audio player for each source to enable the user to perceptually evaluate its accuracy by manually comparing the produced stems side-by-side (A/B test them against the clean signal).

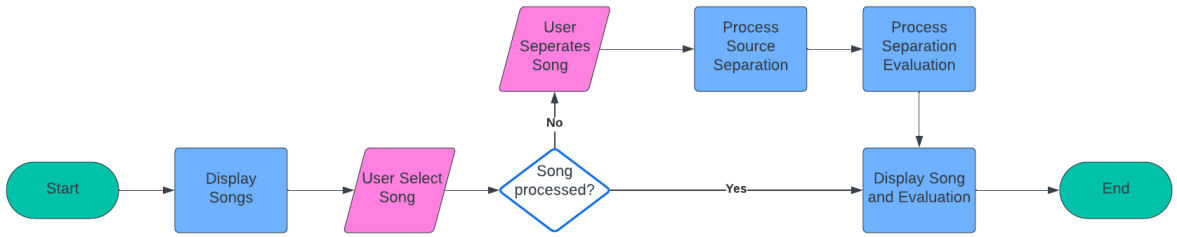

Figure 2.1: Flowchart of application procedure

2.2 Architecture

The application is based on a standard 3-tier architecture.

1. **Data Layer**: Data is stored in an SQLite database, which offers easy integration with Flask APIs using SQLAlchemy. The application hosts modest data requirements: only short strings and floating-point numbers need to be stored, with no relations. This ties well with SQLite's lightweight and fast engine.
2. **Application Layer**: The Flask API handles all back-end services: Audio processing, API endpoints, and Database integration.
3. **Presentation Layer**: A React front-end allows users to browse songs and visualises their evaluation data. React was chosen due to its ecosystem and community; there is a vast amount of libraries and tools to provide an efficient use of components and state management.

2.2.1 Data Layer

The data produced when evaluating both models separation accuracy is stored in an SQLite database, where all metrics are stored for each song. The database has columns for metrics:

SDR, ISR, and SAR for each instrument stem: drums, bass, vocals, and other.
Songs are queried by song title; if the scope of the project was to increase, the unique song ID would be used, as it is possible that songs with equal titles exist in the system.

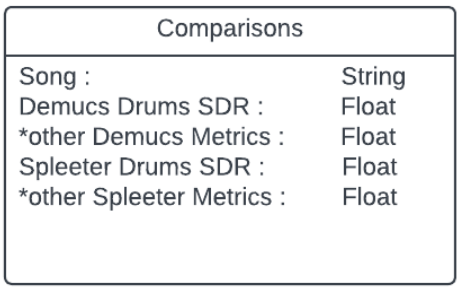

Figure 2.2: UML diagram of evaluation data

As the amount of songs was small (17 songs), a local file system was sufficient for storing all the audio files. Like above, if the scope was to increase and a larger set of songs was required, the file storage system would possibly need to be reconsidered, potentially transitioning to an online file storage solution. This is because for each song, both models produce four source separations, and the system extracts four unique sources from the original track. Collectively, for each song, a total of 12 audio files of equal length are created, which can quickly accumulate and consume a significant amount of storage space.

2.2.2 Application Layer

The project initiated with creating the back-end API. This was to ensure the backbone of the project was feasible before putting any efforts into a front-end, as it was completely dependent on successful implementation of both models and their evaluation.

The student decided to develop the API using the Flask API micro web framework, as they already had experience using it. Development was broken down into the application's individual features. The specific ordering of these tasks depended on their complexity, criticality, and dependencies. Further allowing the student to ensure that more complex tasks and challenges were handled first, so that critical components were functional within a good time frame. The later tasks were comparatively simpler. Leaving those until the end provided a safeguard: if any issues arose that made the project infeasible, the student would have more time to rework the design or reconsider the approach before investing effort into the easier features.

Separating Songs

The API integrates both Spleeter and Demucs libraries to perform source separation. The user chosen song is passed to the separation function where it creates directories to output both model's separations.

To ensure end-to-end safety, validation checks are made at every endpoint. For example, if the song is not found, an error message will be returned, instead of calling the separation functions. In another case, if the system identifies that the audio file has already been processed, the system

then searches the database to retrieve the stored metrics for that song. This increases efficiency and reduces processing time as it allows the system to immediately retrieve the necessary information without needing to reprocess the audio.
Reading the database first is crucial, as both models processing times are dependent on the length of the song. Without it, a 3-minute song that has previously been processed and analysed would require approximately 90 seconds of processing. This check helps avoid redundant processing and improves overall performance.

Analyse Separations and Provide Key Metrics

The API makes use of bss_eval, a python package to evaluate source separation estimates. This tool calculates the common evaluation metrics discussed, allowing the system to compute SDR, ISR, SIR, and SAR values for each source.

Unlike the outputs of the separation programs, there is no individual file for each source of the original track e.g. “drums.wav”. As a result, the system needs to extract these sources from the multi-track file. Stempeg is python package that can achieve this (by reading stem format files). Figure 2.3 visualises how the “compute_metrics” function handles this process.

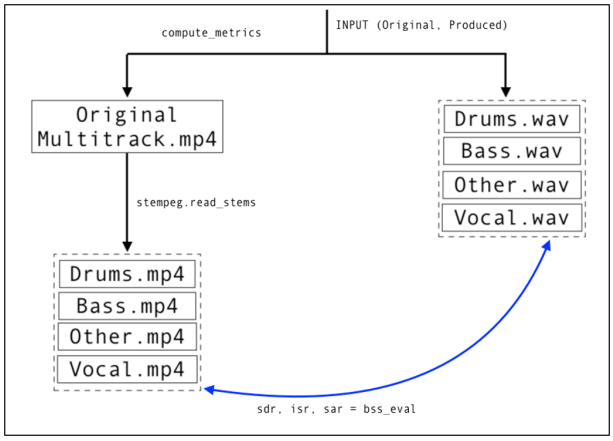

Figure 2.3: Diagram of how multi-track files are handled

It is critical to ensure that all signals are of the same length. Technically, equal lengths should be expected as the separated stem originates from the original mixture. However, as audio is represented as an array of floating-point numbers, there is room for potential errors. Floating-point rounding or other numerical precision issues can occur during the separation process, leading to a sample offset from the original. Bss_eval is used to compare frame-by-frame samples of the original isolation against the produced isolation. If the two samples are not synchronised, an incorrect evaluation will be made. Figure 2.4 shows a basic visualisation of two identical waveforms that appear different because of the sample offset, also known as out-of-phase. Ideally, the function should compare the sample-by-sample alignment for both stems, then determine the appropriate truncation or padding needed to be made. However, sample-by-sample alignment was not a priority and would take a long time to implement. Instead, the minimum length is found across all eight stems (drums, bass, vocal, and other for each stem) which is then applied to all other stems. This essentially truncates all stems to achieve equal lengths, assuming that the difference existed at the end of the song (or array of floating-point numbers).

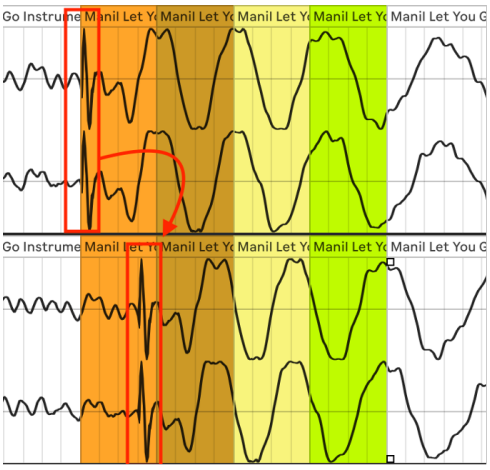

Figure 2.4: Identical waveforms with a small sample offset

Additionally, Bss_eval produces metric values for each point in the audio file, which is not desired for this project; one metric value for each source is required. The solution was to calculate an average of the entire floating-point number array by summing all terms and dividing by the number of terms, which gives the mean metric (e.g. SDR) for the whole duration of that stem.

Get Songs

The application uses a local folder of songs containing a mixture from the musdb18 dataset, as well as the student's creations.

For the front-end to display songs and relative information, it required the API to return an array of songs. The array contains each song's title, artist, duration, and evaluation metrics (if they have been processed).

As the application (both back and front-end) was hosted locally, the API was not able to pass an artwork or audio file path. React would require the object to be imported to display the artwork image. Though the project had a small amount of songs, meaning a small amount of imports, it would still be inefficient. Adding or removing songs would require frequent updates to the code-base. Instead, the Flask API has endpoints that can send local static files from directory, which is used for both song artwork and audio.

Sending audio of the original audio file worked different. As discussed earlier, the file is in a multi-track format, meaning there is no separate file for each instrument stem. The solution was for a dynamic endpoint that requires the song title and instrument name in url-encoded format. When called, the server would extract that layer from the multi-track file, save it in a local directory, and finally send the audio file as usual.

2.2.3 Presentation Layer

The final stage of the project was developing the front-end on React. Only two-page layouts were required, one for the main browser of songs, and the other to view an individual song.

Rawg (2023) was used as inspiration for the main browser page; the user interface took a similar, minimalistic approach by displaying a grid song cards filled with their artwork. Processed songs

display each model SDR value on their card, which can be sorted through along with other attributes (e.g. sort songs by ascending duration).

A light and dark theme is also available, enabled via a colour switch. The chosen theme remains consistent across other components of the application, further improving user experience and accessibility.

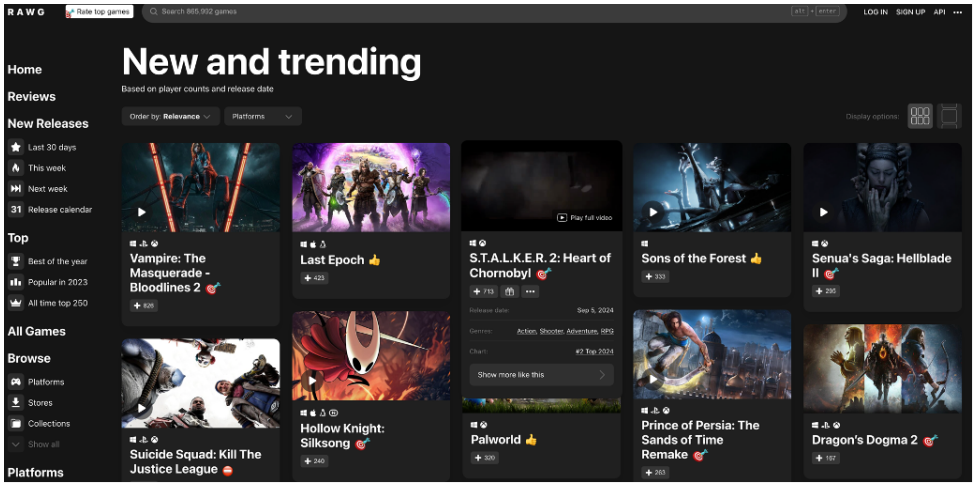

Figure 2.5: Video game browser user interface (Rawg, 2023)

As discussed earlier, audios are visualised via waveforms, similar to DeMiXER's application in figure 1.1. This project utilised wavesurfer, an open-source audio visualisation library for creating interactive waveforms (katspaugh, 2024). Waveforms allow users to visualise and expect audio information before listening; they can skip to timestamps where there is more audible noise, instead of listening to silence when waiting for the desired instrument to appear. This is a common issue for most stems as there can be large sections of silence when that instrument was absent in the original mix.

Figure 2.6: Waveform of isolated stem with long duration of silence

Evaluation data is visualised via radar charts as they provide comprehensive data representation using multiple data points on a single chart (SDR for drums, bass, vocal, and other). This allows the user to easily compare the performance of separation models across all components and identify their strengths and weaknesses in relation to the given metric. It is worth noting that radar charts provide trend identification too; this is important when comparing against several

songs. For example, seeing similar radial shapes for songs in different genres suggests that the model has a consistent generalisation across different genres, implying the model has learned a genre-agnostic relationship between target sources.

2.3 Project Management

Development was tracked using the Git version control system. Commits were left only when complete blocks of work was produced to maintain a more concise commit history, avoiding unnecessary commits for minor or incomplete changes. Smaller backups of the code-base were kept locally, especially for the front-end. This was due to development being individual, meaning online version control is less crucial as there is no need for collaboration.

The student decided to use agile methodologies for the development of this project. Specifically, feature-driven development was utilised, in which features of the application were developed through a series of iterative sprints. The order of feature development was carefully chosen as each new feature typically relied on the correct implementation of the one prior to it. For example, evaluating isolated sources required a correct implementation of source separation as well as ensuring all target sources were summed to the mix.

2.3.1 Sprint One: Set up Development Environment

The first sprint focused on setting up the core development environment and laying the groundwork for the application. This was achieved by first installing required dependencies and ensuring compatibility between them (libraries and frameworks), and then implementing a basic Flask API structure with temporary endpoints to ensure successful handling of incoming requests or data.

2.3.2 Sprint Two: Audio Processing and Source Separation

Sprint two entailed of adding core audio processing and source separation capabilities to the application. This ensured the application could read and return audio file metadata as well as separate audio files. This sprint was also used to test other available models that the Demucs library offers, such as the fine-tuned model, to decide whether these models should be taken into consideration.
Finally, sprint two ended with development of a client to interact with the API.

2.3.3 Sprint Three: Database and Evaluation Capabilities

This sprint focused on creating an evaluation feature. The evaluation tool calculates average values of SDR, ISR, and SAR across the whole song duration and then stores them in the database.

2.3.4 Sprint Four: Code Refactoring and Validation Improvements

After the main back end was completed, a short sprint was used to focus on cleaning up the code base by splitting larger, complex functions into smaller, more modular ones. This increased efficiency but more importantly, it made the code easier to understand for future amendments.

Testing and validation checks were moved from the client application to Postman. Several automated test scripts were developed to ensure error and data handling was as expected.

2.3.5 Sprint Five: Initiate Front-end Development with React

The following sprints focused on the front-end. Sprint 5 initiated this by setting up the development environment for React (utilising Vite) as well as creating a basic homepage component with Chakra UI (component library for React).

2.3.6 Sprint Six: Integrate Flask API and Fetch Song Data

Sprint six worked on integrating the Flask API with the React front-end to enable retrieval of a list of songs. The API was updated to extract song artwork from audio files, save them locally, and finally return image file. On the front end, hooks and Chakra UI card components were developed to retrieve data from the Flask API and to containerise and visualise songs respectively. This sprint was finished with a simple implementation of Chakra UI's light and dark theme.

2.3.7 Sprint Seven: Page Routing and Visualise Evaluation Data

Sprint seven worked on implementing dynamic routes, so each song's unique page can display their evaluation data. This involved designing and implementing a song page layout to display evaluation data through Radar Charts, Data Tables, and Badges, as well as the addition of the separate song feature.

2.3.8 Sprint Eight: Visualise Audio Through Waveforms

The final sprint worked on visualising audio sources through a waveform format (inside a cyclic carousel alternating between each source), and implementing a sort feature for the songs on the homepage. This required updating the API to return the songs list in a given order, if given any.

2.4 Song Data and Preparation

This project made use of a small portion of the musdb18 dataset which was introduced earlier on. Twelve songs were handpicked by the student to have a song selection that meets a variety of the following attributes:

- Genres / Cultural styles
- Duration
- Mixing quality (Clean / Messy)
- Male / Female vocals
- Instruments

This provided a diverse range of samples for testing and evaluating the source separation models. As discussed earlier, these audio files are already packed in STEM format; multi-track files contain each clean source signal in its individual audio layers.

The student expanded the song selection with a personal collection too. These songs were used as real-world examples of electronic dance records. However, songs from this collection had to be converted to STEM format using NativeInstruments (2016) STEM creator tool (see figure 2.7). Conversion was accomplished by bouncing mixed and mastered individual source signals using Ableton Live (Digital Audio Workstation, DAW), and loading them into the STEM creator tool. This tool compiles all four sources into a single file, including the necessary metadata.

When evaluating audio source separation models, it is important that all four source signals sum to the full mixture for both left and right channels. If not, the evaluation will be inaccurate as the model made separations based on the full original mixture, which will not be comprised of the exact sources it will be compared with.

Another crucial step when creating STEM files is to ensure that the full mixture does not distort or peak above 0db. Models will overestimate the loudness of instruments if the mixture clips or distorts, resulting in inaccurate results. To avoid this, all four signals were processed through a limiter, that had a ceiling of 0db, before being imported into the STEM creator.

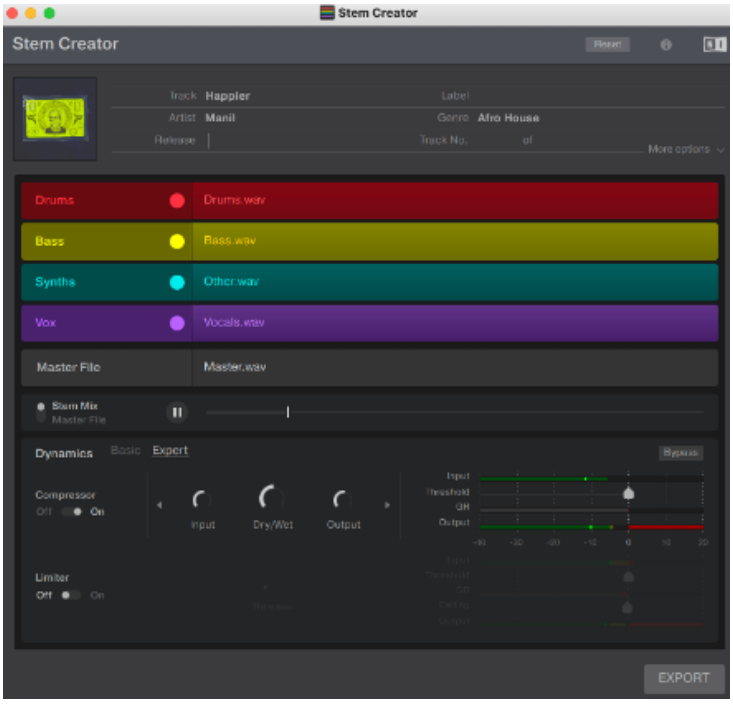

Figure 2.7: Native Instruments (2016) STEM creator tool

Chapter 3

Results

3.1 Testing and Validation

3.1.1 Testing API Endpoints

Originally, API endpoints were tested using a simple Python client that allowed the user to send GET and POST requests to the 'songs' and 'separate' route of the API. As the project went further into development, testing was moved to Postman.

Postman provides a centralised platform to execute API tests. With the ability to compare expected and actual data in an automated workflow using scripts within each Postman request, it streamlined the testing process. Postman also simplified the testing of routes that return images or audio files, as it did not require much additional code in comparison to the Python client. Additionally, Postman increased testing efficiency by automating tasks such as verifying data attributes for each song in the returned array, reducing the need for manual inspection.

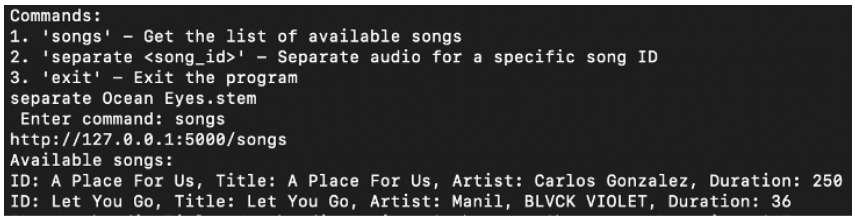

Figure 3.1: API Testing Client

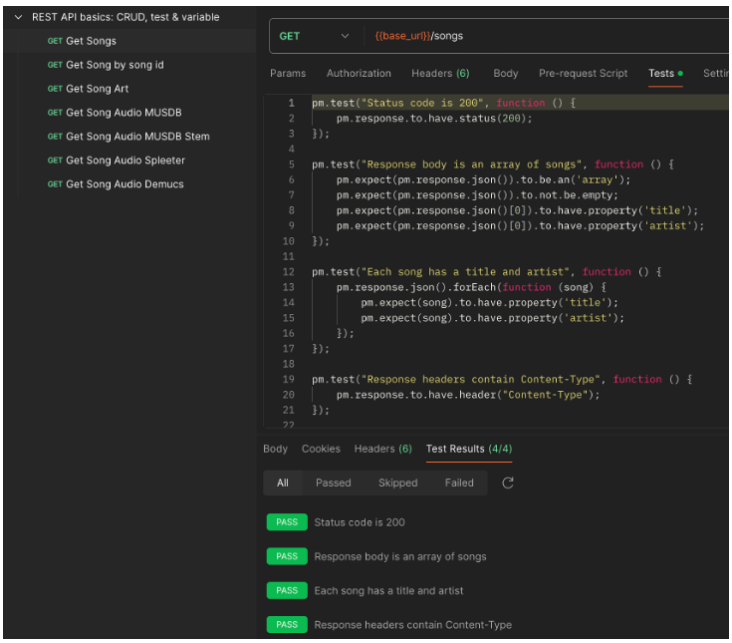

Figure 3.2: API Testing using Postman

3.1.2 Validating bss_eval Results

To ensure the validity of the results produced by bss_eval, results for several songs from the musdb18 dataset were compared with existing online data (an attempt to reproduce known results); Mitsufuji et al. (2021) provided SDR results for two phases of Spleeter and Demucs' separation on the musdb18 dataset. For most cases, the results were similar, confirming the accuracy of the evaluation library.

However, bss_eval consistently produced invalid results for SIR, with values of 'Nan' (not a number) or 'inf' (infinity) across all test cases. It is worth mentioning that efforts were made to work around this issue, as the SIR metric is of equal importance with the others. Approaches such as manually calculating the SIR were explored, but this presented additional challenges. Given the project's strict deadline and recognising the importance of the application to provide accurate results to uphold data integrity, the decision was made to exclude SIR.

3.1.3 Ensuring Sample Synchronisation and Equal Length

To assess whether sample truncation was the optimal move to ensure all three samples are synchronised and of equal lengths, five songs and their separations were randomly selected to manually compare their lengths side-by-side. Findings showed that all three stems were synchronised (figure 3.3). The different lengths existed when the original source duration was greater than the produced separation. This means the audio-to-array conversion or separation left out a small amount of information at the end (less than 0.001 of a second), further supporting the use of the end-of-signal truncation.

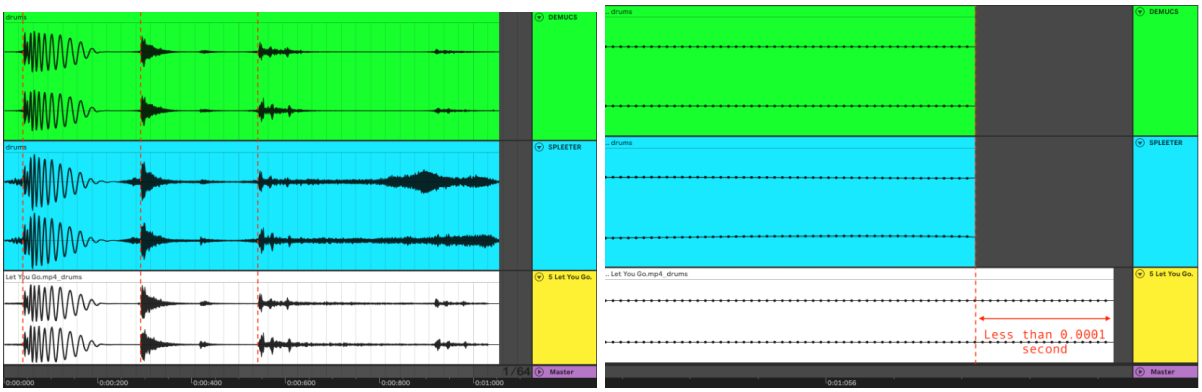

Figure 3.3: Synchronised Drum sources of Original (yellow), Spleeter (blue), and Demucs (green)

3.1.4 Ensuring Sources Summed to Full Mixture

Section 2.4 explains the importance of ensuring that the full mixture is the sum of exactly all four sources. This was achieved by consolidating all four sources, on top of each other, and comparing the grouped signal against the full mixture of the produced STEM format. For the student's personal collection, all songs were validated to make sure no instruments were missing from either the full mixture or the grouped four-source track.
Additionally, the full mixture and each individual source was played out for their entire duration to make sure their volume did not peak above 0db. This was to ensure that the limiter used on

creating each source and full mixture was effective. Figure 3.4 visualises the difference between using a limiter versus without, without producing 6.17db of peak distortion.

Figure 3.4: Difference of full mixture with and without limiting

The chosen songs from musdb data set were not initially validated for this project, as they are widely used and trusted in the source separation community. However, one song 'PR - Oh No' performed particularly bad; it received negative SDR scores for both Spleeter and Demucs, indicating that there was an issue with either separation or evaluation. After further investigation, it was found that the sum of sources did not add up to the full mixture for the left channel, resulting in inaccurate results. Consequently, this song was replaced with another musdb song. Another song from the musdb18 dataset had issues, "Heart Peripheral" partitioned a bass synth into the other source. The instrument is recognised as a bass, and therefore, both models obtained a low SDR score because it did not exist in the bass source in the original signal. This can be seen in the demo video in appendix C.

3.1.5 Other Models

The Demucs library has several other available models, such as the fine-tuned version of the hybrid transformer Demucs (v4). This model is trained on the same dataset as the normal Demucs V4 model; however, it was trained through 50 cycles (50 epochs) of the entire training dataset, as well as an increase in architectural depth. (Rouard, Massa and Défossez, 2023)
Source separation in the fine-tuned model should perform better; however, findings showed very little increase in SDR metrics (increases as little as 0.1) as well as an increase of processing time by approximately 5 times more in comparison to the normal model. A self-perceptual evaluation of the fine-tuned and normal version was made by the student, and no significant difference was found. Consequently, the project disregarded the use of the fine-tuned model.

Demucs also offers a six source version separating piano and guitar, while Spleeter can separate 5 sources with only the additional piano source. However, due to the lack of piano or guitar sources in the musdb18 dataset, neither model could be fairly evaluated. As a result, both models were excluded from consideration to ensure a consistent and unbiased comparison was made across all methods.

3.2 User Evaluation

To justify the effectiveness of the source separation evaluation and comparison tool, a set of users were asked to give feedback on the system. A mixture of participants was selected in order to receive feedback from more musically experienced users, with different musical backgrounds, to those with little to none. This decision was made for several reasons. Audio source separation tools are primarily used by technical and experienced users. They offer insights on the technical aspects, such as a better understanding of hearing spatial information. However, users with less experience often provide fresh, unbiased perspectives as they are less likely to have preconceived notions or expectations. Feedback from users with varying levels of experience can help identify potential areas where additional training or application assistance may be required. Their diverse backgrounds also identify accessibility barriers which can ensure the application is inclusive and user-friendly for people with different abilities.

Ten users were given a brief explanation of audio source separation and a rundown on how to use and navigate the application. They were tasked with separating three songs while being questioned on which model produced the better separation for each source, as well as whether they agree with the radar charts representing the songs SDR, ISR, and SAR. The user manual given to users can be accessed through the link in appendix C.

To minimise potential biases and ensure a consistent listening experience, all users were tested in a quiet environment and were given the same studio-quality audio equipment. Studio monitoring headphones are designed to reproduce audio accurately across the frequency spectrum. The monitors used (DT 240 PRO) has a frequency response of 5 Hz to 35,000 Hz (Beyerdynamic, 2017) which is more than enough to hear all frequencies that can be produced by either model. Most music lies within the 20 Hz to 20 kHz range, which is noted as the "audible frequency range" for human hearing.

Both the user evaluation questions and results can be found in appendix D.

3.2.1 User Feedback: Results On Evaluation Metrics

"Ocean Eyes" was the most user evaluated song, which was from the students personal collection. The song, alongside its separations, can be accessed through the links in appendix C. Tables 3.1, 3.2, and 3.3 represent the produced metrics by the compute_metrics function (higher results are better, marked green). Table 3.4 shows how long each model took to process the separation (lower is better).

The computed results show that Demucs performs better in nearly all realms, other than retaining spatial information and the obvious processing times. It is worth noting that some differences are so small that they may not be audible to even experienced listeners. Figure 3.5 visualises this minuscule difference via a radar chart; both models share a very similar radial shape. Despite this, candidates seem to have opposed thoughts. For example, although Demucs had a greater SDR value for the bass separation by nearly 0.5 more, 3/8 of users thought that Spleeter had a more accurate separation, and one user could not hear a difference. From an initial analysis, this could be due to Spleeter retaining spatial information a little better (+0.278); however, the

bass source for this particular song is a monophonic signal, meaning there is little to no spatial information. This is because monophonic signals combine all sound sources into a single channel without preserving their individual positioning or localisation cues. With this information, it is unlikely that candidates chose Spleeter, in this case, because of the higher ISR value.

There are many reasons why 50% of users disagreed with evaluation metrics and thought Spleeter made a better bass separation than Demucs. Objective metrics (SDR, ISR, SAR) are mathematical calculations that measure the quality of source separation based on the similarities and differences with the original signal. However, there is a large difference between objective metrics and perceptual evaluation, which is influenced by several subjective factors.
Musically experienced users may be better attuned to the nuances or characteristics of a bass signal. Despite the metrics, it is possible that Spleeter's separation captured these nuances more accurately.

Model	Drums	Bass	Vocals	Other
Spleeter	12.141	0.018	3.00	0.503
Demucs	14.633	0.517	3.448	1.600

Table 3.1: Signal to Distortion Ratio (Ocean Eyes)

Model	Drums	Bass	Vocals	Other
Spleeter	18.956	0.955	4.039	0.172
Demucs	18.293	0.677	4.014	0.441

Table 3.2: Source to Spatial Distortion Image (Ocean Eyes)

Model	Drums	Bass	Vocals	Other
Spleeter	12.520	7.386	0.250	-1.609
Demucs	16.008	13.706	1.697	-0.840

Table 3.3: Source to Artifact Ratio (Ocean Eyes)

Model	Processing Time (Seconds)
Spleeter	7.1
Demucs	17.4

Table 3.4: Processing Time (Ocean Eyes)

Figure 3.6 represents 30 song evaluations, divided equally by 10 users; over 80% of them favoured Demucs as the more accurate drum source separation. On many songs, the difference in quality was clear, especially in terms of maintaining transients and capturing the full frequency spectrum.

"Knockout" was a great example. The song's drums consisted of very short and transient elements such as the kick, snare, and closed hi-hat. It produced a drums SDR of 7 whereas Demucs got 14. However, it is important to understand that SDR does not solely focus on transients. Transients are short duration high-amplitude events, for example, the initial attack of a drum hit or a plucked guitar string. Spleeter seems to have a pattern of losing the original source's transients during separation. There are a couple of possible reasons for this.

Transients are usually characterised by their high-frequency content and rapid onset (figure 3.7). As detailed earlier, Spleeter disregards all frequencies above 11KHz. This means any frequencies

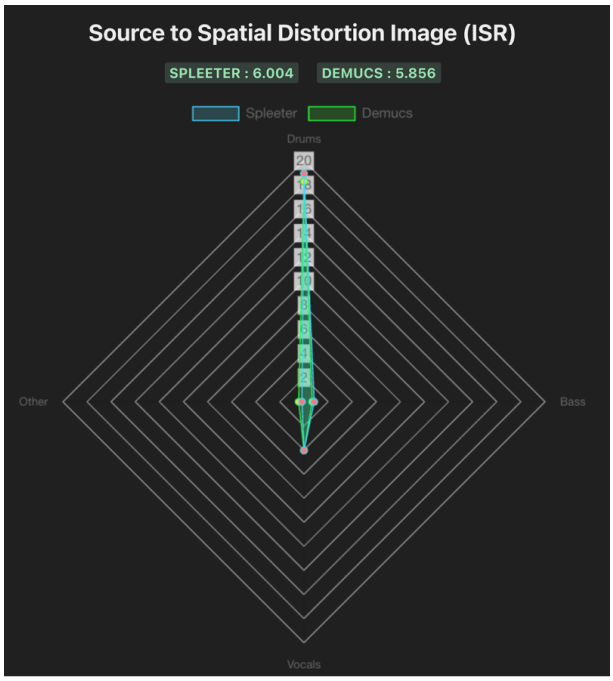

Figure 3.5: Radar Chart of Source to Spatial Distortion Image (Ocean Eyes)

that lie from 11Khz to 20Khz is lost, which is usually where the short 'click' of a sound lies. Additionally, high frequencies are said to give "life" to a music piece by serving as a lush top-end that can provide clarity and energy. Focusing on the drums source, cymbals, and other high-end instruments are less effective because of the frequency cutoff; in some cases, they totally disappear.

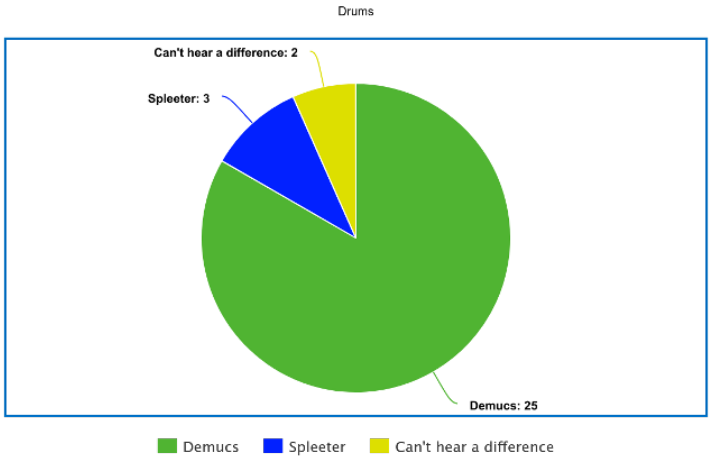

Figure 3.6: Pie chart of drum source user evaluation

3.2.2 User Feedback: Front-end Usability

User testing proved efficient for receiving valuable front-end feedback.
Navigating the application was relatively easy for most users; 80% of users thought browsing songs and completing separation tasks were simple, with the other 20% expressing its medium difficulty. There are a couple of main reasons why certain users have struggled with this.

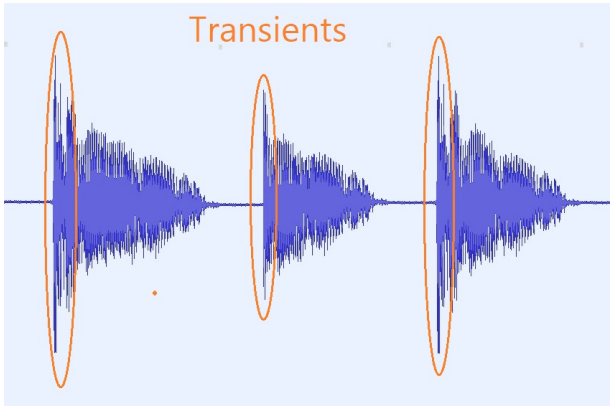

Figure 3.7: Visual representation of what transients are (majormixing, 2023)

Several users mentioned how small the arrows to alternate through graphs and audios were (shown in figure 3.8). The intention for its size came from other user interfaces, which were used as inspiration, as well as the initial goal for the 'minimalistic' design. It is clear from these results that the sizing and positioning of certain components needs reconsideration, especially to cater to users on larger devices or to the visually impaired.

Another repeated enhancement by users was to label each audio track based on which model they represent. This was considered during development; however, the decision was made to use a consistent colour scheme instead of labeling all components. This is because Graphical User Interfaces (GUI) can appear cluttered when many components are labelled; an interface is much more efficient when the user innately knows what a component represents.
Throughout the application, Spleeter is represented as blue and Demucs as green; a key symbolising this is also shown above the radar chart. Despite these attempts, as well as clarification of which colour represents which model within the user manual, it is clear that the waveform components are exceptions to 'label cluttering' and do require labelling.

Radar charts did not represent the data as expected. Figure 3.9 shows how 40% of users thought that they were not efficient at representing a model's performance. Mainly due to the chart's lack of clarity when certain metrics were close together between models. It successfully represents both model's similarities in this case, however there is no way to zoom in the graph to see which model performed better; the points would appear to lie directly on top of each other.
Additionally, data labels on the radar chart are difficult to read on the dark theme, due to the dark grid colour being similar to the page background (figure 3.10). As a result, many users became confused about which source was being observed, during the testing process, further requiring the student to intervene and explain.
Other charts may have provided better insight into the evaluation metrics. For example, separate bar charts for each source would show a distinct difference for metrics that are similar for both models, as each chart can have different axis values. This was not possible with the radar chart; if one source had much larger values to another, for both models, it becomes hard to visually distinguish the difference, as all sources are represented an a single broad axis.

One of the more musically experienced users suggested a promising feature change. They commented on how the click-to-play waveform creates a couple seconds of idle time when alternating

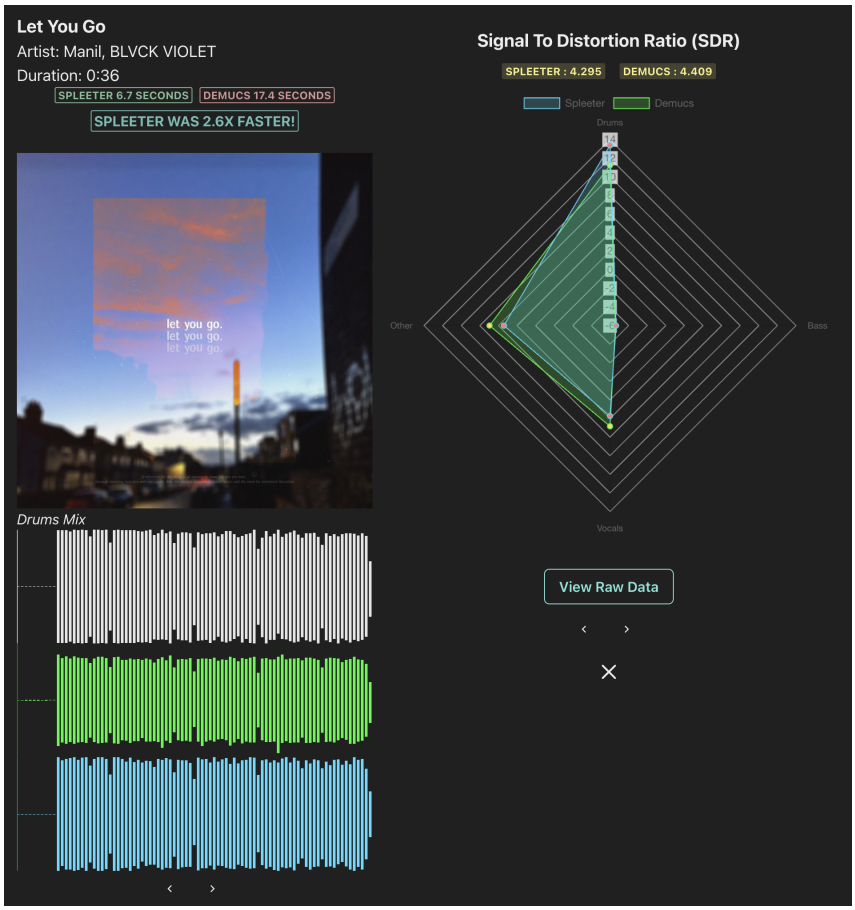

Figure 3.8: Application's individual song view

between models, and suggested that, "it would be nice to be able to synchronise the playback and then choose to isolate one of the options". This would provide a much better listening experience for the user, as there would be no idle time in between selecting models. More importantly, it would allow users to better evaluate models as comparing them from different start times can cause bias.

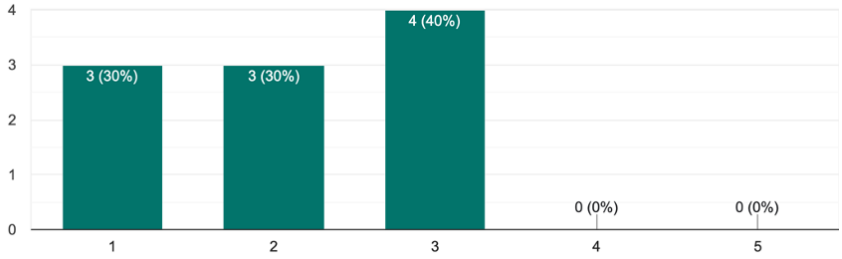

Figure 3.9: Bar chart of radar chart efficiency (1 - Very efficient)

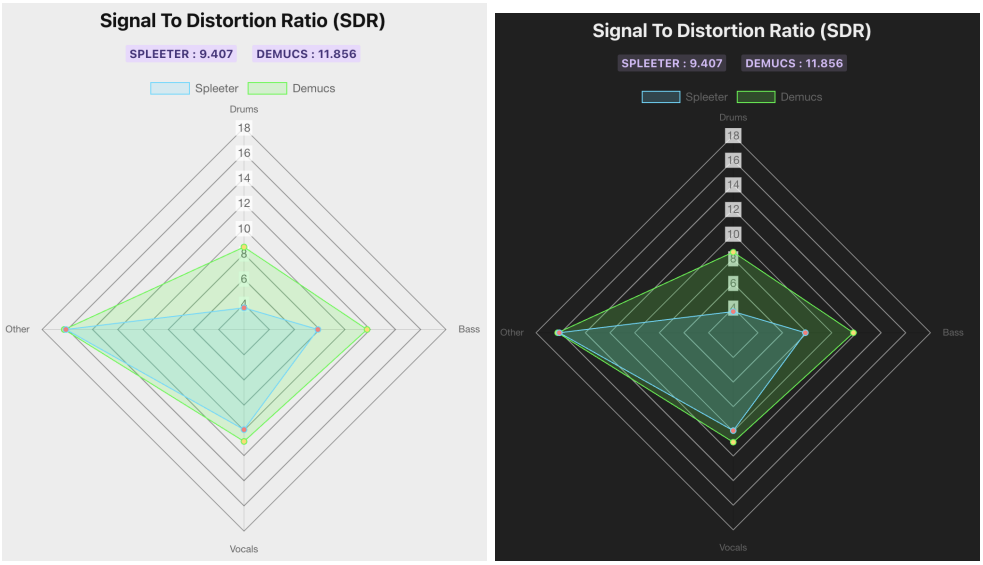

Figure 3.10: Radar chart clarity difference between light and dark theme

3.3 Trends

3.3.1 Demucs Bass Artefacts

Manually comparing both model's bass separation, it seems that Demucs introduced higher frequencies that were not present in the original. Figure 3.11 shows the frequency spectrum for the bass source separation produced by Demucs and the frequency spectrum of the clean bass signal (shaded in red).

Demucs produced high amplitude frequencies past 1.5KHz which are not visible in the original. In comparison, Spleeter's high-end frequencies (shown in 3.12) were produced at a much smaller amplitude, meaning they would be less audible and overall, more similar sounding to the original. These high-end frequencies are considered as artefacts as they are undesirable or incorrect components that are present in the separated sources but were not part of the original signal.

Table 3.3 states that Demucs has much less artefacts (-6.32 SAR) to Spleeter, which is somewhat counter-intuitive to the student's manual observation. This could potentially be due to a few reasons. As explained earlier, the SAR metric is a mathematical calculation; it may not perfectly align with the human perception of artefacts, especially for ones that lie in certain frequency ranges. Alternatively, while Demucs introduces these unwanted high-end frequencies, it could potentially perform better at suppressing other artefacts, such as interference or distortion, ultimately leading to a higher overall SAR value.

3.3.2 Spleeter's Loss of Transients

Manually comparing both model drum sources, figure 3.13 visualises how Spleeter seems to introduce a slow attack into a percussive element. This essentially ruins the transient that comes after, as the initial attack is weaker and more faded in. Some users, who considered themselves musically experienced, expressed how Spleeter's kick drum sounded like it had a preceding "whoosh," aligning with the student's observation.

This could be due to issues with the training data not having a strong emphasis on preserving

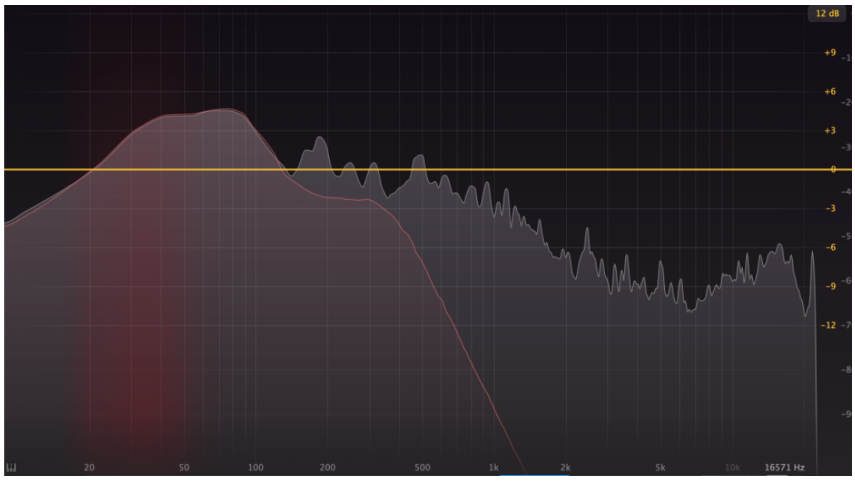

Figure 3.11: Frequency Spectrum of Demucs Bass Separation (Ocean Eyes)

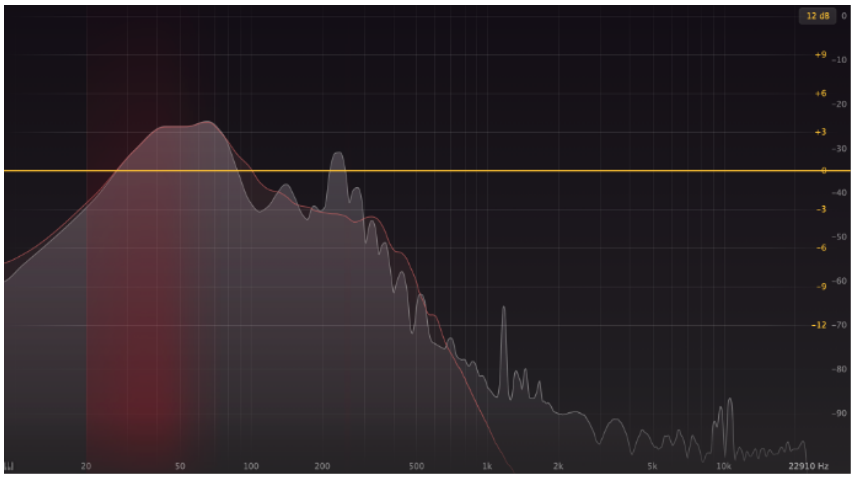

Figure 3.12: Frequency Spectrum of Spleeter Bass Separation (Ocean Eyes)

sharp transients. Alternatively, It is possible there is not much transient preservation due to spectral masking. Spleeter only uses spectrogram information as the input to its neural network. This leaves the possibility of transients becoming smeared or blurred due to the inherent trade-off between the time and frequency resolution in the short-time Fourier transform (STFT) used to produce the spectrogram.

While Demucs is advantageous over Spleeter in terms of preserving the sharpness of transient attacks, particularly for percussive instruments like kick drums and snares, this strength came at a cost. Demucs often produces an overly punchy or exaggerated separation, specifically for the drums source. A perceptual evaluation on Demucs' drums separations found that long decay hats or crashes became shorter, as if the model tends to discard or truncate the natural tails or decays of certain instruments.

3.3.3 Bss_eval on Bass Evaluation

Figure 3.14 shows a pattern of low bass metrics produced, specifically for the students personal song collection, where SDR values for the bass source were all negative. Many users agreed with the objective measures for the other sources, but disagreed with them for the bass, as they

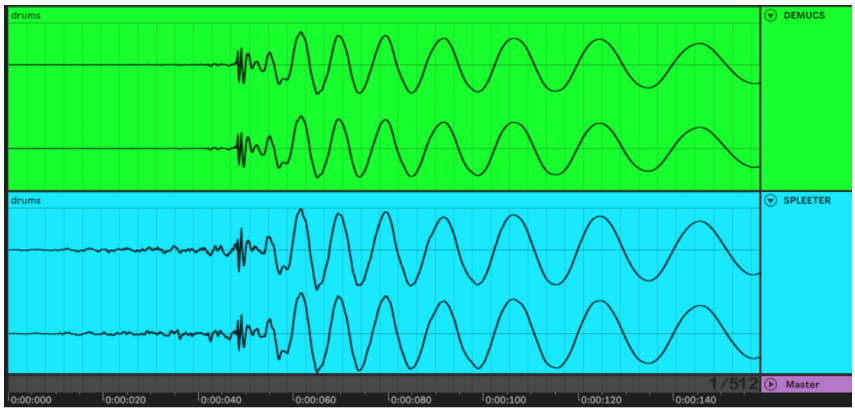

Figure 3.13: Percussive Kick Waveform of Spleeter and Demucs

did not believe that the distorted signal was louder than the desired signal (which a negative SDR represents). Conversely, some users seemed to form a biased opinion as the other source's objective measures seemed accurate, therefore, the bass ones must be too.

It is possible that both models find difficulties when separating loud electronic music. Songs from the student's personal collection were significantly louder than any of the chosen musdb18 ones. To put into perspective, "Heart Peripheral" is an electronic dance record in the musdb18 dataset. Its master produced a -13.6 peak Loudness Units relative to Full Scale (LUFs), whereas "Let You Go", from the student's personal collection, produced -6.7.
LUFs are a standardised measurement of audio loudness that factors both human perception and electrical signal intensity together, a much better loudness metric than peak db levels.
The linkage between the pattern of low bass metrics and loudness seems plausible, as distortions are more likely to occur when processing louder signals. However, if this is the case, it is concerning that only the lower frequencies are affected (bass) and not the higher ones, as 1.3.1 discusses how higher frequencies are more fragile to distortions.

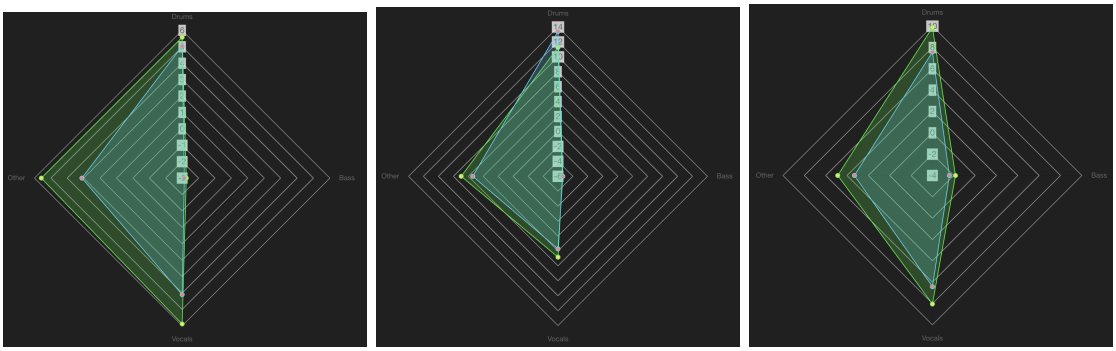

Figure 3.14: Examples of low bass SDR values for "Patadas de Ahogado", "Let You Go", and "Happier" respectively

Chapter 4

Discussion

4.1 Conclusions

This project clearly reveals the discrepancies in quantifiable evaluation metrics and their notable difference with a subjective assessment of the separated audio sources.
The key observation was the contradiction between the objective metrics and the perceived audio quality. While Demucs seemed to have exhibited a higher degree of audible artefacts, as evidenced in 3.3.1, it surprisingly performed better than Spleeter according to their SAR. This metric is a crucial component in the calculation of the SDR, which is widely regarded as the primary metric for comparative evaluation within the source separation community.
Furthermore, user testing revealed that certain artefacts, though quantitatively significant, were not that noticeable or disruptive to the listener.
These findings highlight the limitations of relying solely on quantitative metrics when assessing the accuracy of source separation models. Though they provide a standardised and objective measure of performance, they seem to fail to capture the nuances of human perception and the subjective aspects of audio quality. It would be beneficial for the community to incorporate both quantitative and qualitative approaches in the evaluation of audio source separation. Resulting in a more comprehensive understanding of the practical implications and real-world performance of these models, as ultimately, their goal is not to excel in numerical performance but to produce compellingly accurate separations for human perception.

4.2 Ideas for future work

3.2.2 Provided many improvements for the interface to become more user friendly, and in result, become a better application to evaluate audio source separation models.
The feature improvement that could bring the most benefit to improving the application's main function would be synchronising all waveform playback. This is where all three sources share the same timeline cursor and can be alternated or soloed by clicking either one.
Other improvements tailor more towards the user interface, such as labelling which waveform component represents which model, or enlarging the arrow buttons which cycle through the sources and graphs.

If the project's timeline was to increase, several other features could be implemented to further investigate each model's evaluation.
Updating the API to return bss_eval's raw data from the compute_metrics function would allow the front-end to visualise each metric at each timestamp or frame of the song. For example, it would be possible to see the overall SDR value at one part of the song e.g. the verse and compare it to a section where more complex instruments are being played e.g. the chorus. These quantifiable metrics are then able to help understand a model's performance at specific parts

of the song, which further helps identify trends in what context a model performs better in. Similarly, as discussed in 3.3.3, loudness (or signal energy) of the master may play a major role in the accuracy of audio source separation. Visualising the evaluation metrics at each timestamp alongside the mixture's loudness in LUFs would help recognise whether separation accuracy is directly correlated to loudness.

References

Audioshake (2023). *Compare Audio Models – AudioShake*. www.audioshake.ai.

– (2024). *AudioShake*. www.audioshake.ai.

audiosourcere (2024). *DeMIXER.com | Online AI Vocal Instrumental Remover - Free Demo*. www.demixer.com.

BCS (June 2022). *BCS, THE CHARTERED INSTITUTE FOR IT CODE OF CONDUCT FOR BCS MEMBERS*.

Beyerdynamic (2017). *DT 240 PRO Dynamic Headphone*.

Bhattarai, Bhuwan, Yagya Raj Pandeya and Joonwhoan Lee (2020). "Parallel Stacked Hourglass Network for Music Source Separation". In: *IEEE Access* 8, pp. 206016–206027. DOI: 10.1109/access.2020.3037773.

Computing Machinery, Association for (June 2018). *ACM Code of Ethics and Professional Conduct*. Association for Computing Machinery.

Grais, Emad M. et al. (Sept. 2019). "Referenceless Performance Evaluation of Audio Source Separation using Deep Neural Networks". In: *2019 27th European Signal Processing Conference* EUSIPC0, pp. 1–5. DOI: 10.23919/EUSIPCO.2019.8902932.

Hanssian, Sevag (2023). *free-music-demixer*. free-music-demixer.

Hennequin, Romain et al. (June 2020). "Spleeter: a fast and efficient music source separation tool with pre-trained models". In: *Journal of Open Source Software* 5, p. 2154. DOI: 10.21105/joss.02154.

katspaugh (Apr. 2024). *katspaugh/wavesurfer.js*. GitHub.

Kelly, Tom (2019). *2811 User Interfaces Lecture 7: Beautiful Visual Design*.

Lerch, Alexander and Xinquan Zhou (2015). *Chord Detection Using Deep Learning CHORD DETECTION USING DEEP LEARNING*.

Mitchell, Chris (Dec. 2022). *SpleeterGUI*. makenweb.com.

Mitsufuji, Yuki et al. (2021). *AIcrowd | Music Demixing Challenge ISMIR 2021 | Challenges*. AIcrowd | Music Demixing Challenge ISMIR 2021 | Challenges.

NativeInstruments (2016). *Stems is for producers*. Stems.

Prétet, Laure et al. (June 2019). *SINGING VOICE SEPARATION: A STUDY ON TRAINING DATA*.

Rafii, Zafar, Antoine Liutkus, Stöter Fabian-Robert et al. (2017). *MUSDB18 | SigSep*. sigsep.github.io.

Rafii, Zafar, Antoine Liutkus, Fabian-Robert Stöter et al. (Aug. 2019). *MUSDB18-HQ - an uncompressed version of MUSDB18*. Zenodo. DOI: 10.5281/zenodo.3338373.

Rawg (2023). *The Biggest Video Game Database on RAWG - Video Game Discovery Service*. rawg.io.

Rouard, Simon, Francisco Massa and Alexandre Défossez (June 2023). "Hybrid Transformers for Music Source Separation". In: *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. DOI: 10.1109/icassp49357.2023.10096956.

Wikipedia (Mar. 2024). *Spectrogram*. Wikipedia.

Young, David (Nov. 2022). *Audio Source Separation w/ Deep Learning*. David Young.

Zhang, Xindan and Haoyuan Zheng (Dec. 2023). "Signal Enhancement Methods Based on Wavelet Transform, Fractional Fourier Transform and Short-time Fourier Transform". In: *Highlights in Science, Engineering and Technology* 76, pp. 222–230. DOI: 10.54097/eyfndn07.

Appendix A

Self-appraisal

A.1 Critical self-evaluation

Discovering the difference between objective measures and perceptual evaluation between audio source separation models has been an interesting challenge. Overall, I believe that the project was executed successfully, despite the several obstacles endured along the the way. The majority of the project lied within the realm of unfamiliar territories for me, as I had no prior experience in deep learning, handling audio data, or developing web applications in React.

This project required me to undergo a steep learning curve, as it was difficult for me to understand the intricate design of Spleeter and Demucs' deep learning models; even towards the end of the project, I only understood them at a high-level. If given more time, it would have been beneficial to become more knowledgeable on this field, as it would have allowed me to put together more specific conclusions to the findings of the project, instead of giving a broad range possible reasoning.
Additionally, the development of the web application demanded learning React, a JavaScript library that was entirely new to me. I had prior experience in creating front-end web applications using Flask, although useful, it created a false sense of security leading me to underestimate the complexity of React. Initially, I thought its inner workings would be similar, however, I quickly realised that was not the case. Nevertheless, it was a difficult challenge to unlearn my previous approaches and adapt to a new way of thinking.

A planned timeline was created for the initial outline, however, I overestimated the amount of time I would have for the project during semester two. To add to this, the initial project idea was changed several times due to complexity and viability reasons. Only after many meetings with my supervisors did I have my final project idea, leaving only 2 months for development before the deadline.

A.2 Personal reflection and lessons learned

My prior experience in developing similar scaled web applications helped me when developing the back-end API. During its development, I was simultaneously learning about Django, in my Web Services and Web Data module. Django works similarly to the Flask API, which enabled me to implement the core functionality of the application in fewer days than expected. Because of this, I ended up reducing my planned timeline to make up for my initial overestimation. This eventually caught up to me, as learning and implementing the React front-end took longer than expected. This taught me the importance of using any additional days to my advantage instead of extending breaks and hard sticking towards a plan.

I did well to adapt to changes to the initial plan. Initially, my project was to create a front-end

GUI for Demucs, however, I had concerns that it was not complex enough for a final year project. To reach an understanding of the level of complexity required, I organised several meetings with my supervisors with a list of various ideas that I had interest in. With their advice and my own research, I weighed all my ideas against each other in terms of their complexity, viability, and personal interest. This was a skill that I learned in my Innovation and Thinking module, which was strengthened by its practice in this project.

This was my first project that utilised many different technologies. This taught me important skills, such as the ability to read papers on different technologies while being mindful of the similarities or differences between them. Development wise, it made me realise the importance of code refactoring and modularisation, as further into development the project became too complex; with the use of many different libraries, the project required a sort of abstraction.

As expected, the planned timeline was not completely followed. This forced me to learn to handle the stress of juggling multiple courseworks, exams, and other life's obstacles together. It is a skill required in a most professional settings and has been reinforced through the duration of this project.

A.3 Legal, social, ethical and professional issues

A.3.1 Legal issues

Legal issues during data collection in user testing was mitigated by ensuring each participant signed a consent form. The form outlined their role in the user evaluation as well as their right to be forgotten (or be given full anonymity).

In addition, songs from the student's personal collection included remixes of popular songs without proper copyright clearance. The project did not distribute the content publicly and was used solely for individual research purposes, therefore, it should fall under fair use or creative commons exceptions.

A.3.2 Social issues

The project implemented a dark and light theme to mitigate certain accessibility concerns. As a result, the web application caters to individuals with varying visual needs, enhancing usability for users with specific preferences.

It was noted in 4.1 that certain components were too small and therefore hard to see, especially for users with visual impairment.

A.3.3 Ethical issues

Training data sets for Spleeter and Demucs are possibly skewed towards certain genres, artists, or styles of music. This could lead to biased performance or inaccuracies when used on music outside of its training data set range.
Under-representation can occur if the training data lacks diversity and is heavily skewed towards certain types of music (e.g., predominantly Western pop/rock genres). Subsequently, the models

may perform poorly when separating sources from other genres or cultural styles.
Efforts should be made to curate training data sets that are diverse and representative of all styles, to acknowledge and mitigate any potential biases that may exist.

A.3.4 Professional issues

Development of this project followed the standard principles described by the BCS (2022) and the Computing Machinery (2018).
These ensure that software should respect the public interest and not discriminate on any factor. No claims to a higher level of competency to the students professional competence was made throughout the report.

In addition, the project was conducted in an academic setting without any direct involvement or feedback from professionals in the audio industry. Collaboration with professionals could have provided further insight into the challenges faced in audio source separation.

Appendix B

External Material

This project was written in Python and Typescript. Many external libraries, applications, and datasets were used to ensure the success of this project.
Below is a list of technologies used, separated by its location use. "Other" represents the datasets and applications used for testing and dataset creation.

Back-end:

- Flask-API
- Spleeter
- Demucs
- Bss_eval
- Stempeg
- Soundfile
- SQLAlchemy
- Mutagen

Front-end:

- React
- Vite
- Chakra-UI
- Wavesurfer
- React-chartjs-2

Other:

- Musdb-18
- STEM Creator
- Postman
- Ableton Live

Appendix C

Demo and Resources

Here is the link to the repository. Otherwise: https://github.com/manil1234/AudioSepComparev1

Here is a demo of the web application.

Here is a sample of three songs along with their separations. You will find both model's separations for: "Ocean Eyes" from the student's collection, "Knockout" from musdb18, and "Kaathaadi" also from musdb18.
Please email ed19mk3@leeds.ac.uk if access is required.

Here is a link to the user manual given to participants.

Appendix D

User Testing Results

Below lies the questions results obtained from user testing.

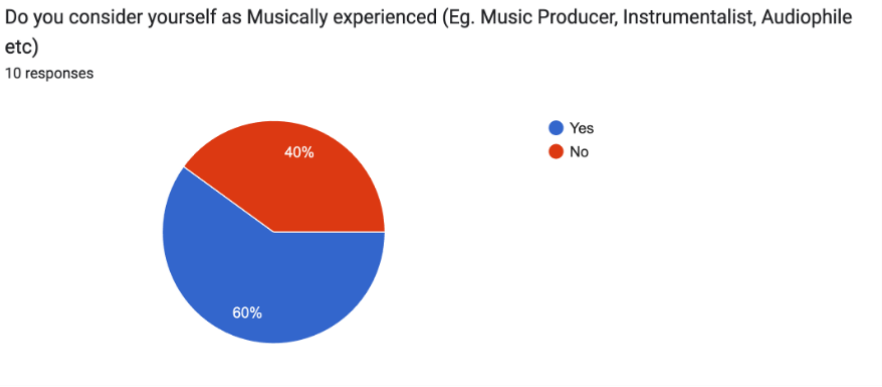

Figure D.1: Do you consider yourself as Musically experienced

Production experience and running music events
Guitarist
Djing
Producer
Musician
DJ

Table D.1: What musical experience do you have?

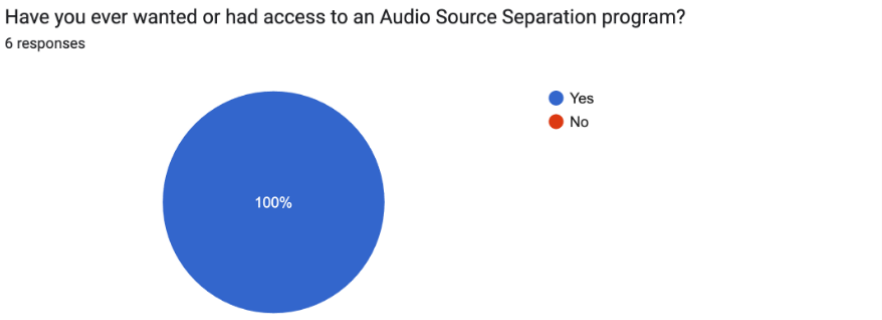

Figure D.2: Have you ever wanted or had access to an Audio Source Separation program?

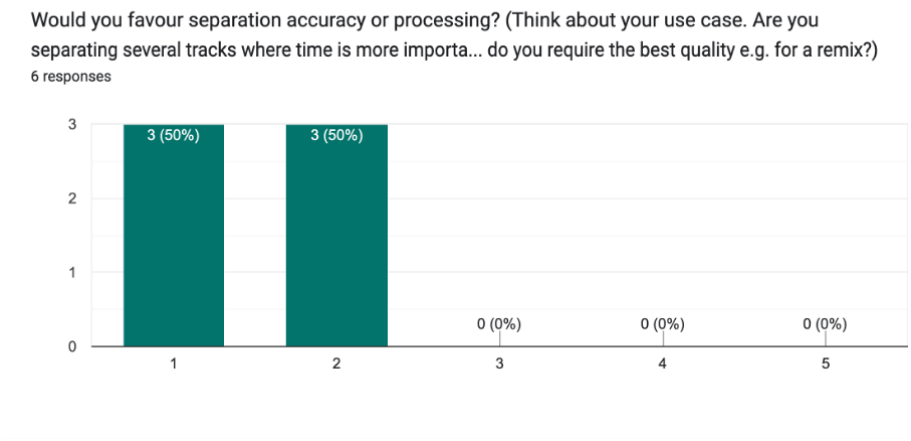

Figure D.3: Would you favour separation accuracy or processing?

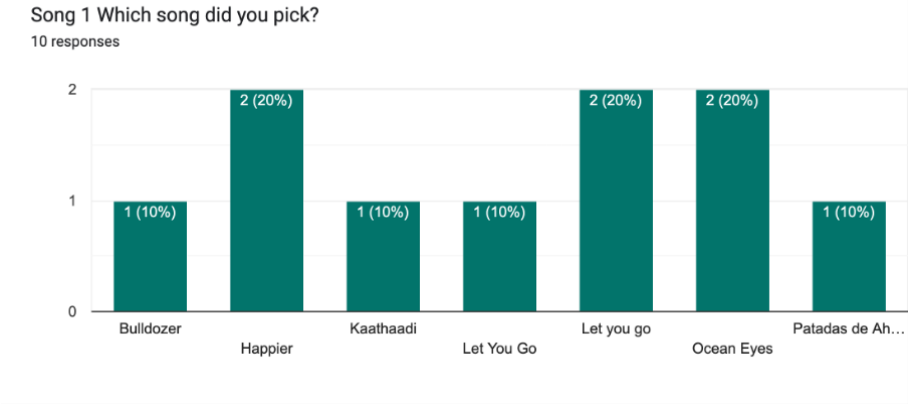

Figure D.4: Song 1: Which song did you pick?

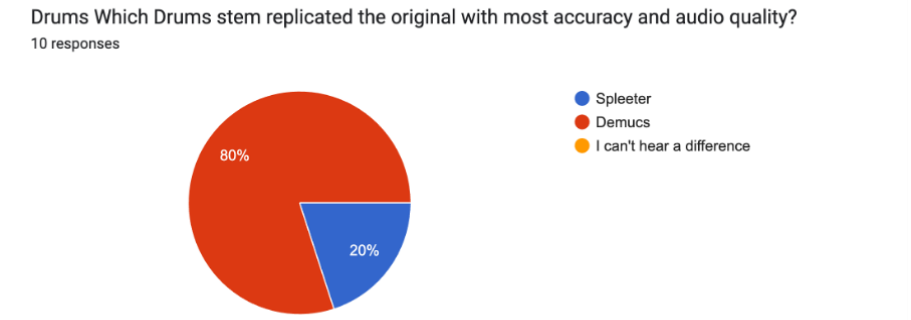

Figure D.5: Song 1: Which Drums stem replicated the original with most accuracy and audio quality?

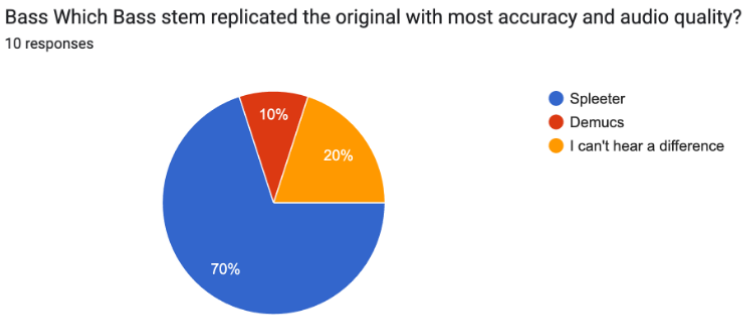

Figure D.6: Song 1: Which Bass stem replicated the original with most accuracy and audio quality?

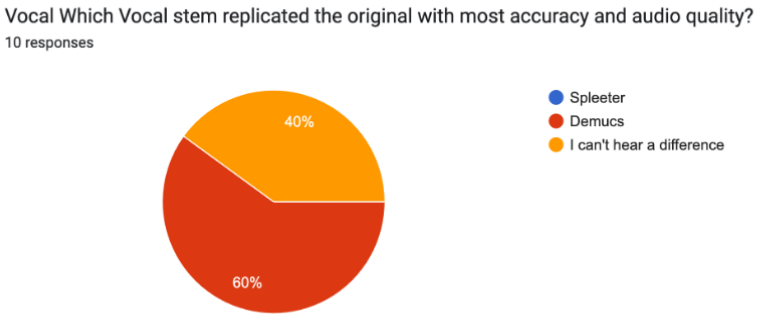

Figure D.7: Song 1: Which Vocal stem replicated the original with most accuracy and audio quality?

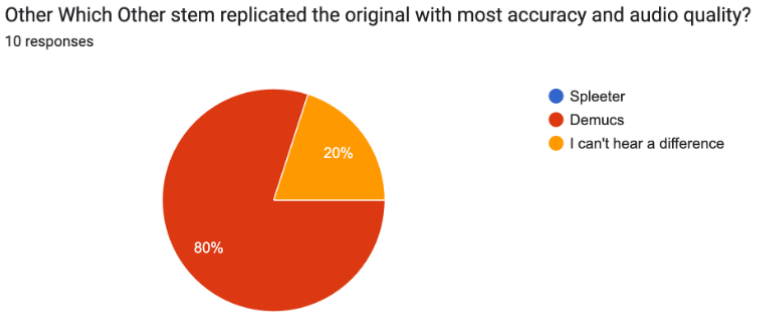

Figure D.8: Song 1: Which Other stem replicated the original with most accuracy and audio quality?

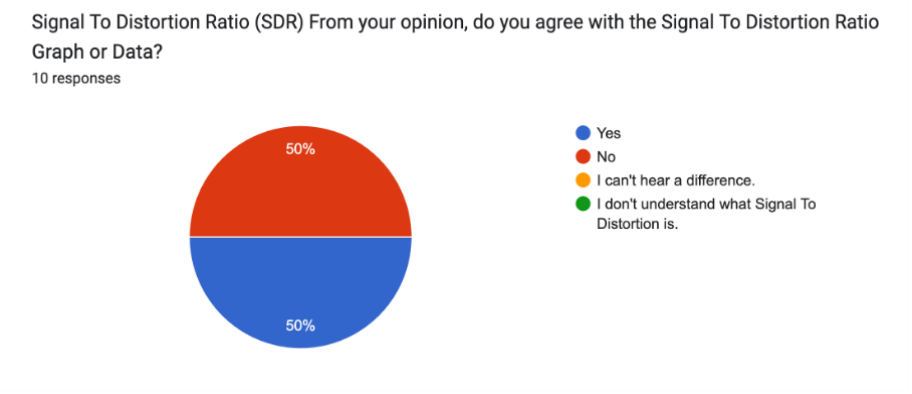

Figure D.9: Song 1: From your opinion, do you agree with the Signal To Distortion Ratio Graph or Data?

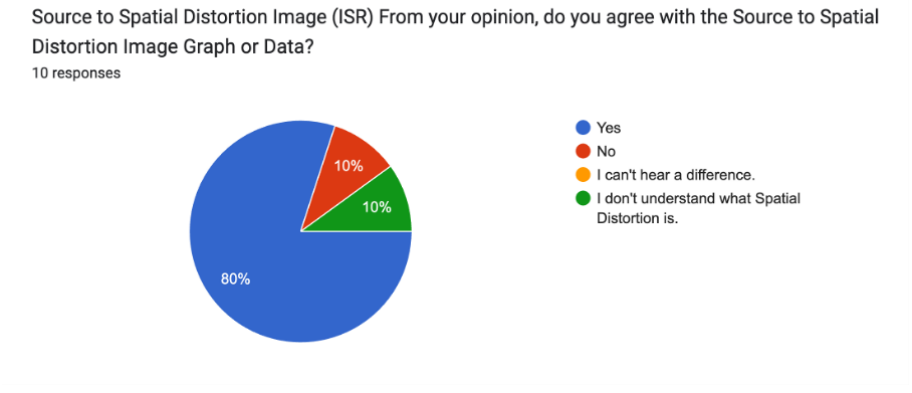

Figure D.10: Song 1: From your opinion, do you agree with the Source to Spatial Distortion Image Graph or Data?

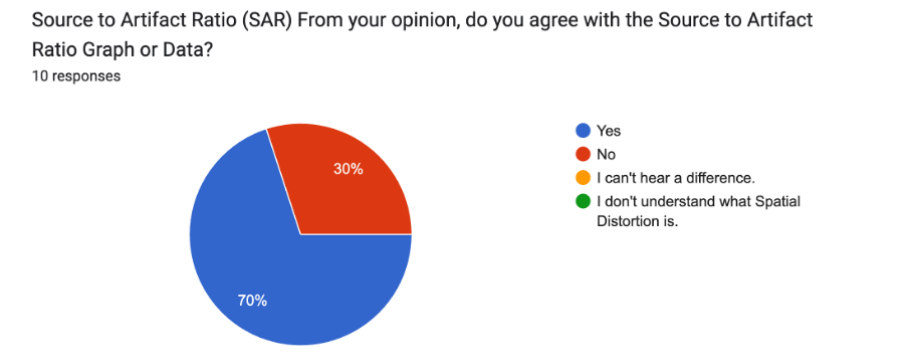

Figure D.11: From your opinion, do you agree with the Source to Artifact Ratio Graph or Data?

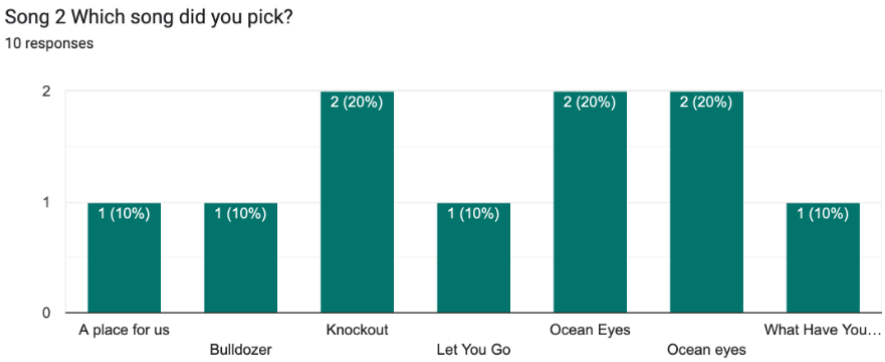

Figure D.12: Song 2: Which song did you pick?

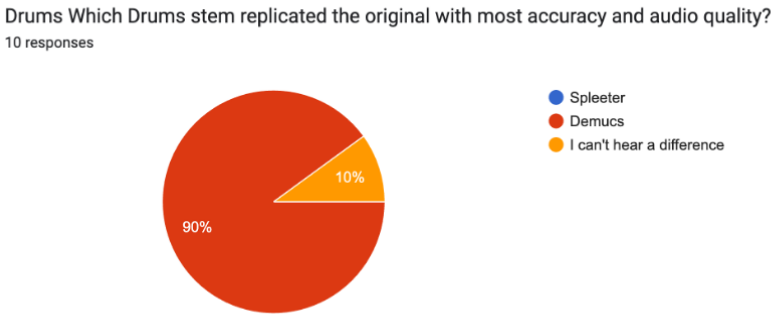

Figure D.13: Song 2: Which Drums stem replicated the original with most accuracy and audio quality?

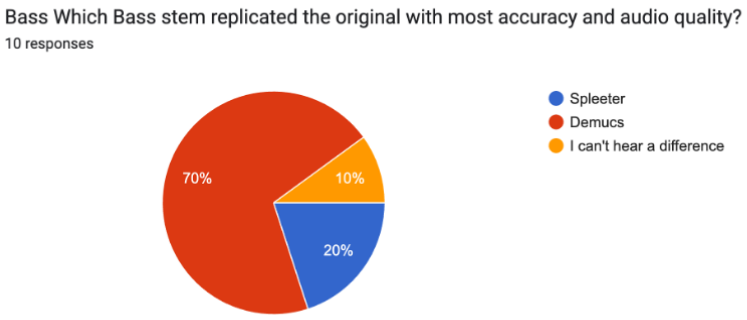

Figure D.14: Song 2: Which Bass stem replicated the original with most accuracy and audio quality?

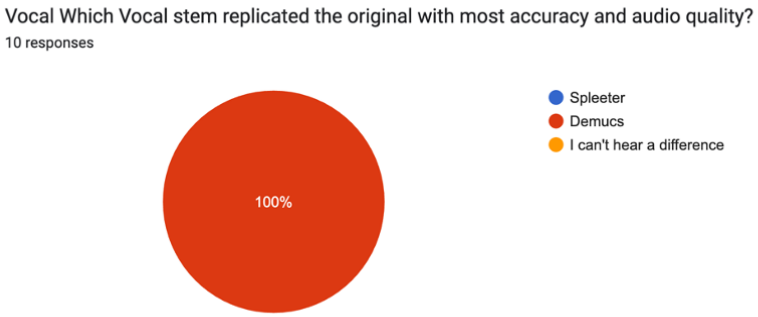

Figure D.15: Song 2: Which Vocal stem replicated the original with most accuracy and audio quality?

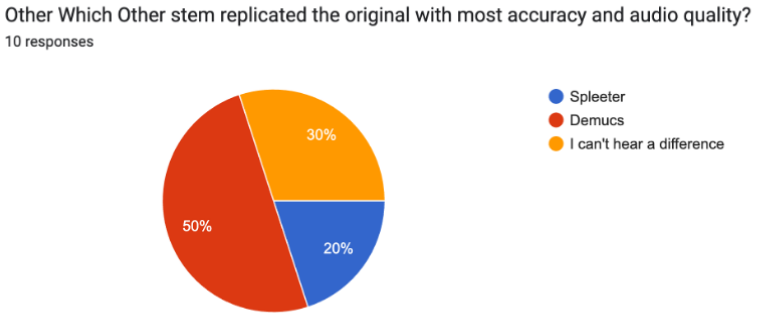

Figure D.16: Song 2: Which Other stem replicated the original with most accuracy and audio quality?

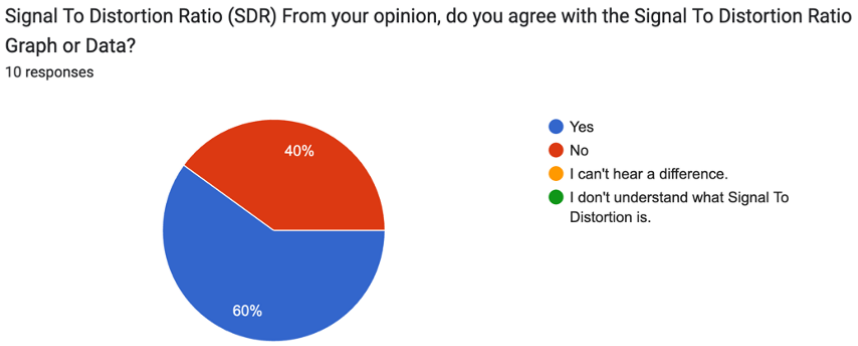

Figure D.17: Song 2: From your opinion, do you agree with the Signal To Distortion Ratio Graph or Data?

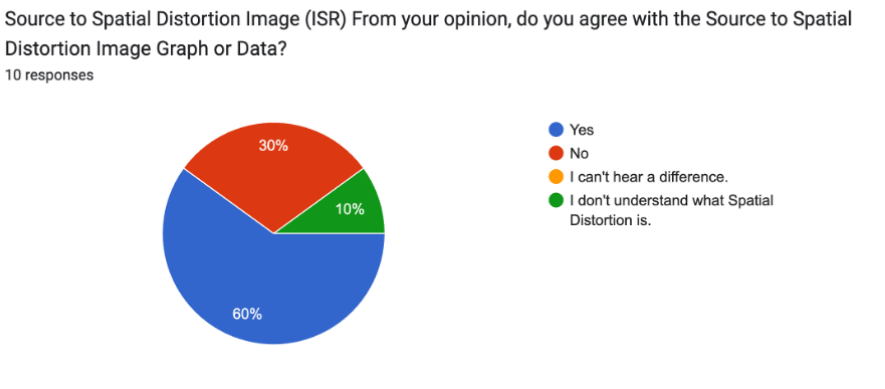

Figure D.18: Song 2: From your opinion, do you agree with the Source to Spatial Distortion Image Graph or Data?

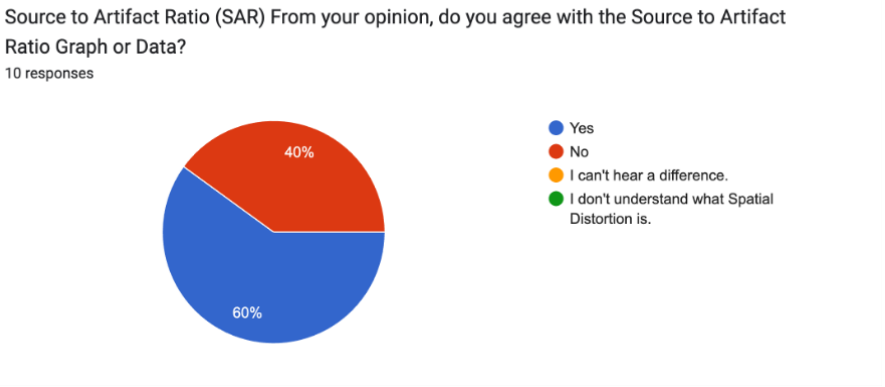

Figure D.19: Song 2: From your opinion, do you agree with the Source to Artifact Ratio Graph or Data?

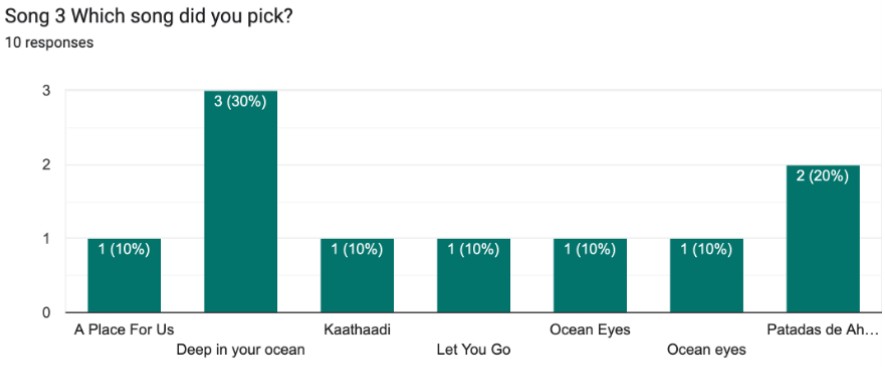

Figure D.20: Song 3: Which song did you pick?

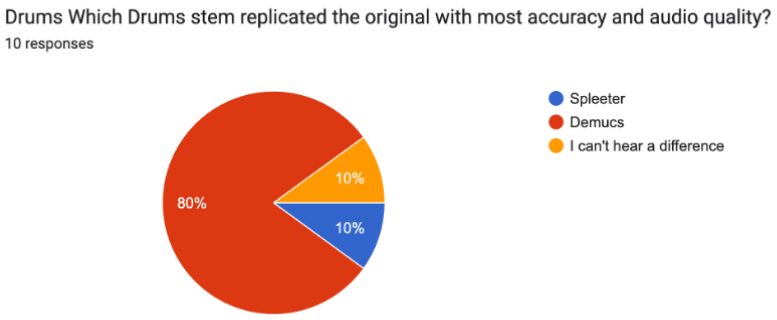

Figure D.21: Song 3: Which Drums stem replicated the original with most accuracy and audio quality?

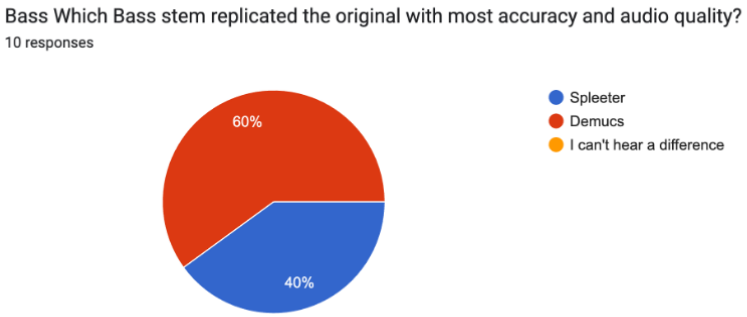

Figure D.22: Song 3: Which Bass stem replicated the original with most accuracy and audio quality?

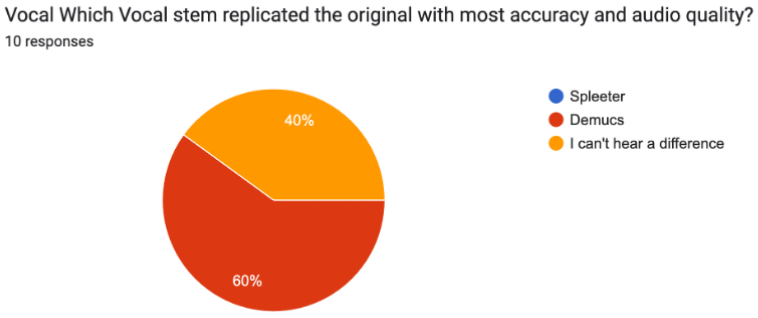

Figure D.23: Song 3: Which Vocal stem replicated the original with most accuracy and audio quality?

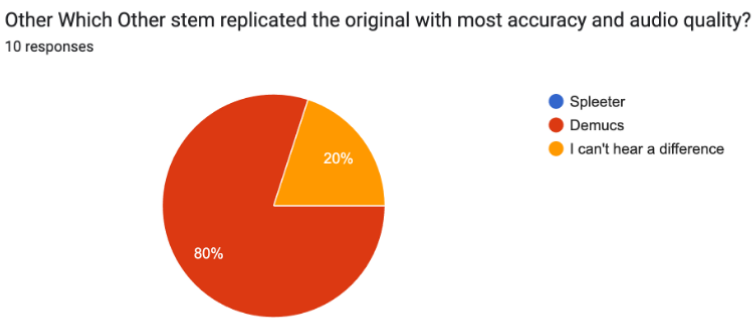

Figure D.24: Song 3: Which Other stem replicated the original with most accuracy and audio quality?

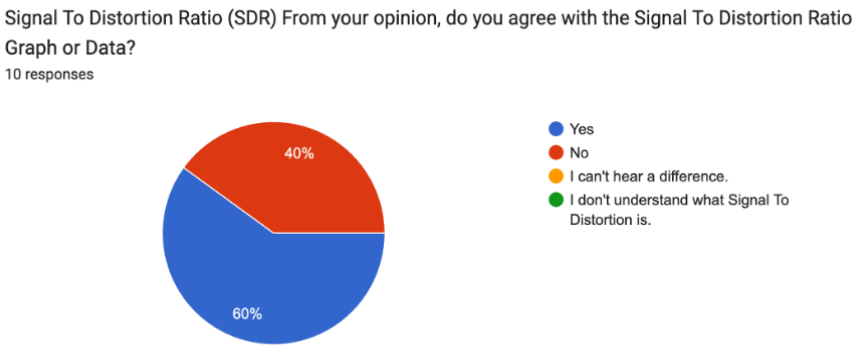

Figure D.25: Song 3: From your opinion, do you agree with the Signal To Distortion Ratio Graph or Data?

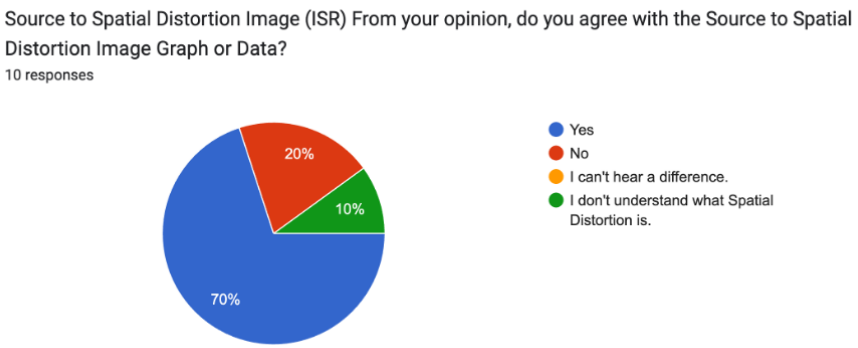

Figure D.26: Song 3: From your opinion, do you agree with the Source to Spatial Distortion Image Graph or Data?

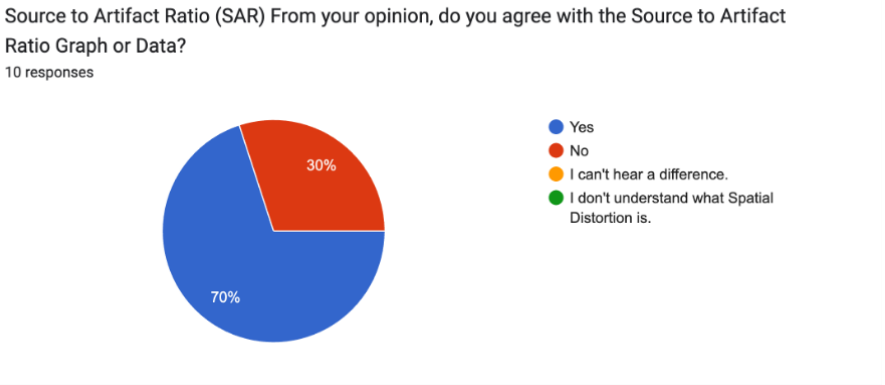

Figure D.27: Song 3: From your opinion, do you agree with the Source to Artifact Ratio Graph or Data?

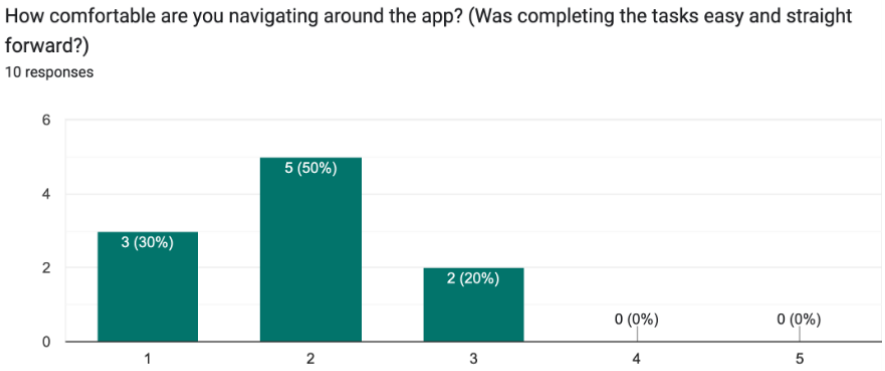

Figure D.28: How comfortable are you navigating around the app?

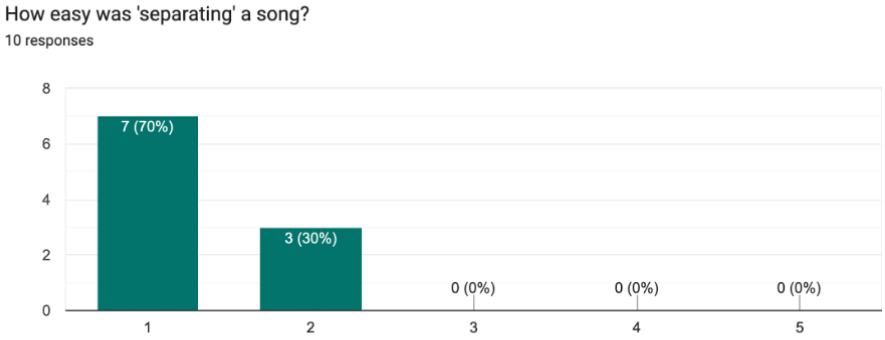

Figure D.29: How easy was 'separating' a song?

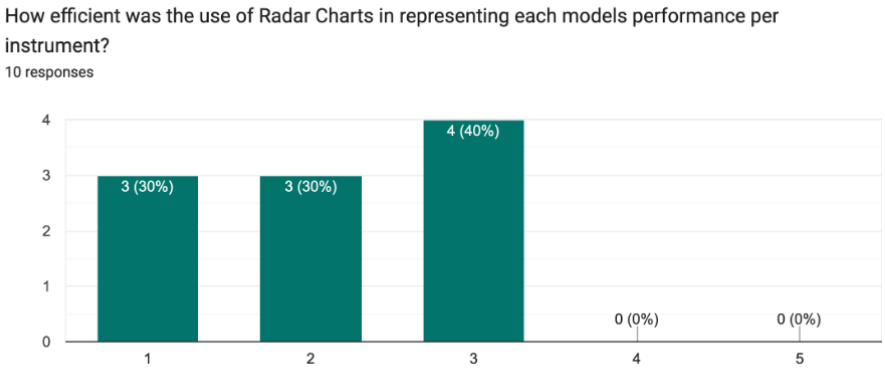

Figure D.30: How efficient was the use of Radar Charts in representing each models performance per instrument?

"The arrow buttoms were too small and kept pressing view raw data instead of arrows - also no label for which sound was spleeter or demucs"
"it would be nice to be able to synchronise the playback and then choose to isolate one of the options, it might make it easier to hear the differences without idle time inbetween"
"Hard to find where to navigate to the home screen"
"put labels on the different tracks hard to remember which is which"
"n/a"
"Label the waveforms to change between instruments"
"Arrows to switch graphs and audios are too small, loading bar for separation."
"Radar charts dont help when both models performed simillar, no zoom for small differences."
"Arrows too small"
"Loading Bar for separation progress"

Table D.2: If you noticed any difficulties, please note below. Enter n/a if not.