Boombox: Temporal Music Embeddings for MIR Tasks

Shrey Sahgal
University of Michigan
shreyps@umich.edu

Benjamin Steinig
University of Michigan
bsteinig@umich.edu

Ashwin Srinivasan
University Of Michigan
asriniv@umich.edu

Lochan Ramu Dayalan
University Of Michigan
lochanrd@umich.edu

Jonathan Bear
University of Michigan
bearj@umich.edu

Abstract

This paper presents a method for creating trajectory (temporal embedding) representations of songs and developing quantitative benchmarks for these representations on downstream MIR tasks. We explore related works in the field and explain how our system differs from previous implementations. We describe our music embeddings process and the web interface we developed to demonstrate our ability to perform music recommendation and genre classification using our novel representation. Finally, we present our results and discuss potential ethical concerns that come with developing a music embedding system.

1 Introduction

With the advent of streaming, accessibility to music has increased astronomically in the twenty-first century. While the field of musical analysis has existed for nearly as long as music itself, it has been found to be a very challenging problem to give to a computer. This is a surprising fact – digital music is simply bits and bytes flowing through circuitry, and it follows that the big data analytics that are ubiquitous in modern life can be applied to this data. But the fact that music so rich, personal, and subjective makes computer analysis a formidable task. The complexity and depth of music makes it nearly impossible to perform analysis directly on the raw audio. It turns out that this is a common problem. There are many modes of data including audio, images, and videos that are incredibly feature-rich, making it harder to create machine learning models that can draw interesting information from the data. In order to solve this, researchers have turned to creating new, latent representations of the data that aims to capture some of the complexities of the original data.

Our project, Boombox, builds on previous work in creating representations for audio by creating high-dimensional representations of songs that can be used in various music information retrieval tasks. We aim to create representations that capture the musical qualities of entire songs in such a way that representations of different songs can be directly compared. In order to do this, we draw inspiration from robotics and treat the features at different time steps as the 'trajectory' which the song takes over time. Our end goal for this project is to generate music recommendations based purely on the similarity between songs, as determined by algorithms based on our trajectory representations. While this approach will certainly be outperformed by complex recommendation models which profile user preferences, we believe that the creation of a musicality-based recommendation system can remove some of the privacy concerns of user-based recommendation, and can expose users to new songs or even entirely new genres.

In recent years, several models have been developed for creating music representations by extracting features from raw music waveforms. Some of these models include Music2Vec (Li et al., 2022a), MERT (Li et al., 2022b), and Data2Vec (Lian et al., 2023). But these models tend to perform poorly on large audio inputs and are instead more useful for shorter segments, 5-15 seconds. Music generation models such as Jukebox (Chowdhury et al., 2020) and MusicLM (Agostinelli et al., 2023) must create representations of music in order to generate novel music, but, again, these models focus on creating representations for shorter audio clips. Our method aims to create representations on the song level rather than on the audio clip level, allowing us to perform music information retrieval tasks such as genre classification and song recommendation on entire songs.

In Section 2, we will explore the related works discussed above and understand how our system differs from previous implementations. In Section

3, we explain the data we use in this project and how than could affect our results. In Section 4 we go into depth about our music embeddings process, our benchmarking methodology, our song recommendation algorithm, and, the web interface we developed to demonstrate our music recommendation capabilities. Finally, we conclude with our results and potential ethical concerns that come with developing a music embedding system.

2 Related Work

The primary goal of our work is to create a music representation that could be used for various downstream MIR tasks. In this section, we provide an overview of related work, including other implementations of creating general-use representations of music as well as performing music generation.

2.1 Representations of Music

Music2Vec is a self-supervised learning (SSL) method for extracting features from raw music waveforms. (Li et al., 2022a). The authors aimed to create a computationally efficient model comparable to Jukebox, which also uses SSL for music representation. They collected 130k hours of music audio files and used a subset containing 30-second long wave files to train the model. They trained the model from scratch with a single-GPU trainable size of 90M parameters, using standard pre-training protocols of data2vec and the fairseq framework. During pre-training, a student model aims to reconstruct the masked music audio by taking the contextualized representations provided by the teacher model as the input target. The model was trained for 400k steps, taking around six days with eight GPUs.

MERT is another SSL model that extracts features from raw music waveforms. (Li et al., 2022b). It was developed by the same authors as Music2Vec, with the aim of improving on the previous method. We use the MERT model in this project to create features for small segments of entire songs. See Section 4 for information.

Data2Vec is another representation that was created with the goal of creating a more generalizable way to represent complex input in a latent vector space. (Lian et al., 2023). The framework combines masked prediction with learning of latent target representations, which are generalized by using multiple network layers as targets. An off-the-shelf transformer network is trained in either student or teacher mode. In teacher mode, representations for the full input data are built to serve as targets for learning. In student mode, a masked version of the input sample is encoded and used to predict the full data representation.

2.2 Music Generation

Music generation is an adjacent problem to that of music representation. In order to generate music, music generation models must build representations of the music they are trying to generate.

Jukebox is a generative model that generates music with singing in the raw audio domain. (Chowdhury et al., 2020). It uses a hierarchical vector quantized variational autoencoder (VQ-VAE) architecture to compress audio into a discrete space. The loss function is designed to retain the maximum amount of musical information, and this is done at increasing levels of compression. An autoregressive sparse transformer is used with maximum likelihood estimation over this compressed space, and an autoregressive upsampler is trained to recreate the lost information at each level of compression. The model learns to encode input sequence x using a sequence of discrete tokens z, which are mapped to embedding vectors from the codebook C. The decoder decodes the embedding vectors back to the input space. The model was trained using a dataset of 1.2 million songs, with 600k in English, and artist and genre labels were provided to the VQ-VAE for conditioning. The model can be steered to generate a particular style of music by conditioning on lyrics.

MusicLM is a language modeling framework for generating symbolic music, such as MIDI files. (Agostinelli et al., 2023). The framework uses a transformer-based architecture and is trained on a large dataset of symbolic music to learn the patterns and structure of musical notes and chords. The model can generate new musical compositions by predicting the next note or chord based on the previous sequence of notes. The framework also includes several features to control the musical style and complexity of the generated compositions.

Riffusion was a ”Hobby Project” that used stable diffusion on spectrograms to generate short clips of music. (Forsgren and Martiros, 2022). The authors used the out-of-the-box v1.5 stable diffusion model fine-tuned to generate spectrogram images. Spectrograms are invertible, so once a spectrogram is generated with stable diffusion, it can be converted

into an audio clip using the Griffin-Lim algorithm. When an input is put in, they first generate an initial image, then they use image-to-image generation to generate variations on the same spectrogram. Then they smoothly interpolate between the original and loops of the variations to generate longer-form music. However, it was not clear if they had a separate dataset of spectrograms or audio files with matching descriptions.

3 Data

We primarily use three datasets in this project. Oneiswas used for training the encoding model, one is used to benchmark the representations for the MIR task of genre classification, and one is used as the collection of music from which we recommend songs to users.

The first dataset, which we call the Small Boombox Dataset, is a small collection of 365 songs from 5 different genres: 90s hip-hop, 90s rock, 2010s pop, country, and classical. We generated this dataset by downloading videos from YouTube. We found five playlists that contain between 60 and 90 songs from each of the genres and download each song in the playlist. We can then use the raw audio files as the data and the genre of the playlist as the label when performing genre classification. One potential problem with this method is that genre is not a fixed and definitive label – what might be considered country music by one person might be considered rock by another. And because we are scraping from playlists generated by anonymous YouTube users, we are locked into using some unknown definition of the genre. Moreover, songs can be fall into many genres, so it might be reductive to consider one song to be only 90s hip hop if it has some elements of rock or even country.

The second dataset we use is the GTZAN dataset, which has been widely used in MIR literature for the task of genre classification. The dataset contains 1,000 thirty-second song clips from ten genres: blues, classical, country, disco, hiphop, jazz, metal, pop, reggae, and rock. The potential issues and limitations of the GTZAN dataset are well-studied (Sturm, 2014) and it is known that there are problems with mislabeling and distortion within the dataset. But since it is commonly used in MIR literature, we still believe it provides an important benchmark of how well we can perform genre classification. Rather than having raw audio for entire songs, the GTZAN dataset contains the raw audio for only thirty-second song clips that do not have the same musical structure as full songs. Because we are trying to representations on the song level rather than on the music clip level, the GTZAN data might not perform as well on our implementation. This fact could make it harder to benchmark our implementation with other genre classifiers. But still, the ubiquity of the GTZAN dataset has motivated us to use it to benchmark our own implementation.

The final dataset we use, which we call the Large Boombox Dataset, is very similar to the Small Boombox Dataset, except with considerably more songs from a more diverse range of genres. This dataset contains 2,900 full songs from 14 different genres: 2000s alt rock, 90s rock, movie scores, 2000s indie, classical, reggaeton, 2000s r&b, 2010s hiphop, country, 2010s pop, jazz, 80s pop, 90s hiphop, and lofi. Like the Small Boombox Dataset, this dataset was downloaded using YouTube playlists containing music from each genre. It is subject to the same limitations as the small dataset but to a higher degree. Because there are so many more songs and even more overlapping genres, we occasionally see songs that appear in multiple genres, for example songs that appear in both 2000s alt rock and 2000s indie, as these are musically very similar genres. We get around this problem by cleaning the data before using it.

4 Methods and Algorithms

This section is split into four primary parts. The first explains how we create a trajectory representation for any given song. The second concerns how we went about quantitatively benchmarking our representations using the downstream MIR task of genre classification. The third section describes how we then use these representations to generate song recommendations. Finally, the fourth section detail how we created our web interface that allows users to generate song recommendations by inputting songs that they like.

4.1 Creating Trajectory Representations

Given a new song, creating the trajectory representation for the song (a process we call "trajectorization") involves three steps. In the first step, we use MERT, a BERT model fine-tuned for music from Li, et. al (Devlin et al., 2019) (Li et al., 2022b). We then trained an encoding model that reduced the dimensionality of each feature. Lastly, we split each

song up into S segments (or "songlets"), summing over songlets to get a time-normalized feature trajectory that we can then use in further downstream tasks.

Li et. al.'s MERT model takes as input a short (3-15 second) music clip and generates 13 feature vectors each of size 768×1, one feature vector for each hidden layer, meaning we have a 13×768 feature for each window. We create a slide a window of length 3 seconds over the song with approximately 1 second overlaps to split the song up into audio clips that we can pass into MERT. The stride of the window dynamically changes to ensure that the entire song is captured, but it is bounded between 1 second and 2.5 seconds. We then pass each of this audio clips through the MERT model, creating a $W \times 13 \times 768$ matrix where the ith column contains the feature for the ith window and where W is the number of 3 second windows created for the song.

To reduce the dimensionality of the features, we train a model that takes as input individual flattened 13×768 features, has an intermediary hidden layer of dimensions 1×768, and then predicts the genre of the input features. To encode a new song, we simply pass the song through the first layer of the model, which allows us to convert a 13×768 output from the MERT model into a single 1×768 dimensional feature vector for a particular time window. To train this model, we use the Small Boombox Dataset. Despite training on such a small sample (365 songs from 5 genres), we found that the encoding model generalized well to both the GTZAN and Large Boombox Dataset, which speaks to the robustness of the MERT features. We go into further detail about the generalizability of the encoding model in Section 5.

Once we pass each sliding window feature of a song through the encoding model, the result is a $W \times 768$ representation of the song, which we call the "feature matrix." Because songs are of different lengths and have different number of windows, feature matrices will have different dimensionalities, which makes it difficult to directly compare songs. The ameliorate this issue, we introduce the idea of "songlets." A songlet is a fraction of a song, where the number of songlets into which songs are divided is defined as S. For a song containing W windows, we create $S - 1$ songlets that contain $\lfloor W/S \rfloor$ and create the final songlet containing the remaining features, meaning it will have $W \mod S$ features. The number of songlets S is a heuristic, but for our data where songs are generally between three and five minutes, we found $S = 10$ to work well.

A key part of our strategy for creating these representations is to treat each 1×768 feature vector at each time step of a song to be the 'direction' in which the song moves over that time interval. Using this framing, we can consider the feature matrix for a song to represent a sort of trajectory for that song – the feature matrix describes the direction of the music over the entirety of the song. Thinking about the feature matrix this way, we collapse are able to collapse the features of a songlet by simply summing over the columns of the songlet. This operation becomes a vector addition, meaning that the sum of the songlet will capture how the song moved over the entire songlet, rather than just over a single time step. This allows us to reduce a song's $W \times 768$ feature matrix to an $S \times 768$ matrix, where S is generally much smaller than W. The results of this operation are "songlet trajectory" representations of standardized dimension, $S \times 768$, that are significantly lower-dimension than the original features. Because songlets capture a different number of features depending on the length of the song, we normalize the sums of songlets to unit length. This allows for more direct comparison between songlets, although it does result in some loss of information about the length that the songlets capture. The pipeline of creating these songlet trajectory representations of songs is detailed in the Appendix.

4.2 Benchmarking Trajectory Representation

Our goal for this project is create representations of songs that can be used in a myriad of downstream music information retrieval tasks. Our philosophy is that if we can show qualitatively that trajectory representations of songs are useful in downstream MIR tasks, then our representations do, in fact, contain rich information about the musicality of songs. Thus, we can use these representations to determine the similarity between songs, allowing us to perform song recommendation based purely on the musicality of songs. This section will describe how we use the GTZAN data and different parts of our pipeline to benchmark our representations. For further detail about the results of the benchmarking, see Section 5.

In order to do this, we benchmark our representations on the GTZAN dataset for the task of genre

classification. As described in the Section 3, the GTZAN dataset is commonly used in MIR literature for this task. We trained three different genre classification models using the GTZAN data: one on the feature level, one on the individual songlet level, and one on the songlet trajectory level. Training different models for different granularity allows us to see what parts of the trajectory representation generation pipeline result in the most loss of information.

The feature-level classifier takes as input individual 1×768 features from each song after the MERT model and the encoding model are applied. Each feature's genre label is the genre of the song from which it was taken. We used a relatively simple fully connected neural network that had five layers to go from an input vector of dimension 768 to a 10 dimensional output. We use softmax activation for our last layer and categorical cross-entropy loss to train this model. The results were good; this relatively simple model achieved a final accuracy of 91%, which is considerably larger than random chance (10%, since there are 10 genres in the dataset).

The songlet-level classifier takes as input individual songlets, each of size 1×768. Similarly to the feature-level classifier, the genre labels for each individual songlet is the genre of the song from which the songlet was taken. We again implemented a relatively simple fully-connected network that had five layers to go from a 768-dimensional input to 10-dimensional output with softmax activation for the last layer and categorical cross-entropy loss during training. Similar to the feature-level classifier, the simple songlet-level classification network had very good results. The final test accuracy of the model was 93%, which is much better than random chance.

The trajectory-level classifier takes as input an entire song trajectory of size $S \times 768$. For this classifier, we used a CNN architecture, as we believe this helps maintain the temporal nature of our representation. Using two convolutions and a single fully connected layer after flatting, softmax activation for the last layer, and categorical cross-entropy loss, the highest final accuracy for this model was 73%. While this is certainly worse performance than the classifiers on the feature- and individual songlet-levels, it is still much better than random chance. We will further discuss the reasons for the poor performance and analyze the confusion matrix for this model in the Results section of this paper.

4.3 Generating Song Recommendations

Using the GTZAN dataset, we have verified that our trajectory representations of songs contain rich information about the musical qualities of songs. And since these representations are built with the idea in mind that the representations can be thought of as trajectories for the songs, we then aim to generate song recommendations based on the similarity of songs in a machine learning-free and purely mathematical way.

In order to generate these recommendations, we leverage some linear algebra background. Given a collection of M songs that a user has indicated they like and a collection of N songs from which recommendations are selected, we first convert each song to an $S \times 768$ trajectory representation. We then create S different 'liked' subspaces, one for each songlet. We do this by creating S matrices V_s, where the ith column of V_s contains the ith song's s songlet feature, where $1 \leq s \leq S$ and $1 \leq i \leq M$. We can then say that each of these liked subspaces captures the musical qualities that the user likes at each particular songlet. We also create S projection matrices that project each feature into a particular liked subspace:

$$P_s = V_s(V_s^T V_s)V_s^T$$

We then consider the trajectory representation for each song of the N possible recommendations. For a candidate song C, we project each of the song's songlets C_s into the corresponding liked subspace V_s. We then take the L2 norm of the difference of these vectors, which we can then say captures the difference between the original songlet and the songlet projected into the liked subspace. We perform this operation for each songlet and sum the norms over the songlets, giving us a numerical value for the difference between the original song and the collection of liked songs. More formally, the song difference D between a candidate song C and the collection of liked songs is:

$$D = \sum_{s=1}^{S} \|P_s C_s - C_s\|_2$$

We can then calculate this song difference for each song in the collection of candidate songs and return the R songs with the lowest song difference. R is a heuristic that depends on the intended usage of the recommendation algorithm.

4.4 User Interface

The goal of our user interface system was to create an application which would best showcase the tasks that we completed in the previous sections, packaged in an interface which was accessible to users. The system that we designed allows users to select songs that they are interested in through a YouTube search API, generates visualized trajectories, and provides recommendations for the given songs. Given the significant compute cost of generating trajectories and recommendations we determined that any implementation would require a dedicated CUDA enabled back end.

In order for this system to work we need to track a process as it progresses through the server and link that information to the UI. We decided to use a hear beat connection, where the back end continuously updates its progress and estimated completion, and the front end sends requests based on that same estimated completion. The responses sent by the server can be seen in Figure 1, these responses are generated from the corresponding endpoints on the API for the trajectories and recommendation respectively. The server interprets the status by pinging the endpoint routinely based on the previous messages' expected finish prediction. This hear-beat chain is triggered by the server response to the client's initial POST request. The specific details of these systems will be discussed in the following sections.

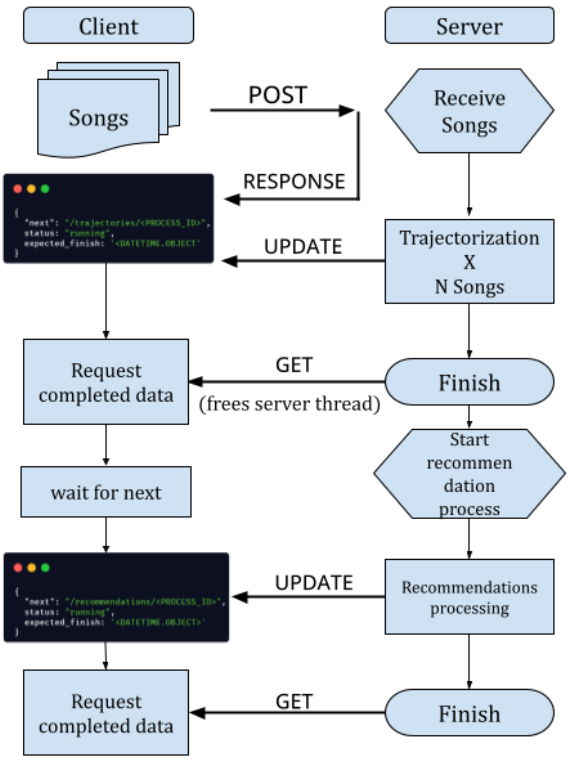


Figure 1: Client-Server Architecture

4.4.1 Front End

The primary goal of our front end interface was to create a seamless and interactive environment for users to interact wither the recommendation system. A facet that we thought would be important to highlight was the visualization of the trajectories, since we found them to be quite visually striking in our original testing. Given our objectives for the front end system, React was a clear solution for our implementation. We used the NextJS library which extends the React system to support routing and server side rendering out of the box.

Based on the general architecture described in the previous section, it was important to use dynamic routing to separate different requests. The added benefit of this system is that past recommendation generations are infinitely retrievable with their corresponding process id. We implemented unique process ids to separate processing on the server side, but found it was also effective for driving the system flow of the front end. When a generation process is submitted, the front end receives a unique URL to route to. This routing triggers the process to create the trajectories for visualization and thereafter the recommendation process. The front end mediates the status of the process through a persistent connection with the relevant back end endpoints. As trajectories and recommendations are returned the front end propagates the UI with the corresponding components.

Following a generation request, the front end first receives the visualized trajectories, this allows us to create an interactive loading environment as the recommendations continue to process. The trajectory viewer allows users to interact with 30 three-dimensional visualizations. The visualizations are passed from the back end as (N, F_1, F_2) where N represents the number of timesteps and F_1 and F_2 represent the reduced features. The component then interpolates a Centripetal Catmull-Rom Spline through the points. We determined that a Catmull-Rom spline was most appropriate spline algorithm for this use case since it retains the positions of the input points and produces a constrained, continuous shape aiding in the aesthetics of the trajectory. The component itself was

built using react-three, a react port of the popular JavaScript graphics library ThreeJS.

In addition to the trajectory viewer, we also visualized the relationships between genres during recommendations. We found that some genres were more likely to be recommended by others. We visualize these relationships in a chord diagram, provided by the AirBnb visx library.

4.4.2 Back End

The back end server was designed to provide the needed functions from our music representation system to create the trajectories and recommendations needed to make the system work. Since our system was primarily designed using python and pytorch functions it made sense to create our server using the same language for compatibility. This led us to choose Flask, a lightweight and easily scalable python WSGI Server. Flask allowed us to create a REST API with dynamic endpoints for requesting trajectories and recommendations. While there was some linear order to the processes that we needed to complete to serve recommendations there were several areas where we were able to achieve speedups and reduce overall compute cost.

One of the primary methods for speedup we used was to cache trajectories and embeddings as we processed them. Along with our dataset, as new songs are added we store trajectories and embeddings identified by their YouTube ids in an SQL database. The database stores visualization features and embedding features separately, identified by their $youtubeid$ and linked by the $processid$ they were created with as seen in Figure 2. Likewise, the recommendations table connects back to the process id which generated it, and the list of YouTube id's mapping to the recommendations. When a user sends a request the system first checks the database before performing any new compute. As our library grows we expect this to allow for significant speedup to our system.

Additionally, we use python's multi-threading library to parallelize the trajectorization task. In our initial testing, trajectorizing an uncached set of songs took an average of 2 minutes to complete. In order to be feasible as a web interface, we needed the system to complete in less that 30 seconds. By creating a new process with N threads, 1 per song in the request, we could perform YouTube download and trajectorization in parallel. Using the multi threaded approach our average compute time for an uncached set took 25 seconds on average. In the future, we believe that it may be more efficient to compute the trajectories in a batched operation as opposed to running individual CUDA threads.

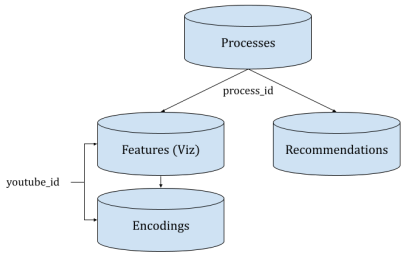


Figure 2: Server Database Schema

Once, the trajectorization of the songs is completed. The system automatically launches the recommendation sub-process. This process is relatively cheap and would not benefit significantly from parallelization.

Unfortunately due to cost constraints of hosting a CUDA enabled backend we are unable to provide a live demo, but Appendix B contains screenshots of this system in action.

5 Results

In this section, we present the results of the methods we discussed in Section 4. We split this section up into three parts: the results of the encoding model, the results of GTZAN benchmarking, and the results of song recommendation. Due to space constraints, we present the results of the more granular GTZAN classification models and the results of performing genre classification on the Large Boombox Section in Appendices A and B, respectively.

5.1 Encoding Model

For the encoding model, we evaluated the accuracy and loss based off of the MIR task of genre classification. The model performs quite well: the final accuracy of the genre classification is in the mid-90s, which is considerably higher than random chance, which is 20% as there are five genres in the Small Boombox Dataset.

There are a few potential problems with this model. One is that it is very shallow, although it

does seem like the model is learning quite a bit. Another problem is that it is trained on a very small dataset, and we were worried that this would result in poor generalization to the GTZAN dataset and the Large Boombox Dataset. But as the next subsection and Appendices B and C show, the resulting features are actually quite rich and generalize well to datasets that include genres on which the encoding model was not trained. Training plots for the encoding model are provided in the Appendix.

5.2 GTZAN Benchmarking

The final model for performing genre classification on songlet trajectories with the GTZAN dataset achieves around mid-70s final test accuracy, which is lower than the low-90% accuracy that the feature- and songlet-level classifiers achieved. Looking at the confusion matrix, we found that the model seemed to perform very well on genres that are distinct from others: classical and jazz performed particularly with very few misclassifications on the test set. Genres that are less distinct from the others seemed to result in more misclassifications. The model seemed to struggle with rock in particular. We argue that this is due to the similar instrumentation between rock songs and other genres of music. That said, it is surprising that rock was frequently misclassified as disco, as there are not many musical similarities between these genres. That said, we are very satisfied with these benchmarking results, and we believe that they strengthen our argument that our trajectory representations adequately capture the musical qualities of songs. Training plots and the confusion matrix for the GTZAN genre classifier are provided in the Appendix.

5.3 Song Recommendation

Interpreting our song recommendation results was quite challenging, as 'musical similarity' is a qualitative and subjective judgement. To determine the efficacy of our song recommendations, we ran the following experiment. For each genre in the Large Boombox Dataset, we selected 10 random songs as the liked songs. We then calculated the song similarity between every song in the dataset and this single-genre liked song list. We then averaged the song difference across each genre, giving us an average song difference within and between genres. The resulting confusion matrix for this experiment is shown in the Appendix.

6 Discussion

The bias in our data exists in that it is for the most part English songs. On a bigger scale, our project would have to incorporate different languages to accommodate the user in a more holistic fashion. If our project can only recommend English songs, then it is ignoring an entire portion of the population, which means an entire portion of the population wouldn't be able to use our application simply due to ignorance.

Another possible ethical issue revolves around user identity security. While currently, our project does not collect user data and store it, in the future it could. It could become possible that a user leaves a digital fingerprint behind based on their playlist and the subspace it encompasses. We would have to take extra precaution to make sure that not only is the basic user identity information safe, but any data we get with our MIR process.

Finally, we have to take into consideration copyright issues with the music we have in our own training dataset. The music industry is an industry of creative work, and so we have to make sure that any data we use is properly sourced and that we don't break any rules surrounding possession and usage rights.

As of now, these are the main ethical issues we have given consideration to. As the project grows in the future, other issues may arise, and so we will give proper consideration to these issues and make sure we are being responsible with our project's source and usage.

7 Conclusion

Our aim for this project was to create feature-rich representations of entire songs. While previous work has been able to use self-supervised learning to create represenation for song clips, we treat create 'trajectory' representations for entire songs. This process allows us to perform music information retrieval tasks such as genre classification and song recommendation on entire songs. Based on quantitative results using the GTZAN dataset and qualitative results from song recommendation, we believe we have succeeded in this goal. We also created a web interface that allows users to visualize these song trajectories and obtain recommneded songs.

References

Andrea Agostinelli, Timo I. Denk, Zalán Borsos, Jesse Engel, Mauro Verzetti, Antoine Caillon, Qingqing Huang, Aren Jansen, Adam Roberts, Marco Tagliasacchi, Matt Sharifi, Neil Zeghidour, and Christian Frank. 2023. Musiclm: Generating music from text.

Anurag Chowdhury, Austin Cozzo, and Arun Ross. 2020. Jukebox: A multilingual singer recognition dataset.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Seth* Forsgren and Hayk* Martiros. 2022. Riffusion - Stable diffusion for real-time music generation.

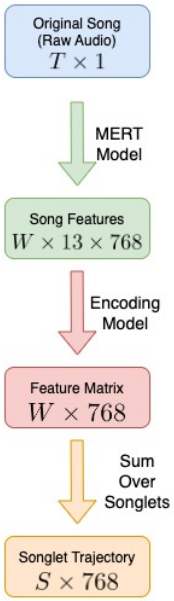
Yizhi Li, Ruibin Yuan, Ge Zhang, Yinghao Ma, Chenghua Lin, Xingran Chen, Anton Ragni, Hanzhi Yin, Zhijie Hu, Haoyu He, Emmanouil Benetos, Norbert Gyenge, Ruibo Liu, and Jie Fu. 2022a. Map-music2vec: A simple and effective baseline for self-supervised music audio representation learning.

Yizhi Li, Ruibin Yuan, Ge Zhang, Yinghao Ma, Chenghua Lin, Xingran Chen, Anton Ragni, Hanzhi Yin, Zhijie Hu, Haoyu He, et al. 2022b. Large-scale pretrained model for self-supervised music audio representation learning.

Jiachen Lian, Alexei Baevski, Wei-Ning Hsu, and Michael Auli. 2023. Av-data2vec: Self-supervised learning of audio-visual speech representations with contextualized target representations.

Bob L. Sturm. 2014. The state of the art ten years after a state of the art: Future research in music information retrieval. *Journal of New Music Research*, 43(2):147–172.

A Representation Creation Pipeline



B Boombox Interface

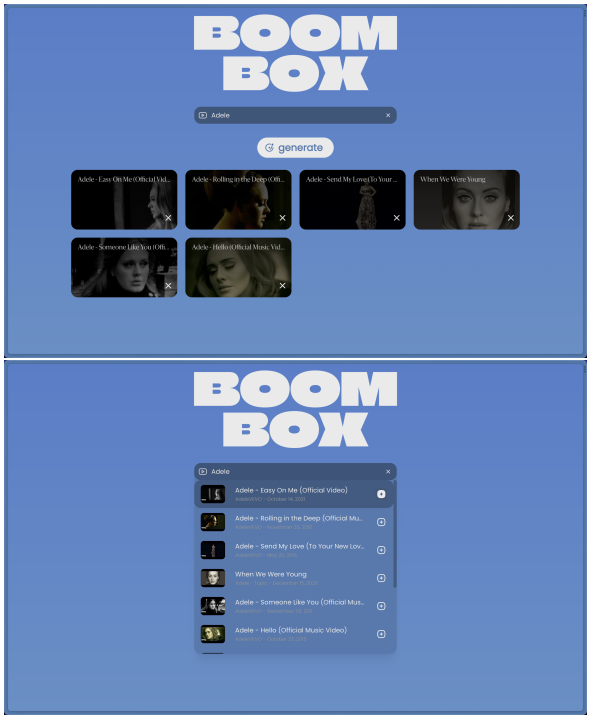


Figure 3: YouTube Search and Music Selection

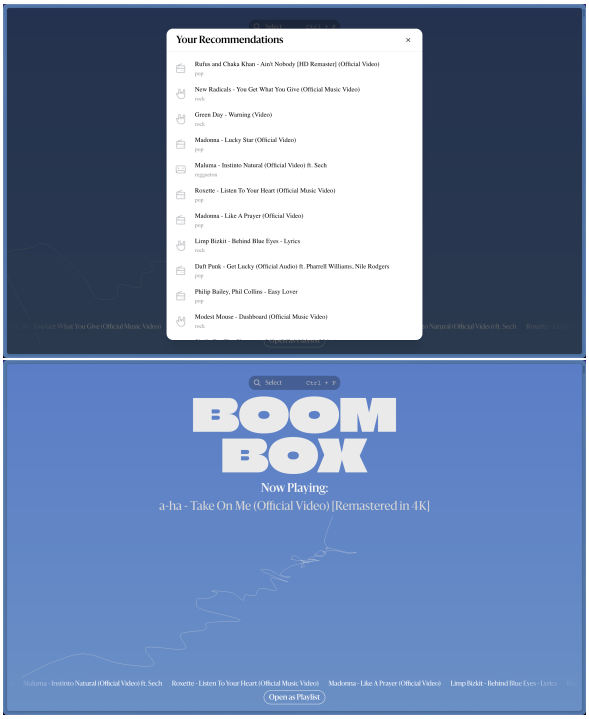


Figure 4: Recommendations and Trajectory Viewer

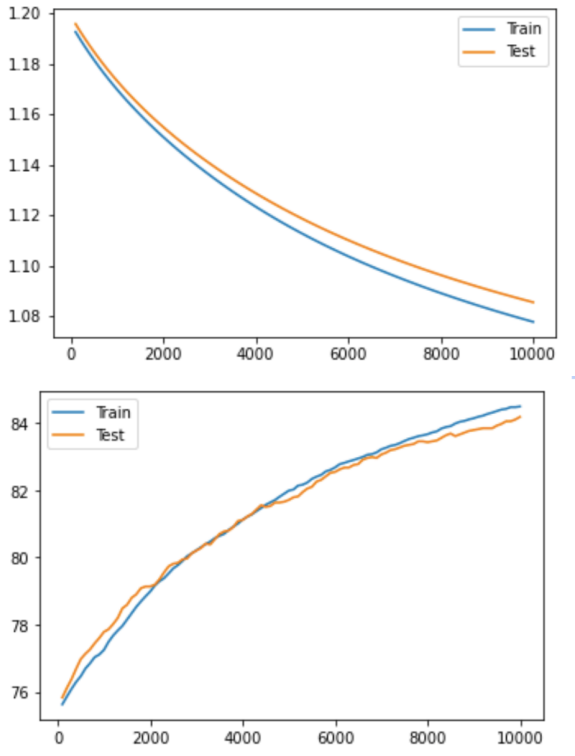


Figure 5: Above: Training and test loss for the encoding model over 200 epochs of training. Below: Training and test accuracy for the encoding model of 200 epochs of training.

Figure 6

C Encoding Model Training Curves

C.1 Songlet Trajectory-Level Clasiifier

Figure 6 shows the training curves for the encoding model.

D GTZAN Benchmarking

D.1 Feature-Level Classifier

Figure 7 shows the training curves for the feature level classifier.

D.2 Individual Songlet-Level Classifier

Figure 8 shows the training curves for the songlet level classifier.

D.3 Songlet Trajectory-Level Clasiifier

Figure 9 shows the training curves for the songlet trajectory level classifier. Figure 10 shows the resulting confusion matrix.

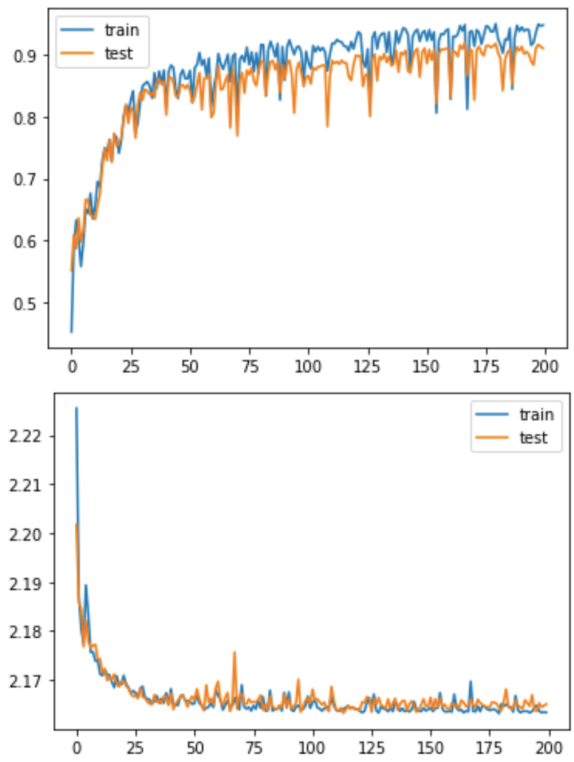


Figure 7

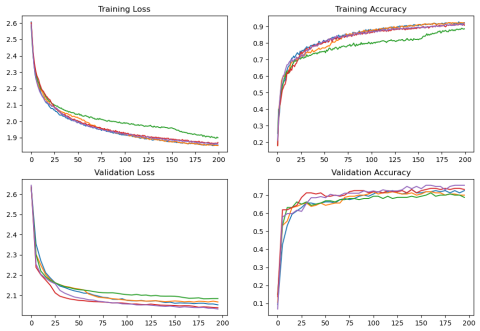


Figure 9

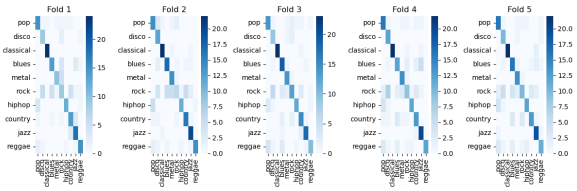


Figure 10

E Song Recommendation

Figure 11 shows the resulting confusion matrix of the experiment described in section 4.

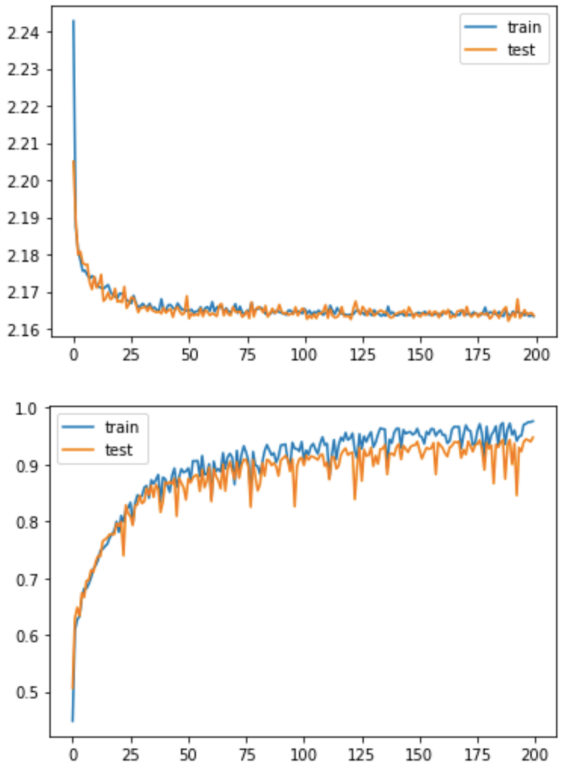


Figure 8

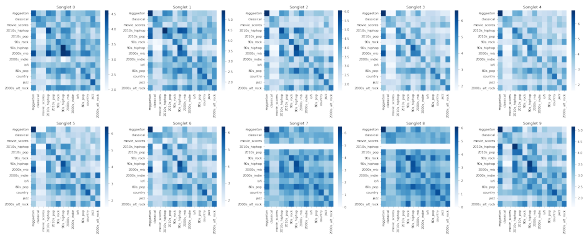


Figure 11