Playlister

Ofir Nissan 313292294 Yael Shemesh 208788034

May 2024

1 Opening

A year ago we went to a dance party with no DJ. Instead, the hosts played a playlist using popular application. It was nice, but the transitions between songs sometimes didn't match and ruined the flow of the music. A few days later we started to think about creating an automatic Playlist Player.

2 Background

2.1 The DJ

A disc jockey, commonly known as DJ plays recorded music for an audience. The DJ combine music records together such that the whole mix sounds like a continuous music stream (or one long song). The recorded music can be songs, tracks, remix, beats, special effects etc. This task can be very complex and require skills and creativity. Two important tasks are: *track/song listing*, meaning set the order of the songs/tracks ; *transition points selection*, meaning select the point to stop current song and start the next one (also, at the next song, select the specific point to start it). There are many criteria that the DJ uses to set the order of the songs and select the transition points. For example, if the DJ wants to create a "ramp up" mix, than the songs need to be ordered according to their tempo (song with lowest tempo first, highest tempo last).

2.2 Automatic DJ Systems

There are attempts of creating automatic DJ systems, so the concept is not new. There are many ways to approach this problem. For example: [1] uses DSP methods for tracks beat matching and setting the order of the tracks according to required tempo trajectory ; [2] uses more complex architecture which includes beat tracking, time stretching, cross-fading etc. and evaluate its success using ML methods (such as SVM), but it is limit to Drum and Bass, a specific genre of electronic dance music.

3 The Problem

Given a list of song (playlist), we consider two problems that we want to address:

1. Choose the right transition point between two songs that comes one after the other in the playlist order.
2. Find the best playlist order so that transition between the songs will be as seamless as possible.

In this paper, we will discuss and exhibit a novel approach that exploits music generative model for *track listing* and *transition points* estimation. As far as we know, this approach is first of its kind in this field. With this method, we create an easy to use system with simple architecture that can be applied for creating songs mix.

4 Introduction

Music generative model creates new pieces of music autonomously. As part of the model's flow, it calculates the conditional probabilities (given some piece of audio). The main idea of our work is to extract these probabilities and use them for songs listing and transition points selection. We use the DTW algorithm as baseline solution for the same problems.

4.1 Initialization

Given a set of songs, for each song we cut its suffix and prefix with the same length (we set the default to be 45 seconds). The length of suffix and prefix has a clear impact on the solution to our problem. For example, the longer the suffix and prefix, there will be more options for transition points.

4.2 Finding Songs Order - General Method

We calculate the cost/distance between each pair of suffix and prefix. We will elaborate the cost calculation for both methods at the appropriate sections. This settings induces a graph in which there are edges from suffixes to prefixes and the weights are the costs. Now, the goal is to find the path with the minimal cost that goes over all nodes (songs) exactly once. This is an NP-hard problem that is similar to traveling salesman problem. We use two approaches:

1. Greedy solution. Start with random choice, find its minimal cost edge, pick the next node accordingly, finds its minimal cost edge and so on.
2. TSP 2-approximate algorithm as explained in [4].

In our evaluated runs, we used the TSP 2-approximate algorithm for both MusicGen based and DTW based solutions.

4.3 Avoiding Transitions in non-music Intervals

Both DTW and MusicGen methods for finding the optimal transition point has a bias to match silent suffixes to silent prefixes. The reason for that is obvious in both methods. To avoid transition in non-music intervals, we ignored low energy intervals. To do that we used following method: given potential transition point, we calculated the energy of the last seconds from the current song and the first seconds from the next song and make sure that their mean is bigger then predefined threshold. Normally (and in our evaluated runs), this threshold defined to be zero. We will not use the transition points with low energy.

4.4 Special Effects Note

There is no use of special effects at the resulted playlist. We focused only on selecting the transition points and track listing. So, we did not expect the resulted transitions to be absolutely seamless, but we hoped to get an enjoyable playlist that do sound like one long song/track. Having that said, we use only a simple fader with fade duration of 2 seconds to fade out the current song and fade in the next.

5 Baseline - DTW Based Solution

We use Dynamic Time Wrapping, or DTW, to align two audio signals and calculate their cost. We use the cost and alignment to solve our two main problems:

5.1 Finding Songs Order Using DTW

Given set of songs, we calculate the chroma-stft[1] of the suffix and prefix of each song. Then, we calculate the DTW cost between each pair of suffix and prefix (over their chroma-stft). Then we use the cost as the weights of the induced graph and calculate the best approximated path (see 4.2). Then we extract the songs order from the previous algorithm.

5.2 Finding the transition points using DTW

For each pair of consecutive songs, we calculate the DTW between each pair of cur song's suffix and next song's prefix (over their chroma-stft). Then, we use the alignment to find the transition point. Each element at the alignment object contains pair of indices that can be interpret as transition point (point in time to stop current song and point in time to start next song). We implemented several methods for selecting the specific point from the alignment:

1. Random selection. Choosing index randomly from the alignment.

[1] short-term Fourier transformation to compute Chroma features

2. Computing the index that gives the max propagation over some window. The idea is to find the point in which the alignment path propagate the most distance over the DTW cost matrix (diagonal step at the matrix). First, we set a window with some arbitrary length. Then, we calculate a propagation array from the alignment. Lastly, we take the point in which the next window has the maximum propagation.

3. Computing the index that cost less over some window. The idea is to find the point in which the alignment path has the lowest cost. To do that, we calculate the path cost and then use consecutive subtraction to find the window of the smallest cost. Then we take the point in which the next window has the smallest cost.

In our baseline runs for the evaluation stage, **we use the third option**. For every potential transition point, we make sure that the last seconds of the current song are not silent (see 4.3).

6 MusicGen Based Solution

6.1 MusigGen Model

MusicGen [6] is a Language Model that operates over compressed discrete music representation, i.e., tokens. The token is obtained from the EnCodec model, using Residual Vector Quantization (RVQ), and an adversarial reconstruction loss. Each quantizer encodes the quantization error left by the previous quantizer, thus quantized values for different codebooks are in general not independent, and the first codebook is the most important one. MusicGen uses 4 codebooks and each token is corresponding to 0.02 seconds from the audio.

6.2 Finding the Best Transition Point Using MusicGen

In order achieve our first goal, we wanted to find the best time to stop the current song and the best time to start the next song so that the transition will sound almost like the continuation of the same song. We divided the end of the current song into a list of suffix pieces (the length of each piece and the hop size between two different suffix pieces are hyper parameters):

$$S = \{s \mid s = (s_1...s_n) \text{ is a token sequence in the suffix of the current song}^2\}$$

and we divided the start of the next song into a list of prefix pieces (the length of each prefix piece and the hop size between two different pieces are hyper parameters):

$$P = \{p \mid p = (p_1...p_m) \text{ is token sequence in the prefix of the next song}^3\}$$

That is, we wanted to find the best pair of audio sequence from the current song and audio sequence from the next song. Formally:

$$\text{argmax}_{s \in S, p \in P} Pr(p|s)$$

We believe that this will cause the transition to be the best fit among all options (the continuation of the current song with sequence from the next song has the highest probability).

6.3 Calculate Every Continuation Option Using MusicGen

While trying to address the problems we defined above, we needed to calculate for given suffix piece s from the current song and prefix piece p from the next song, the following probability: $Pr(p|s)$, and we defined this probability as "what is the probability that the sequence p from the next song will come after the suffix piece s from the current song".

We use MusicGen model in the following way: We feed the model with a prompt of EnCodec codebooks corresponding to the original audio. The output from the transformer is transformed into logits prediction for each possible token at each step. We considered two methods for calculating $Pr(p|s)$:

1. Method 1: Feed the model with a prompt of EnCodec codebooks corresponding to the sequence of suffix piece concatenated with the prefix piece tokens $sp = (s_1...s_n p_1...p_{m-1})$. The shape of the output logits is $(\#codebooks, n + m - 1)$. Taking the last m logits and summing the log softmax over the actual tokens from the prefix piece will give us $log(Pr(p|s))$.

[2]The token sequence in the current song corresponding to a partial audio from the end of the current song.
[3]The token sequence in the next song corresponding to a partial audio from the start of the next song.

2. Method 2: Iterate over the number of tokens in the prefix piece sequence, and for each token i in the sequence run forward pass over the tokens $sp_i = (s_1...s_n p_1...p_{i-1})$ and use the logits and softmax to calculate the probability for each optional next tokens, and take the specific probability for token p_i. Summing the log probability for every p_i will give us $log(Pr(p|s))$.

After implementing both methods, we saw that the output $log(Pr(p|s))$ is approximately the same in both methods for given s, p. We decided to continue with the first method, since it requires only one forward pass for each pair of suffix piece and prefix piece, instead of m (the number of token in the prefix) iterations of forward pass.

6.4 Finding the Best Hyper Parameters (our training)

In order to understand if our method works well, we defined our loss in the following way:

$$\text{Loss} = \text{average}(log(Pr(p|s))) - \text{average}(log(Pr(s_{i+1}|s_i)))$$

where s_i, s_{i+1} are consecutive token sequences from the same song, and p, s are token sequences from the different songs. The length of s_{i+1}, p is equal, also the length of s_i, s is equal. Note that we want average$(log(Pr(p|s)))$ to be small, and average$(log(Pr(s_{i+1}|s_i)))$ to be high, i.e., we want the loss to be smallest as possible, so the word "loss" is appropriate in this context. First of all, our first sanity check is that **the loss is negative**, since we are expecting that the probability of consecutive token sequences from the same song, will be higher than the probability for token sequences from the different songs.

Second, we used this loss in order to find the best hyper parameters.

The first thing that we wanted to examine is the effect of different codebook on the loss. The results are:

1. First codebook loss = -255.88
2. Second codebook loss = -73.95
3. Third codebook loss = -8.41
4. Fourth codebook loss = -9.25

As we thought, the first codebook has the lowest loss. We decided to use only this token in our probability calculation.

Next, we continue with finding the best hyper parameter. We considered the following hyper parameters:

1. Window size suffix - the number of tokens in a suffix piece (corresponds to the audio length from the current song we conditioned on).
2. Window size prefix - the number of tokens in a prefix piece (corresponds to the next song audio length that we calculate its probability)
3. Hop size - The hop size both in the current song suffix and in the next song prefix (distance between each piece).

6.4.1 Hop size

We expect that small hop size will improve the results, because reducing the hop size will make us compare more options of suffix pieces and prefix pieces, and among them taking the most likely transition. We chose hop size = 0.5 seconds.

6.4.2 Window Size Prefix

The size of each prefix piece is the most tricky one. On one hand, we want it to be large in order to create transition that sounds smooth, i.e that the current song fits well to long duration from the next song. On the other hand, if we will increase it too much, our output will be probably affected by the next song itself and not by the transition between the songs. We started with plotting the effect of different window sizes of a prefix piece on the loss:

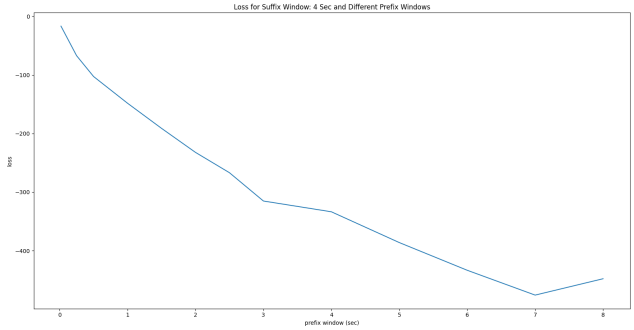

Figure 1: Loss for different window size of a prefix piece. x's axis is presented in seconds (#of token in a prefix piece$*$ 0.02)

The minimal loss is obtained in window size = 7, but the slope is steepest at lower window sizes.
Next, we tried looking separately on average($log(Pr(p|s))$) and average($log(Pr(s_{i+1}|s_i))$) (the probability for different songs and the probability inside the same song) normalized by the number of tokens in the prefix (in order to get the average probability for one token in every option).

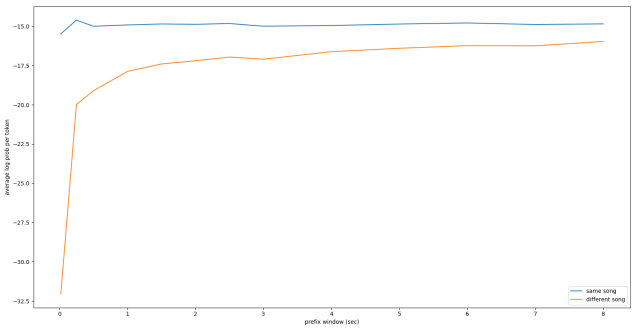

Figure 2: Average log probability normalized by the number of tokens in the prefix. blue line - two consecutive token sequences inside the same song, orange line - two token sequences from different songs.

It can be seen from the blue line that the probability for token in the same song does not affected by the length of the tokens sequence (prefix piece). On oppose to that, the probability for token in different songs does affected by the length of the tokens sequence, and the first tokens in the prefix piece are the most important ones. We decided to try prefix windows with sizes 1, 2, and 3 seconds and listen to the results.

6.4.3 Window Size Suffix

We believe that except for running time considerations,it is better taking large window size. Furthermore, we believed that the size of a suffix piece will not have a dramatic effect on the loss. We calculated the loss over the songs:

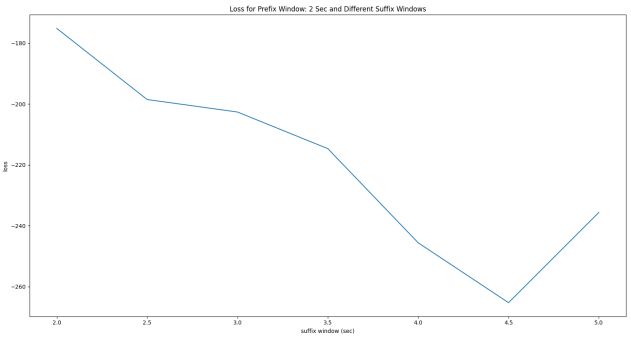

Figure 3: Loss for different window size of a suffix piece. x's axis is presented in seconds (#of token in a prefix piece$*$ 0.02).

We can see that the loss range is not so big (-180 to -270) compared to the range with different prefix sizes (0 to -500). We decided to try suffix windows with sizes 4 and 5 seconds and listen to the results.

6.5 Finding Songs Order

Here, we defined the distance from song i to song j in the following way:
denote $S_i = \{s \mid s = (s_1...s_n)$ is a token sequence in the suffix of the the ith song,
$P_j = \{p \mid p = (p_1...p_m)$ is a token sequence in the suffix of the the jth song,

$$\text{distance}(i,j) = -\log(\text{argmax}_{s \in S_i, p \in P_j} Pr(p|s))$$

Then, we used the general approximation solution described in Section 4.2

7 Spleeter

Spleeter library. Spleeter is a source separation library with pretrained models that uses Tensorflow. It allows performing various types of separation. We used Spleeter to separate singing voice (vocal) and melody of music instruments (accompaniment).

7.1 Feeding MusicGen With Melody

We used Spleeter to feed MusicGen with the accompaniment only. We though that with this approach, the order of the songs and the transition point selection, would be more accurate and would take in account the rhythm and beats better. In practice, there was no significant improvement (based on our opinion).

7.2 Attempts to modify transitions to be seamless

We assume that transitions over melodies (without singing voice) would sound more natural and seamless, then transitions over singing voice. There were two attempts we made. Both didn't have major improvement:

1. First we tried to improve our baseline. We used Spleeter to find the intervals in which there are no vocals (over the suffix and prefix of two consecutive songs). Then we activated DTW algorithm for each pair of suffix piece and prefix piece with the same logic as in 5.2. The result was that the transition point didn't occur at the vocals intervals. The main problem with this approach is that it adds very strong constraints over the transition point selection. So, for many examples, the selected transition point was at the very end of the song.

2. We used Spleeter to fade-out current song and fade-in the next song only over the singing voice, while keeping the melody the same. Meaning, concatenate the melody as is and activate a fader only over vocals.

8 Evaluation

8.1 Survey

For the purpose of the evaluation, we conducted a survey among a sample group, including 4 musicians. As part of the survey, we let the group hear one of two playlists, each one of the playlists has been generate using 2 solutions: our MusicGen based solution ; DTW based solution (our baseline). Meaning, overall the survey contained 4 audio files and each participant had to hear 2 audio files corresponding to one of the playlists. The group was asked to rate each transition point and the order of each playlist. Links to the survey:
playlist1-survey ; playlist2-survey
Here are the results:

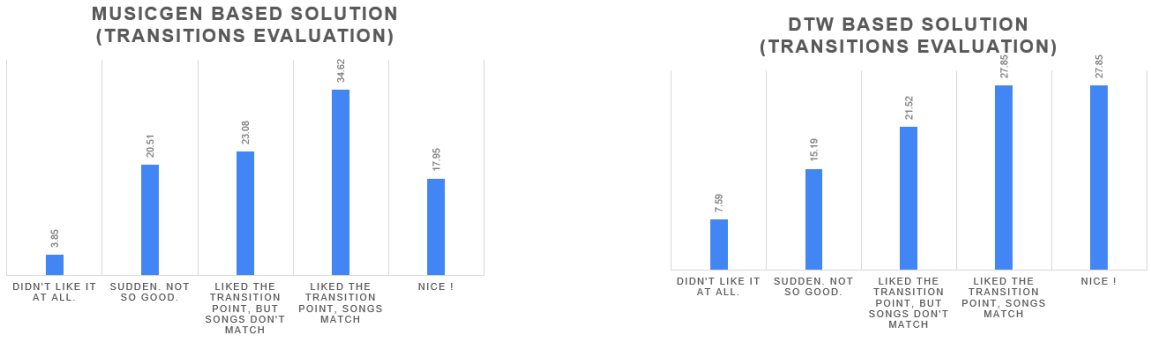

Figure 4: Transitions evaluations of MusicGen based solution in percentage

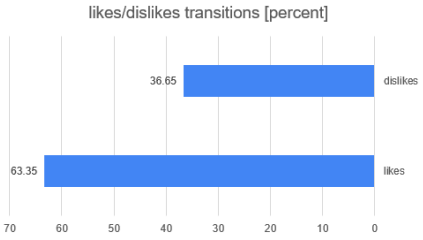

Figure 5: Likes and dislikes over all transitions [percent]

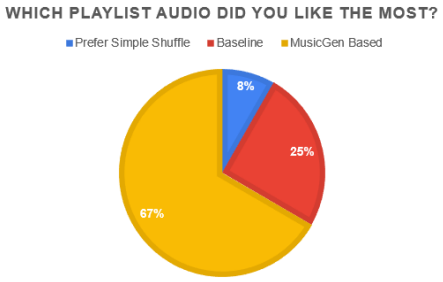

Figure 6: Favorite audio [percent]

Overall, the participants liked most of the transition points in both solutions. Despite the fact that it seems like there is a bit of advantage to the baseline solution in terms of transition point evaluations, the general preference is to the MusicGen based solution. One of the participant said: "Sometimes it felt like the fadeout should have been started exactly at the transition point. Overall, I had a great time. amazing work". Another participant said "Some of the transitions point were really good but sometimes the first song suddenly cuts. I believe special effects would help".

8.2 Tempo comparisons

We present two metrics for transition evaluation using tempo and beat track:

1. Prefix and suffix Tempo Ratio. Checks the tempo ratio of 2 audios (suffix and prefix of consecutive songs) over a window of 10 seconds.

2. For each pair of consecutive suffix and prefix, we calculate the average tempo. From the average tempo we conclude the expected distance between the last beat of suffix and first beat of prefix. Then we check the ratio of the actual distance between these 2 beats to the expected ratio.

We use these metrics to check the size of our suffix window and the size of our prefix window. Here are the results:

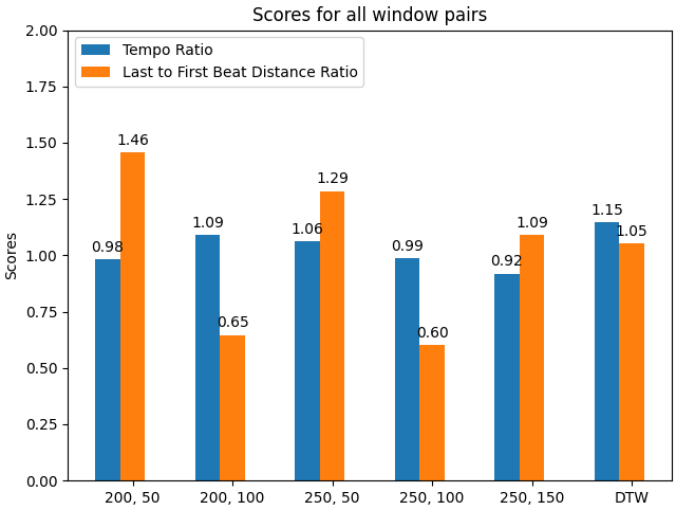

Figure 7: x-axis: each point (besides DTW at the rhs) represents a setting of suffix window size, prefix window size (in this order). The units of the window size is "token size", meaning [sec*0.02]. For example the pair 200,50 represent suffix window size of 4 seconds and prefix window size of 1 second.

According to the metrics we define, the baseline solution is doing a good job in keeping the tempo right. Also, according to the metrics, we can conclude that among the examples in figure 7, the best pair of suffix and prefix window sizes is 250, 150 (suffix window size of 5 seconds and prefix window size of 3 seconds).

9 Discussion and Future Work

In this paper we presented a solution for song listing and transition points selection, based on a music generative model (MusicGen). We compared our implementation to DTW based solution and the results were encouraging. However, there is significant work to do in order to get a seamless transition. Here we will discuss some ideas for future work:

9.1 Jukebox Model

When we start working on our project we initially thought to try using Jukebox model [3]. As we progressed with the project implementation, we found out that this model requires large amount of memory and compute time. Since we had a lot prefixes and suffixes pieces and we calculated the transition probability between every song to every song in the playlist, we have decided to continue with MusicGen model [6]. MusicGen is much smaller model that requires less compute time. One disadvantages of MusicGen model is that it has been trained mainly on melody. We thought that using Spleeter will overcome this disadvantages. we tried even using fader only on the vocal part, but there was not significant improvement (based on our opinion). We think that our method may work better using Jukebox model.

9.2 Insert Generated Transition Between two different songs

We believe that choosing the best transition point between songs is not always enough. As discussed in 4.4 the special effects are necessary to create a seamless transition. Another idea that we had is to insert a generated transition part between the songs. We had two idea for this goal:

1. **Use Music Generative Model with Masked Tokens [5]** With prompt of the current song, followed by masked tokens (depending on the order of the generated transition part length that we would like to create), and last the next song.

2. **Train right-to-left Generative Model** Using the same architecture and dataset used for the left-to-right music generative models, but doing reverse operation on the songs in the dataset. Then, combine both models in order to create generated music that fits the current song and the next song.

We are planning to continue working on the Future work we have right here. We had a great time working on this project and truly enjoyed the course.

References

[1] Dave Cliff. "Hang the DJ: Automatic Sequencing and Seamless Mixing of Dance-Music Tracks". In: *Systems and Systems Laboratory* (2000).

[2] Len Vande Veire. *From raw audio to a seamless mix: an Artificial Intelligence approach to creating an automated DJ system*. 2017.

[3] et al. Prafulla Dhariwal Heewoo Jun. "Jukebox: A Generative Model for Music". In: (2020).

[4] "https://www.geeksforgeeks.org/approximate-solution-for-travelling-salesman-problem-using-mst/". In: (2022).

[5] et al. Hugo Flores Garcia Prem Seetharaman. "VampNet: Music Generation via Masked Acoustic Token Modeling". In: (2023).

[6] et al. Jade Copet Felix Kreuk. "Simple and Controllable Music Generation". In: (2023).