Evolutionary Music

By

Shouyue Hu

Brendan Nugara

Chen sun

Project Type: [Research]

Course Code: [CISC 499]

Supervisor(s): Ting Hu

Date: April 15th

Table of Content

Chapter 1
Introduction

1.1 Motivation

In 2020, an event called Yearly Music Profile in NetEase Music App went viral by generating users' Annual Music Report including the type of music they are into, most listened to music in the year, the callback of a day where the user was looping a song, and more.(Zhou,2018) The Annual Music Report was shared all over social media in the Mandarin context during 2020, mostly by teens who want to show their taste in music and their personality reflected by the music they listen to.

This project aims to extend one more feature of the Annual Music Report, generating a user's own music track that represents the user based on their favorite music. To fulfill this goal, features in tracks need to be extracted to become parents in genetic programs, and the output will be audio that mixes the user's preference or genre. Such music generator models can also be used by artists and creators, to experiment with music creations. Current art in the field can already provide generative models for music(Boulanger-Lewandowski et al., 2012; Hadjeres et al., 2017; Roberts et al., 2018).

There are two traditional ways of music generation, rule-based scoring and similarity scoring. Rule-based scoring requires the application of rules containing music theory and musicality-based rules to generate music, which has the advantage of generating music with a greater degree of rule-based tendencies, but the disadvantage is that music as abstract content is difficult to quantify in terms of rules. Similarity scoring is another branch of music classification, where the representative application is OpenAI's Jukebox (Dhariwal et al., 2020). This kind of music generation uses a neural network to generate music with commonality using big data. It can be good for generating the same type of music but depends on the classification of good data. If the data in the training set contains too many styles of music, it will affect the quality of the generated product.

In this project, considering that a user's favorite songs are usually from different genres, it is not appropriate to give similarity to the neural network music generation. Evolutionary algorithms provide a more suitable solution to this problem. One of the advantages of using evolutionary algorithms in music generation is that even though it is difficult to use a predefined set of rules to rate music when the input is different kinds of music, by using evolutionary algorithms this rating can be done by user selection so that the rules and directions of the generated rating can be standardized according to the user's preferences. Thus, this project achieves this approach using Evolutionary Algorithm.

An Evolutionary Algorithm is a subset of evolutionary computation(Eiben, 2015), inspired by the evolutionary mechanisms of living organisms and simulating the evolutionary processes such as reproduction, mutation, genetic recombination, and natural selection to perform evolutionary computation on the candidate solutions of optimization problems.

1.2 Related works

Jukebox (Dhariwal et al., 2020)is a neural network-based machine learning modeling framework published by OpenAI that generates new music samples when the original music is in a certain genre and style. Jukebox uses CNN coding to encode raw audio into samples, and these sample sequences are passed through a trained transformer VQ-VAE(Gârbacea, Cristina, et al.2019) conditioned on lyrics to generate novel patterns, which are then upscaled by the transformer and decoded using CNN to output new music samples. The model generates music that is excellent in terms of musical quality, coherence, length of audio samples, and adaptability to artists, genres, and lyrics, but there is still a gap between it and human-created music.

One of the challenges Jukebox faces is that learning music generation models becomes increasingly computationally demanding as the music duration increases. In order to capture a variety of musical structures from timbre to overall coherence, Jukebox uses an autoencoder to compress the raw audio into a lower-dimensional space by discarding some perceptually irrelevant bits of information. The model is then trained to generate audio in this compressed space and upsampled back to the original audio space. Jukebox's future direction is to optimize its performance for diverse input and its accuracy in processing ultra-long distance structures.

NEUROGEN, done by Gibson and Byrne 1991 processes music separately with rhythm, melody, interval, and melody. A modulized pipeline decomposes the complex music into lower-dimensional data, so that each assessor generates a part of the music, and finally a harmonizer composites all those elements to form a piece of music. Specifically, a neural network is applied to each assessor so it brings a shortage that the length of music is fixed because the number of input layers is a constant. Longer music generation requires a restructure of the neural network or multiple genetic programming should be added.

Chapter 2
Algorithm

2.1 Algorithm workflow

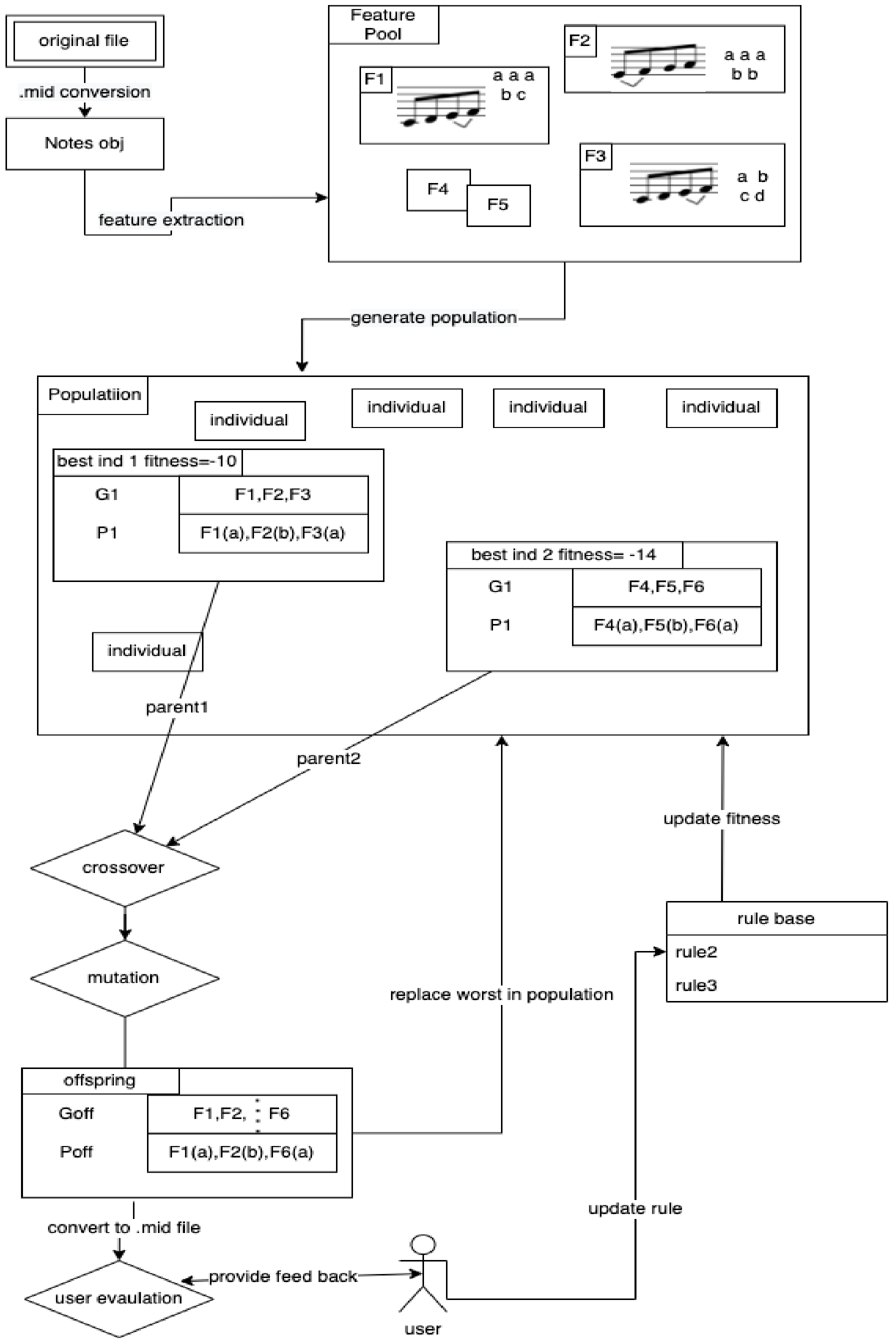

Figure 1

Figure 1.The evolutionary algorithm workflow. Firstly, the user input music file is converted into an operable object, then the music objects are analyzed to form a feature pool. Individuals connected by random features will constitute the population. Once the population is obtained, the evolutionary algorithm starts the cycle by selecting the parents through a tournament, and the parents are recombined and mutated to obtain the offspring. Offsprings are then added to the population. The evaluation method then assigns a score to each individual and eliminates the lower ones according to their score. After each of the n rounds, the software provides the user with multiple choices via the front-end, allowing the user to select the one he feels is the best, and this user's choice is then converted into a rating scale and added to the rating database. After the cycles, the rating database will contain several preferences of the user, which will influence the higher fitness of his preferred genre, thus achieving the purpose of filtering.

2.2 Evolutionary Algorithm

File conversion: MIDI(Musical Instrument Digital Interface), is a technical standard for describing music files, describing a communication protocol, digital interface, and electrical connector for connecting various electronic instruments and computers to play, edit and record music. The music input used in this article uses this vehicle. Due to the uniqueness of a MIDI file, converting it to a pentatonic score is complicated, so this paper provides a unique object for converting it for later in the evolutionary algorithm. Each MIDI file will contain multiple instrument tracks, each with multiple notes distributed in time, each with its duration and pitch. Our conversion algorithm reduces each track to an object, flattens out the sequence of notes it uses, and enters it into a list object to be stored for further use.

Feature Extraction: The evolutionary algorithm is characterized by the representation of each individual as a gene set. Each individual is defined by their own genotypic and phenotypic genes. Here, the expression is based on the combination of notes that appear several times in the original music as genes, which then form a gene pool for the formation of individuals. A characteristic, or in this case a gene, is expressed as a specific combination of notes, such as 'mi fa so' in three or four beats, where the length of each note is a phenotypic gene.

Initialization: The initialization will use the feature pool taken from the original song in the previous step to randomly take features as the smallest unit to randomly form a population of individuals of specific population size, thus forming a population in which each individual has a phenotypic gene and a genotypic gene. The genotypic gene is the feature that constitutes the individual, e.g. [feature1,feature4,feature4]. Given that each feature is a combination of notes, each occurrence in the original song may have a different duration pairing, taking the dominant gene [feature 1, feature 4, feature 4] distance = for example, where the three notes of feature 4 = [do, do, so] can be interpreted in an equal length or a one-to-two length relationship, and the choice of this interpretation is its phenotypic gene.

Mutation: Mutation is an operation at the gene level and creates new traits for the gene pool. Therefore, our mutations will modify the note sequence which is the base element of the genotype. Inverse mutation and switch mutation are examples. The former inverses the order of notes in a single feature, and the latter randomly exchanges the position of two notes in a single feature. Those mutations are musically meaningful because the notes in a feature are expected to follow the same tonality for smooth soundness. The notes feature pool is already in a considerably harmonious melody, so the pitch leaves us little to modify; otherwise, it is very likely to break the consistency of the melody.

Crossover: One-point crossover is employed to allow a piece of music to permutate its feature list. Some features represent the verse part of music while others may represent the intro, chorus, or outro part. A feature-level modification of order increases the narrative and emotional diversity of music.

Fitness function: The fitness function is the weighted sum of a function set consisting of several dynamic rules, each with a dynamic weighting factor. One of the references is the similarity to the user's preferred music, i.e., music can be scored higher if it possesses a sequence of features similar to the pattern. Another reference is the degree of similarity of the individual to the root parent, which scores the soundness of the individual. The reason for this is that the root parent has the best sequence of notes because it comes directly from the dataset. The coefficients change with the evolutionary process, and in earlier generations, rules with user-selected output will have a lower weight than rules with similarity to the root parent, because music should "sound good musically" before being added to more personal preferences, preserving its musicality.
The similarity is measured by the Levenshtein Distance, which is the minimum number of steps required to switch, add and append to transform one string into another. In our scheme, it becomes the least number of note modification steps between two genotypes.

Selection: Selection is done by tournament selection. Three randomly chosen individuals complete with their fitness, followed by the best individual replacing the worst one. Multiple tournaments can be held during a generation.

2.3 Code architecture

The minimum unit of the evolved population is Music. We write the Music as a python class and a list which stands for the population. There is a top-down hierarchy for music such that, Music =[Track, Track, …], Track =[Feature, Feature, …], and Feature =[Note, Note, …]. Below is a deeper introduction to it.

Music: The music object has 3 attributes, tracklist, speed, and fitness. The tracklist includes all tracks that composite the music, the first of which is the base melody -genotype. The speed is a musical term, tick per beat, that represents at what rate all notes are played. Finally, the fitness records the value after evaluation. Besides, some helper functions, such as "save to file" and "visual display notes" are coded in this class.
Track: The Track is an object stored in the tracklist of Music. It dynamically calculates the size and stores the sequence of features in the "feature_list" attribute.
Note: The Note is a class representing a single gene with parameter channel -instrument type, note -the pitch of this note, velocity -the volume of this note, start_time -the waiting time after the last note and before playing this note, and duration -the time length of playing this note.
Runtime data flow: For easier development and evolution process examination, the data of population and parameter tuning data are stored as files individually that can be read and written from the outer program, Population. The list is a file exported by the a python package,"pickle" storing the list of music individuals. Weight.txt saves the weight of each rule for the fitness evaluation.

Chapter 3
Web Application Design

3.1 Design Overview

With the design of this project's implementation, we wanted to achieve a web application where a user would select from multiple music pieces for more to be generated based on their selection by our algorithm. Then they would repeat the process of selecting a piece of music under they reached one they were satisfied with. Each piece of music would also have the option to change the midi instrument so the user could listen to it in a different context. To make this happen we needed a front-end user interface that could make calls to a backend containing our algorithm. That front-end would need to be able to play midi files in the browser, and be able to change the midi instrument, while the backend would need to be able to generate pieces based on the frontend input, and save those midi files in a way that the front end could access them.

3.2 Front end Design

The front end was built using plain HTML and embedded Javascript, with libraries like JQuery to help make API calls, and Bootstrap to help with user interface design. The choice to use plain HTML with embedded Javascript stemmed mainly from the Javascript library we needed to use for reading and playing midi files, as the library could only do both if the code was in the browser. This library is called JZZ, and it was the only Javascript library we found that could read and play midi files in the browser.

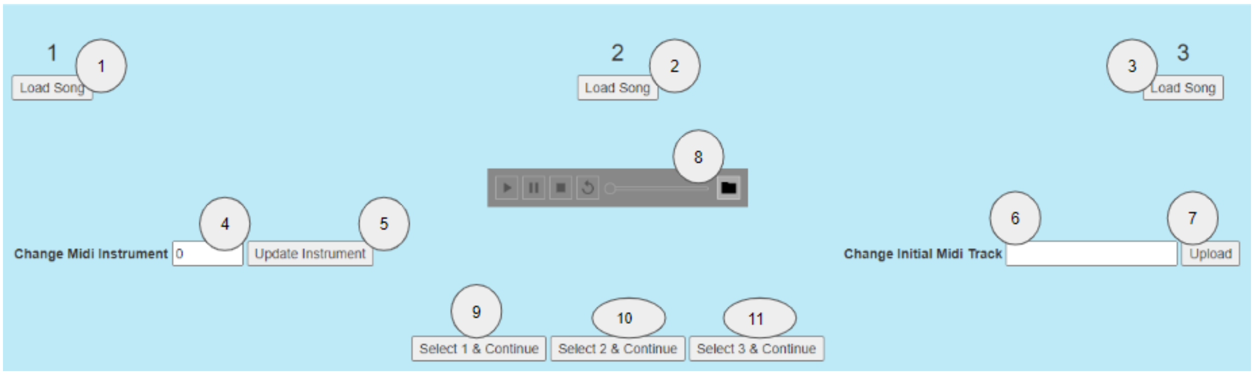

Figure 4.2.1

The user interface itself consists of 3 HTML files corresponding to 3 different pages. Each page corresponds to the 3 stages of the application. For each stage, the legend, (Figure 4.2.4) says what each part does. In the first stage of the application (Figure 4.2.1) the user is given music pieces generated initially by the algorithm from a pre-existing piece of music. The user has the option to change this music piece from the default, which will generate new initial music pieces. They can then listen to and select their favorite piece of music, then the application will go to stage 2.

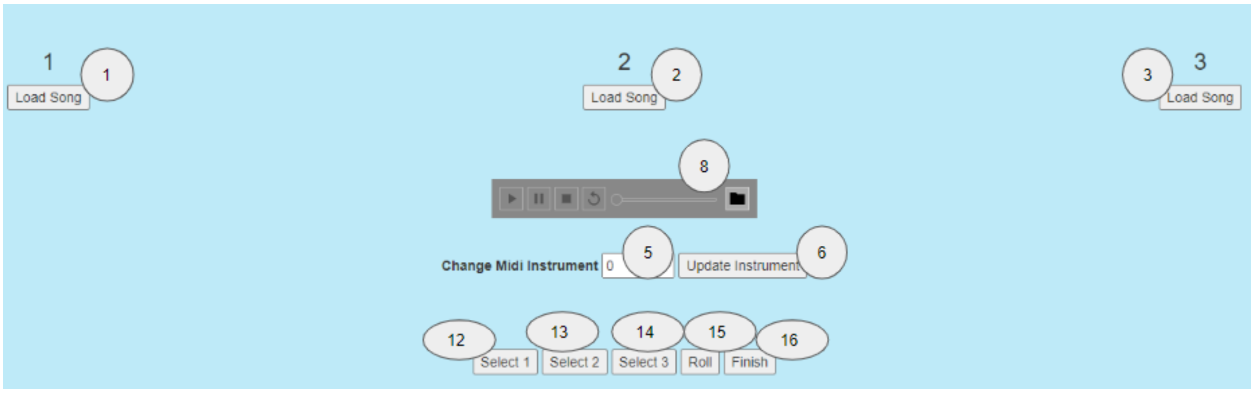

Figure 4.2.2

In stage 2 (Figure 4.2.2), the user is given more pieces of music, generated from their previous choice. The user can repeatedly select which they like best and generate more pieces until they select one they are satisfied with, then they can select to finish the process. The user also has the option to roll, which allows for the music pieces to be regenerated from the user's last selection.

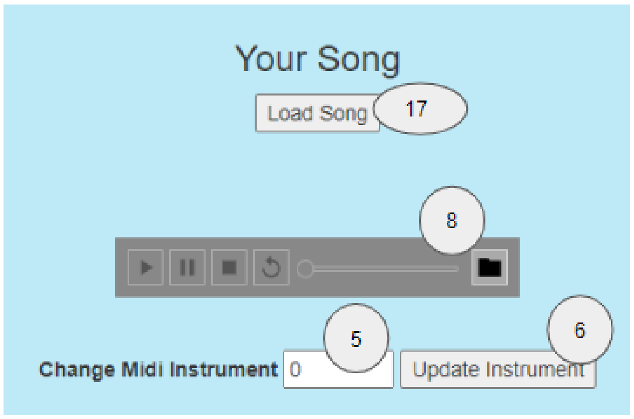

Figure 4.2.3

1. Load piece 1
2. Load piece 2
3. Load piece 3
4. Input midi program value (max. 115)
5. Update midi program value (midi instrument)
6. Filename for starting midi piece
7. Generate new initial midi pieces
8. Control playback for loaded midi file
9. Select piece 1 and continue to stage 2
10. Select piece 2 and continue to stage 2
11. Select piece 3 and continue to stage 2
12. Select piece 1 to generate new music pieces
13. Select piece 2 to generate new music pieces
14. Select piece 3 to generate new music pieces
15. Regenerate music pieces from last selection (Roll)
16. Finish process on last selected music piece
17. Load your song

Figure 4.2.4

In the third stage (Figure 4.2.3), the user is presented with the piece of music they like best to listen to. In all three stages, the user can also change what midi instrument is used to play the midi files, so the user can hear the music pieces in different contexts. After modifying the library we are using for the midi files, JZZ, we were able to pass back a parameter to allow the user to increase the instrument value, allowing for the functionality of changing the midi instrument used to play the midi file. The HTML pages associated with the first two stages also utilize the JQuery Javascript library to call the back end API when a choice is made by the user. This function also triggers a page change when the call is related to making a selection that would take the user to the next page.

3.3 Back end Design

The back end was based around the evolutionary algorithm written in python, and the other python files it uses. These were all broken down into 2 python functions in the main file, the first initializes the midi files the user will first listen two, while the second handles all the midi files generated from the user selections, including when they roll for other choices. These two functions are then called by a python file that acts as a server and uses the Flask python library to create APIs. When the server file starts up it calls the initialize function in the main python file. When an API call is main to the server python file it then calls the second function in the main python file that handles all the user selections. When the backend generates a file, we are currently saving them in the same directory as the front end so they are served along with the front end making them reachable via a port at localhost. This way was chosen mainly for simplicity reasons, and hosting the files separately would make more sense in a production setting.

Chapter 4
Discussion and Future Work

4.1 Discussion

4.1.1 Selection Pressure

As our goal is to generate music according to the user's selection, there is very limited information the algorithm can obtain from the user's several choices. The evolution process usually takes over hundreds of generations before music grows to be "soundness". Since the user would not supervise every generation of the selection, a mechanism should play the role in such missing supervision. We research different solutions.

One of those is to assume users making the same decision in the future generation for a period. By memorizing the characteristics of user-selected music such as feature order and speed, the algorithm runs for considerable generations to produce a significant evolution so the user can make a further decision on that. But some drawbacks come with this method, some bias exists because the number of user selections is too small, furthermore, in some scenarios, all choices are bad, the user has to choose the one they dislike less.

To have a better simulation of user selection, a Long Short Term Memory-based Evaluating Model was proposed (Sak, Haşim 2014). It is a bi-directional neural network that syntax expresses music by analyzing the relative relationship of notes. Input takes a sequence of notes, and Output is a single score. The Bi-LSTM simulates the user scoring, so every new generated music can be given a moc score in the absence of the user.

4.1.2 Music Evaluation

We tried to evaluate music by setting a predefined musical theorem, but there seems no strong connection between the soundness and how much it follows musical rules. Evaluating music is complex because the melody is of a very personalized standard. It is almost impossible to have a general rule for diverse kinds of music. Some rules can check if peace of voice follows rhythmic discipline which can distinguish "music" from "noise" but cannot numerically tell how good the music is.

We discover a past project utilizing a classification neural network, and sets the ratio of true positives and false positives as the quality of music, in other words, the more music is likely jazz, the better the music is considered to be. (Ramirez, Rafael, 2008) A previously discussed paper evaluates music by a statistical method. (Farzaneh, Majid, and Rahil Mahdian Toroghi, 2019). Building a large feature database of diverse styles of music by as many composters as possible. The author believes that in a piece of generally good music, the frequency of features that appears in peace of music should approach the frequency that this feature appears in the global dataset.

4.2. Future work

Consistent bar length: The current musical features are not aligned with multiples of bar length, resulting in rhythmic inconsistencies, and in the future, the musical features that make up an individual should correspond to bar length. If two parents have the same bar length, the features can be aligned with multiples of the bar length to obtain a better rhythm. It is common that a measure occupies a different number of notes; some music has 2 quarter notes in a measure (Beethoven's 5th movement), while another s has 4 quarter notes in a measure (a twinkle twinkle little star). We can find the least common multiple.

Average dissimilarity allowed: Current fitness function is mostly based on the similarity of the selected string and the individual string. Every single character is compared; however, in the scenario of music, some micro differences may not influence the soundness. Therefore, a tolerance rate is suggested to allow two pieces of music to be “equally fit” with some minor differences. (Madsen, Søren Tjagvad, and Gerhard Widmer,2005)

Harmonization: Harmonization accompanying melody improves the soundness and increases the diversity of music. One of the solutions is to employ a composer of harmony. A bass evolutionary composer is introduced with pre-defined harmony data. (De Prisco, Roberto, Gianluca Zaccagnino, and Rocco Zaccagnino, 2020). Our architecture is extendable because multi melody or harmonization can the tracklist preserves space for other instrument tracks.

Front end: In this project, we are currently using the Javascript Library JZZ to read and play midi files, as it was the only Javascript library that offered these functionalities. The library was slightly modified to allow for a change in midi instrument sounds in a midi file when playing, but due to the limited documentation of the code and the time we had available, we were unable to make any further modifications to the library as they would have taken quite a bit of time. In order to achieve functionalities we wanted to add like playing a midi file from a specific timestamp in the file, changing the tempo, and more precisely changing the midi instrument, building a new midi Javascript library based on JZZ or other midi libraries, that more so handle midi data extraction, would be a good idea. While it would take quite a bit of time, it would allow for a midi library that better suits what our project is trying to accomplish.

With the above improvements, we also wanted to change the user interface to make use of the new functionalities. The first change would be to make a method of selecting a specific instrument for the piece of music. This would require a precise way to change the midi instrument for a piece of music, as well as a way to select all possible midi instruments available. The second thing would be to allow for tempo change, the speed at which the music piece is played. For the front end, we would need to have a way in the midi library to modify the tempo of a midi file, then we would need to have a way to increase and decrease the tempo of the music piece from the user interface.

References

Zhou, Zhenkun, Ke Xu, and Jichang Zhao. "Homophily of music listening in online social networks of China." *Social Networks* 55 (2018): 160-169.

Gârbacea, Cristina, et al. "Low bit-rate speech coding with VQ-VAE and a WaveNet decoder." *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019.

Eiben, Agoston E., and James E. Smith. "What is an evolutionary algorithm?." *Introduction to evolutionary computing*. Springer, Berlin, Heidelberg, 2015. 25-48.

Sak, Haşim, Andrew Senior, and Françoise Beaufays. "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition." *arXiv preprint arXiv:1402.1128* (2014).

Ramirez, Rafael, et al. "A genetic rule-based model of expressive performance for jazz saxophone." Computer Music Journal 32.1 (2008): 38-50.

Farzaneh, Majid, and Rahil Mahdian Toroghi. "Melody generation using an interactive evolutionary algorithm." arXiv preprint arXiv:1907.04258 (2019).

Wallis, Isaac, et al. "A rule-based generative music system controlled by desired valence and arousal." Proceedings of 8th international sound and music computing conference (SMC). 2011.

Al Biles, Rochester Institute of Technology. “Evolutionary Music Tutorial” http://igm.rit.edu/~jabics/EvoMusic/BilesEvoMusicSlides.pdf.

Madsen, Søren Tjagvad, and Gerhard Widmer. "Exploring Similarities in Music Performances with an Evolutionary Algorithm." FLAIRS Conference. 2005.

De Prisco, Roberto, Gianluca Zaccagnino, and Rocco Zaccagnino. "Evocomposer: An evolutionary algorithm for 4-voice music compositions." Evolutionary computation 28.3 (2020): 489-530.

“JZZ.js: Midi Library for node.js and web-browsers.” https://jazz-soft.net/doc/JZZ/.

Dhariwal, P., Jun, H., Payne, C., Kim, J. W., Radford, A., & Sutskever, I. (2020). Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*.