Music Generation using Generative Models

Venkata Saai Praneeth Thota   Pranav Sarvesh Kulkarni   Mike Li   Ankith Rajendran

Abstract

Music Generation Using Deep Generative Models has emerged as a fascinating area of research, offering promising avenues for creating human-like and harmonious compositions. The main aim of this study is to explore the potential of sequence-to-sequence models in generating human-like music compositions, contributing to the advancement of deep learning techniques in the field of music generation. In this project, we explored various architecture styles for modeling music compositions, including simple decoder-only architectures, BiLSTM for bidirectional context learning, transformer-based mechanisms for both monophonic (chordless melodies) and polyphonic (chords included melodies) for single and multi-instrument setups. Our investigation revealed that the choice of input representation for MIDI files significantly impacts model performance. Furthermore, we observed that transformer-based models with chord information produce harmonious compositions compared to other models. The code is available at `https://github.com/ProPranu6/MusicGen/`

Introduction

Music generation using deep generative models has gained significant attention in recent years, leveraging advancements in deep learning techniques. This project proposal aims to explore the applications of sequence-to-sequence models for crafting human-like and harmonious musical compositions. Additionally, the project aims to investigate the potential of generating music tailored to specific artists or genres, leveraging the power of advanced encoding techniques like Variational Auto Encoders (VAE) adding that help add a nuanced layer of style to the generation process.

This project focuses on the exploration of the intersection between artificial intelligence and artistic expression. Music, with its temporal structure and multi-instrumental complexities, presents a unique challenge for deep learning models. Unlike the vision models that deal with static data like images, music generating models are required to capture dependencies across sequential events. Although, such modeling looks similar to natural language modeling, unlike in natural language processing, there is more than one type of token prediction at every time step as music encompasses various elements such as chords, melodies, and rhythms, coming from single/multi instruments each contributing to its overall composition.

A wide range of methodologies and model architectures have been proposed to tackle the task of music generation. Early approaches predominantly focused on simple recurrent neural networks (RNNs), to capture temporal dependencies present in the music. Abundant research has been done later to improve recurrent networks' performance in capturing longer temporal dependencies by making use of memory-based recurrent units like Gated Recurrent Units (GRU), Long Short Term Memory (LSTM) etc. Recent advancements have seen the emergence of model that use sliding window based context like convolutional neural networks (CNNs) for learning the latent variable representations leveraging the spatial dependencies present in the music sheets. Furthermore the use of attention-based layers in models, facilitate generating music compositions using selective context.

(Skúli, 2017) and (Zachary, 2020) utilized LSTM networks for single-instrument music generation, laying the groundwork for subsequent research. DeepMind's WaveNet (van den Oord et al., 2016) showcased the effectiveness of CNNs in generating raw audio, while (Yang et al., 2017) introduced MidiNet, leveraging DCGANs for multi-instrument music generation. (Dong et al., 2017) pushed the boundaries further with a multi-generator approach, facilitating the synthesis of complex, multi-instrument compositions.

Recent endeavors have also explored the application of transformers, inspired by their success in natural language processing. (Shaw, 2019) introduced MusicAutobot, a multi-task engine combining BERT and Transformer-XL for music generation and harmony creation. Furthermore, innovative approaches such as the Transformer-GAN proposed by (Neves et al., 2022). demonstrate the potential of conditioning music generation on human sentiment, adding a layer of emotional depth to the process.

In parallel, researchers have investigated various neural network architectures and training strategies to enhance music generation capabilities. (Jiang et al., 2019) introduced a bidirectional recurrent network for music generation, capturing global dependencies within musical sequences. Abbou explored the performance of LSTM networks with attention mechanisms for piano score generation, emphasizing the importance of structural coherence in generated compositions. Mohanty et al. proposed a hybrid model combining CNNs, RNNs, and GANs, aiming to preserve temporal information while generating musical melodies.

Additionally, studies like that of (Hewahi et al., 2019). have focused on specific musical styles, such as Bach's compositions, employing LSTM neural networks to capture intricate patterns and nuances. These endeavors highlight the ongoing pursuit of refining machine learning techniques for music generation, addressing challenges such as long-term coherence and expressive quality.

This project seeks to contribute by leveraging sequence-to-sequence models for music generation building upon existing research and methodologies. Specifically, we aim to divide the task of music generation into generating monophonic and polyphonic compositions, and explore the ability of attention-based encoder-decoder models in generating human-like music compositions.

Methods

Algorithms

A. DECODER-ONLY LSTM NETWORK FOR POLYPHONIC MONOPHONY MUSIC

Initially, the baseline model was composed of a decoder only architecture and the task was reduced to generating polyphonic monophony music compositions, which are compositions of music involving multiple instruments/tracks each with an independent line of melody, that doesn't contain any harmony/chords. The decoder architecture consisted of stacked up layers of LSTM cells, and provided an indicator of the possible frailties of the decoder only architecture.

From the tracks generated by this architecture, we came to the conclusion that the model was unable to capture the style of the original track, and the model was simply replaying the input along with its pauses. Thus,it was concurred the tracks produced by the model was incoherent.

Unlike natural language, where tokens (words) change from one time step to another very frequently, the pitches in music compositions are played for long consistent periods of time constantly and only change rarely. In addition, in autoregressive models without initial context, like the aforementioned decoder-only models, the predictions are vulnerable to deviate from ground truth probability distributions if the initial predictions are not appropriate.

B. ENCODER-DECODER TRANSFORMER FOR MONOPHONIC MONOPHONY MUSIC

To tackle this, we decided to improve on the previous model by introducing a encoder-decoder architecture that stores a portion of the original composition, called the *cue* which was passed on as an additional input.This model improves on the previous model by basing its predictions not just on the past t time steps but also on the style of a part of the original composition, known as the cue, which is encoded by the encoder and serves as the context vector to the decoder. Furthermore, the presence of self-attention, masked-attention and cross-attention layers in the transformers allows for selective context-learning during encoding as well as decoding processes. This allows the model to learn much longer temporal dependencies and generate much coherent compositions. To tackle the redundancy in pitches we employed an algorithm called $time\ tokenization$, and $rest\ note\ sampling$. Furthermore, to simplify the learning task on the limited data we could use, we downgraded the task of music generation to single-instrument monophonic (no-chords in the music) compositions as discussed in subsection .

However, monophonic melodies generated by the model don't sound as harmonious as polyphonic melodies. So, to realize our aim of generating harmonic compositions but also not complicate the learning process, with limited data ($\sim 0.2\%$ data), we came up with a middle ground for the task to be chosen.

C. ENCODER-DECODER TRANSFORMER FOR MONOPHONIC POLYPHONY MUSIC

Monophonic polophony music indicates multiple-note generations ($polyphony$) simulataneously (or chords) by single-instruments ($monophonic$). We have considered $Piano$ as our instrument as it is most widely used primary instrument and all the other instruments are considered conditional on the harmony of piano (Tham, 2021)

C.1. MODEL COMPONENTS

The proposed model consists of several key components that work in harmony to generate music:

Input Layers: Two input layers, X_{inp} and $X_{promptinp}$, accept sequences of single-instrument music tokens with variable lengths. $X_{promptinp}$ serves as a prompt for the decoder.

Embedding: The note_to_vec model embeds each music token into a dense vector representation of size 100. The output is then passed onto the Transformer Encoder.

Transformer Encoder: Two TransformerEncoder layers process the embed input sequence X, capturing long-range dependencies and generating a contextualized representation. The layer uses self-attention to get attention scores and uses them to generate contextualised embeddings for each input music token

Transformer Decoder: The prompt input sequence Y is embedded using the same note_to_vec model and later processed by three TransformerDecoder layers. The first layer makes use of masked self-attention and embeds contextual information of all the tokens that occur before the current token. The second layer attends to both the embedded prompt sequence Y and the encoded input sequence X, using cross-attention, allowing the decoder to incorporate information (like style of the composition etc.) from the input sequence.

Output Layers: TimeDistributed Dense layers are applied to the decoder output Y, producing a softmax output over the music token vocabulary for the instrument.

Loss and Metrics: SparseCategoricalCrossentropy is used as the loss function, with the option to ignore a specific class (e.g., a Out Of the Vocabulary (oov) token). Accuracy is also computed during training and validation to monitor the model's performance.

Model Compilation: The model is compiled with the Adam optimizer (learning rate 1e-3) using the specified losses and accuracy metric for a $batch\_size = 128$.

C.2. Architecture Advantages

This encoder-decoder transformer architecture leverages the self-attention mechanisms and parallelization capabilities of transformers to capture long-range dependencies and generate coherent music sequences. The modifications made to handle multi-note (polyphonic) data, identify and tokenize chord information, embed identified music tokens, and use separate input layers for the input sequence and prompt are tailored to the task of music generation. This design enables the creation of complex music across multiple pitches.

Data

For the implementation of generative models,the Lakh Pianoroll Dataset (LPD) has been utilized, which is a collection of 174,154 unique pianoroll representations. Every pianoroll is stored in a MIDI (Music Instrument Digital Interface) file representation, involving multiple tracks - Piano, Drums, Guitar, Bass, Strings. These files are of varying lengths depending upon the type of composition short melodies, loops, long symponies, suits or songs.Information regarding the dataset can be found here

However, throughout the project we considered only 10,000 out of 174,154 MIDI files (which is 5.74% of the entire data) at a beat resolution of 8 with a tempo in the range of 60-120 bpm (beats per minute). These 10,000 files are processed to make 2.64M sequences each of length 300 timesteps / 38 beats / $\sim$28 seconds. The train validation and test split of these sequences is $64\%, 16\%$ and 20% respectively. The prime reason behind considering only such a small subset is plainly the time and resource limitations we have. Considering the entire dataset is estimated to take about 79hrs of training time for just 1 epoch on CARC's A100 40GB GPU, and even with altering the *steps_per_epoch* parameter to the floor of *train_length.shape/batch_size*; it still takes around 30 hours per epoch. To make the project feasible, we had to draw these boundaries on our dataset size.

Data Preprocessing

As mentioned in the above section, we will mainly be working on Music Instrument Digital Interface (MIDI) data that contains a series of music notes, pitches, and other information related to the musical composition as time series data. We followed different pre-processing strategies for each of our algorithms mentioned in subsections A. , B. and C.

A. Polyphonic Monophony Music

For this algorithm we considered the 10,000 MIDI files and pruned away all the velocity information, percussive chords (chords that are played by hitting the instrument's body), pitch-bend information etc from the MIDI files. Instead we only considered binary pianoroll representations each of t time steps long and 128 pitches. Therefore for 5 instruments we have multidimensional array of size $(5, t, 128)$ where a cell with 0 means the note is off and 1 means note on. To mitigate the imbalance in the dataset caused by rest notes we down sampled them by resampling with 0.3 probability. Later we pruned away all the chord information, thus making this is a polyphonic (for multiple instruments) monophony (for single note at a time) music generation. We passed these sequences of categorical variables and average pooled them to create pooled vector representations before sending in to the model for predictions.

Now we considered sequences of length 600 time steps, forming 435,000 sequences in total from the 10,000 MIDI files.

B. MONOPHONIC MONOPHONY MUSIC

For this algorithm we followed the same pruning of information, like velocity, percussive chords, pitch bend information etc., as in the processing for algorithm A. But for this reduced task of generating music compositions for 1 instrument, we only considered piano instrument's music charts. Therefore every midi file information is abstracted to a binary multidimensional array of size $(1, t, 128)$ where a cell with 0 means the note is off and 1 means note on for t number of time steps. Now we pruned away the chord information by considering only one note that is on for each time step. Then we represent the consecutive repetitions of each of these notes by a separate token called the duration token dn where $n \in \{1, 8, 16, 32\}$ represents time step repetitions. Any number of repetitions is represented as a combination of these dn tokens, thereby reducing the redundancy in the sequences. Then we convert both the pitch tokens and time tokens into embedding vectors that the model learns on how to best represent the pianorolls for the task of music generation the at each time step.

This algorithm takes as input two sequences of length 100 and 200 time steps respectively, thereby the model is given a total dataset of 2.64M sequences made from the 10,000 MIDI files.

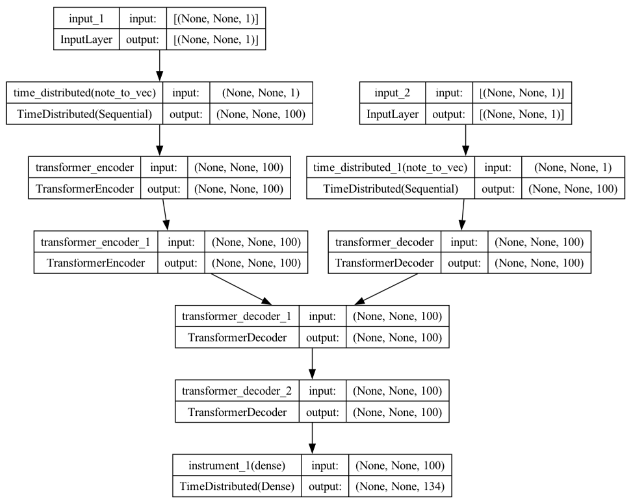
input_1 | InputLayer | input: [(None, None, 1)] | output: [(None, None, 1)]
time_distributed(note_to_vec) | TimeDistributed(Sequential) | input: (None, None, 1) | output: (None, None, 100)
input_2 | InputLayer | input: [(None, None, 1)] | output: [(None, None, 1)]
transformer_encoder | TransformerEncoder | input: (None, None, 100) | output: (None, None, 100)
time_distributed_1(note_to_vec) | TimeDistributed(Sequential) | input: (None, None, 1) | output: (None, None, 100)
transformer_encoder_1 | TransformerEncoder | input: (None, None, 100) | output: (None, None, 100)
transformer_decoder | TransformerDecoder | input: (None, None, 100) | output: (None, None, 100)
transformer_decoder_1 | TransformerDecoder | input: (None, None, 100) | output: (None, None, 100)
transformer_decoder_2 | TransformerDecoder | input: (None, None, 100) | output: (None, None, 100)
instrument_1(dense) | TimeDistributed(Dense) | input: (None, None, 100) | output: (None, None, 134)

Figure 1. Plot of the architecture of the polyphonic monophony model from the algorithm B

C. MONOPHONIC POLYPHONY MUSIC

For this algorithm we followed the same pruning of information as followed in algorithm B. However, to facilitate the model to generate polyphonic music (music melodies with multiple notes/chords playing in a time step), we use a python library $music21$, which parses a given midi file to output list of chords, and note information along with the duration (in sec) for which the chord or note is played. We prune away the duration information and consider only the top 2000 chords made up with same pitch classes into the vocabulary and the rest are mapped to being Rest note 'R'. Then we convert both note tokens and chord tokens into embedding vectors that are learned by the model such that it best represents the input pianorolls for the task of music generation at each time step.

This algorithm takes as input two sequences of length 100 and 200 time steps respectively, thereby the model is given a total dataset of 2.64M sequences made from the 10,000 MIDI files.

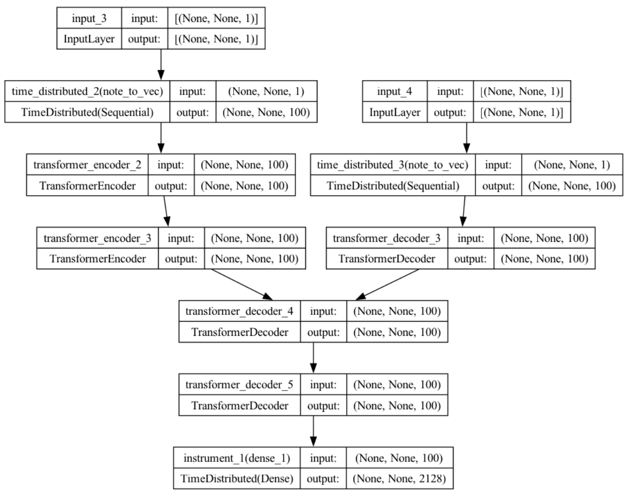
input_3 | InputLayer | input: [(None, None, 1)] | output: [(None, None, 1)]
time_distributed_2(note_to_vec) | TimeDistributed(Sequential) | input: (None, None, 1) | output: (None, None, 100)
input_4 | InputLayer | input: [(None, None, 1)] | output: [(None, None, 1)]
transformer_encoder_2 | TransformerEncoder | input: (None, None, 100) | output: (None, None, 100)
time_distributed_3(note_to_vec) | TimeDistributed(Sequential) | input: (None, None, 1) | output: (None, None, 100)
transformer_encoder_3 | TransformerEncoder | input: (None, None, 100) | output: (None, None, 100)
transformer_decoder_3 | TransformerDecoder | input: (None, None, 100) | output: (None, None, 100)
transformer_decoder_4 | TransformerDecoder | input: (None, None, 100) | output: (None, None, 100)
transformer_decoder_5 | TransformerDecoder | input: (None, None, 100) | output: (None, None, 100)
instrument_1(dense_1) | TimeDistributed(Dense) | input: (None, None, 100) | output: (None, None, 2128)

Figure 2. Plot of the architecture of the monophonic polyphony model from the algorithm C

Evaluation

The evaluation methods of AI-generated music are divided into subjective evaluation, objective evaluation, and combined evaluation. Each of these methods has its own limitations. Some of the key limitations of subjective evaluation are they are costly to perform, require music expertise from the listeners, and the results are often not reproducible (Xiong et al., 2023). On the other hand, the existing objective evaluation metrics are also divided into model-based metrics and music domain metrics. Model-based metrics can involve training loss and reconstruction accuracy(Xiong et al., 2023). One of the key limitations of these metrics is that the alignment of these metrics with music quality is very questionable, and these metrics are limited to specific models (Xiong et al., 2023). Due to the limitations of subjective evaluation methods and objective model-based metrics, we decided to adopt an objective music domain evaluation for the music generated by our different approaches.

Method

We adopt the approach proposed in the article On the Evaluation of Generative Models in Music (Yang & Lerch, 2020). We modified the script provided in the GitHub link in the original paper to adapt to newer Python package versions and our own midi files. And we capture a selected subset of both pitch-based features and rhythm-based features in both absolute and relative ways (Yang & Lerch, 2020). Below is a list of features we capture in the modified script.

Pitch Based Features

- Pitch Count: the number of different pitches within a sample (Yang & Lerch, 2020).
- Pitch Range: subtraction of the highest and lowest used pitch in semitones (Yang & Lerch, 2020).
- Pitch Class Histogram (PCH): an octave-independent representation of the pitch content (Yang & Lerch, 2020).
- Pitch Class Transition Matrix (PCTM): a representation of transition of pitch classes (Yang & Lerch, 2020).

Rhythm based features

- Average inter-onset-interval (IOI) (Yang & Lerch, 2020)

For the features listed, we take the absolute measurement and the relative measurement. We use two sets of midi files for evaluation. Set 1 contains 15 midi files generated by our model, and set 2 contains the same number of midi files from the validation set (not used in training). We repeat this process for both polyphonic music generation and monophonic music generation, keeping the number of samples in each set the same. The absolute measurement results are displayed in table 1 and table 2:

Table 1. Absolute Measures for polyphonic generation

Metric	Mean	Standard Deviation
Pitch Count		
Set 1	25.5	9.426
Set 2	11.5	4.272
Pitch Range		
Set 1	44.8	9.379
Set 2	25.4	10.375
Average IOI		
Set 1	13.337	10.422
Set 2	18.530	9.434

Table 2. Absolute Measures for monophonic generation

Metric	Mean	Standard Deviation
Pitch Count		
Set 1	36.8	15.973
Set 2	17.8	13.614
Pitch Range		
Set 1	58.7	26.922
Set 2	27.9	18.923
Average IOI		
Set 1	0.240	0.127
Set 2	0.870	0.677

As shown in the absolute measure results, for polyphonic music, set 1 and set 2 are statistically different in pitch count and pitch range, but not average IOI. For monophonic music, the difference in all three metrics between the two sets seems not to be statistically significant.

The results of the relative measures involve calculating the intra-set distance and inter-set distance of different features. We also generalize the similarity between two sets by measuring the difference between them using KL divergence and overlapping area (Yang & Lerch, 2020). The results are shown in figure 15, figure 16, table 3, and table 4:

Table 3. Relative Measurement for Polyphonic Music Generation

Metric	Set	Kullback-Leibler Divergence	Overlap Area
Total Used Pitch	Set 1	0.09787	0.37092
	Set 2	0.17542	0.19212
Pitch Range	Set 1	0.37214	0.26396
	Set 2	0.19980	0.45192
Average IOI	Set 1	0.28054	0.11830
	Set 2	0.17905	0.50031
Total Pitch Class Histogram	Set 1	0.18462	0.56666
	Set 2	0.16661	0.58342
Bar Pitch Class Histogram	Set 1	0.16089	0.57246
	Set 2	0.08301	0.66314
Pitch Class Transition Matrix	Set 1	0.31834	0.59428
	Set 2	0.15887	0.40410

Table 4. Relative Measurement for Monophonic Music Generation

Metric	Set	Kullback-Leibler Divergence	Overlap Area
Total Used Pitch	Set 1	0.45996	0.59608
	Set 2	0.46136	0.55054
Pitch Range	Set 1	0.33462	0.59479
	Set 2	0.45122	0.36699
Average IOI	Set 1	0.13162	0.20313
	Set 2	0.05726	0.43467
Total Pitch Class Histogram	Set 1	1.78725	0.72376
	Set 2	0.34173	0.49119
Bar Pitch Class Histogram	Set 1	0.97015	0.70937
	Set 2	0.85283	0.70712
Pitch Class Transition Matrix	Set 1	0.13822	0.57683
	Set 2	0.05708	0.25896

A decently high value for overlapping areas for polyphonic generation in the pitch class histogram and pitch class transition matrix, suggests the model's success in capturing some pitch features presented in the training process. However, the relative evaluation results for monophonic music generation boast a much more impressive higher overlap area in all metrics. This result suggests that generating monophonic music using our model is more successful than polyphonic music generation.

Process

There had been a lot of experimentation and exploration done during this project, which played crucial role in shaping the direction and creative choices made regarding the algorithms used or pre-processing methodologies followed. The subsections delineated here represent major decision points in our project and document the reasons for taking the decision and the experimentation that went behind the reasoning.

A. PIVOT TO GENERATING MONOPHONY (CHORD-LESS) MUSIC MELODIES

The project initially aimed at using recurrent neural network architectures, primarily, LSTM and Bidirectional LSTM (BiLSTM) to generate polyphonic polyphony music.However we soon realised that trying to generate multi-instrument multi-note music melodies with recurrent neural network architectures could be very demanding in terms of compute and data. Because the input shapes to such a modeling recurrent network for taking input music composition for t time steps would be $(batch\_size, t, \#instruments * \#pitches)$. For a 5 instrument model with 128 pitches in MIDI notation, that would mean sampling from distributions of 640 sigmoid units in the final layer of the model for each time step. This is not only computationally expensive but is also non-ideal in generating music compositions, as there is no clear idea as to how many pitches from each of the instrument needs to be sampled and how many instruments in total to be sampled for every time step.

So we pivot the direction of the project to generating multi-instrument monophonic music melodies.

B. DECISION TO USE DISTRIBUTED VECTORS FOR MUSICAL PITCH REPRESENTATION

Generating multi-instrument monophonic music requires the recurrent neural architectures to take as inputs a vector of 5 dimensions, each representing an instrument, and each vector belongs to range $[0 - 127]$ indicating the pitch value it belongs to. These pitch values are initially represented as one-hot vector encodings, where the model tries to do an average pool across instruments for every dimension. However, from the literature review it is realised that due to the overlap in the tonnes of musical notes from one octave to another (e.g $C1$ note has same tonal similarity as $C2$, but different from $B1$), it is ideal to represent musical notes or pitches as distributed vectors so that the model can learn the similarities.

C. DISCUSSIONS ABOUT CONVOLUTIONS ON MULTI-INSTRUMENT VECTOR REPRESENTATIONS

With the word embeddings for each instrument stacked at every time step of a recurrent model, the inputs to the LSTM units are images which can be processed using 1d convolutions to capture cross-instrumental contextual information, however, we soon realised that the usage of learnable convolutions for such small training data (5.7% of data) is leading to overfitting of the model. Hence to ensure, that we make the model more generalisable while including cross-instrumental contextual information, we introduced a pooling layer that is not required to have any parameters that it should learn.

D. DISCUSSIONS ABOUT CHORD REPRESENTATIONS

For chord representation in monophonic polyphony music generation algorithm, we initially tried to capture all the chord combinations, as a series of pitches (both pitch class and octave included), of fixed max-length l. But that would be a total of $(12 * 7 = 84)^{l+1}$ combinations (since there are 12 semitones or octaves and each octave has 7 pitch classes). Since this is an exponentially large number even for chords of small max-lengths, we discussed alternatives to include chord information. We explored the possibilities of capturing all the chords irrespective of any lengths that occur in the training set to be included in the chord vocabulary. But for the 10,000 MIDI file dataset, this resulted in a chord vocabulary of 200,000 chords, meaning the model is now required to have 200,000 softmax outputs which would cause the model to have 140M parameters. Since that is not feasible too, we have instead chosen to consider chords as series of pitch classes ignoring the octaves from which they come from, as the tones of notes of same pitch classes are similar. This means the chords $C1.E1.G1$ is considered to be same as $C2.E4.G1$ that is represented as $C.E.G.$ This chord representation when used to considering only the top 2000 highly frequent chord representations in the train dataset, results in an significant yet efficient chord representation that can capture major chord dependencies in the music sheets.

References

Dong, H.-W., Hsiao, W.-Y., Yang, L.-C., and Yang, Y.-H. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment, 2017.

Hewahi, N., AlSaigal, S., and AlJanahi, S. Generation of music pieces using machine learning: long short-term memory neural networks approach. *Arab Journal of Basic and Applied Sciences*, 26(1):397–413, 2019. doi: 10.1080/25765299.2019.1649972. URL https://doi.org/10.1080/25765299.2019.1649972.

Jiang, T., Xiao, Q., and Yin, X. Music generation using bidirectional recurrent network. In *2019 IEEE 2nd International Conference on Electronics Technology (ICET)*, pp. 564–569, 2019. doi: 10.1109/ELTECH.2019.8839399.

Neves, P., Fornari, J., and Florindo, J. Generating music with sentiment using transformer-gans. *arXiv preprint*, 2022. URL http://arxiv.org/abs/2212.11134.

Shaw, A. A multitask music model with bert, transformer-xl and seq2seq. *Towards Data Science*, 2019. URL https://towardsdatascience.com/a-multitask-music-model-with-bert-transformer-xl-and-seq2seq-e8e9846951f9.

Skúli, S. How to generate music using a lstm neural network in keras. *Towards Data Science*, 2017. URL https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5.

Tham, I. Generating music using deep learning, 2021. URL https://towardsdatascience.com/generating-music-using-deep-learningcb5843a9d55e.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. Wavenet: A generative model for raw audio, 2016.

Xiong, Z., Wang, W., Yu, J., Lin, Y., and Wang, Z. A comprehensive survey for evaluation methodologies of ai-generated music, 2023.

Yang, L.-C. and Lerch, A. On the evaluation of generative models in music. *Neural Computing and Applications*, 32(9):4773–4784, 5 2020. doi: 10.1007/s00521-018-3849-7. URL https://doi.org/10.1007/s00521-018-3849-7.

Yang, L.-C., Chou, S.-Y., and Yang, Y.-H. Midinet: A convolutional generative adversarial network for symbolic-domain music generation, 2017.

Zachary. How i built a lo-fi hip-hop music generator using lstms and recurrent neural networks. *Artificial Intelligence in Plain English*, 2020. URL https://ai.plainenglish.io/building-a-lo-fi-hip-hop-generator-e24a005d0144.

Appendix

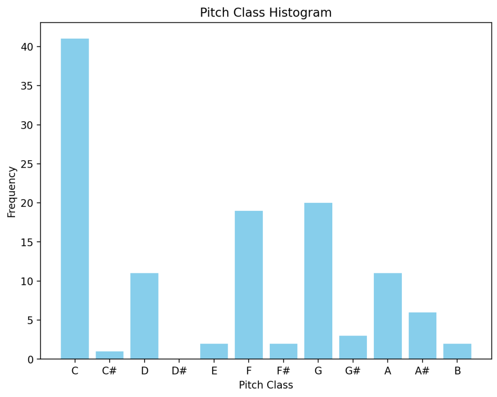

Figure 3. Pitch class histogram for an original composition involving monophonic (chordless) melodies

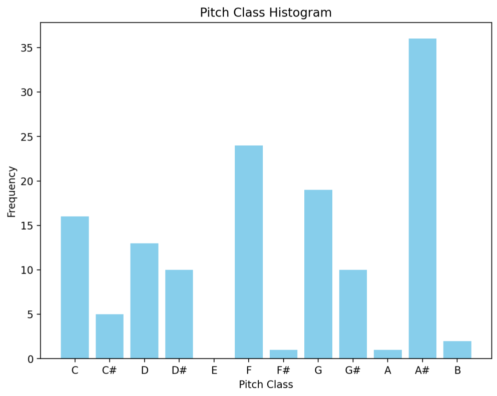

Figure 4. Pitch class histogram for a generated monophonic composition

Figure 5. Comparison of pitch class histograms between ground truth and generated compositions

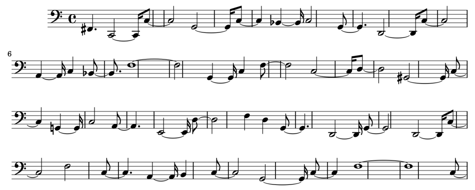

Figure 6. Music sheets for an original composition involving monophonic (chordless) melodies. Click here for audio

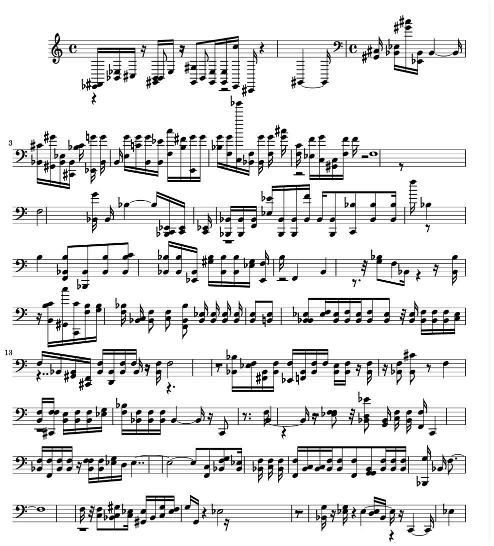

Figure 7. Music sheets for a generated monophonic composition. Click here for audio

Figure 8. Comparison of music sheets between ground truth and generated compositions.

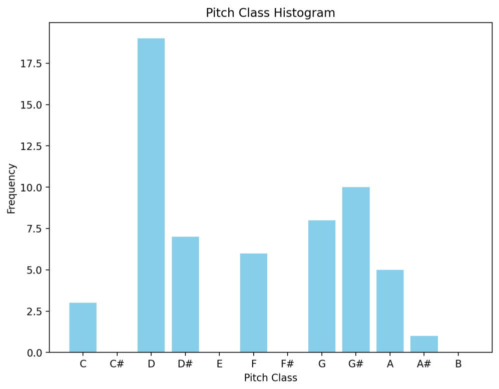

Figure 9. Pitch class histogram for an original composition involving polyphonic (including chords) melodies

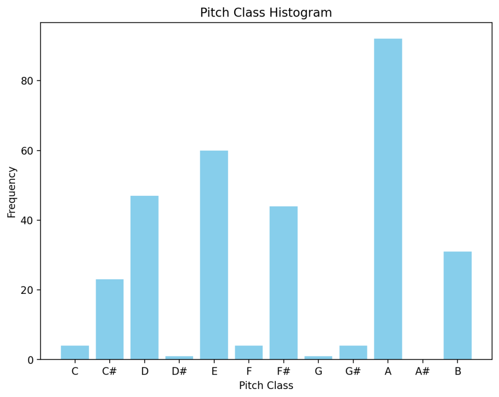

Figure 10. Pitch class histogram for a generated polyphonic composition.

Figure 11. Comparison of pitch class histograms between ground truth and generated compositions

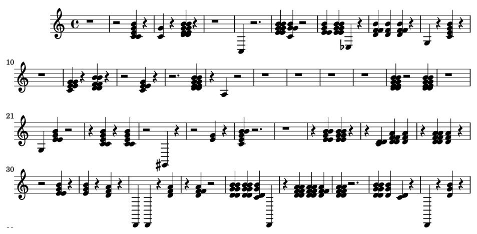

Figure 12. Music sheets for an original composition involving polyphonic (including chords) melodies. Click here for audio

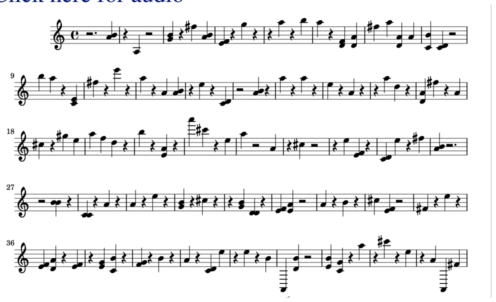

Figure 13. Music sheets for a generated polyphonic composition. Click here for audio

Figure 14. Comparison of music sheets between ground truth and generated compositions

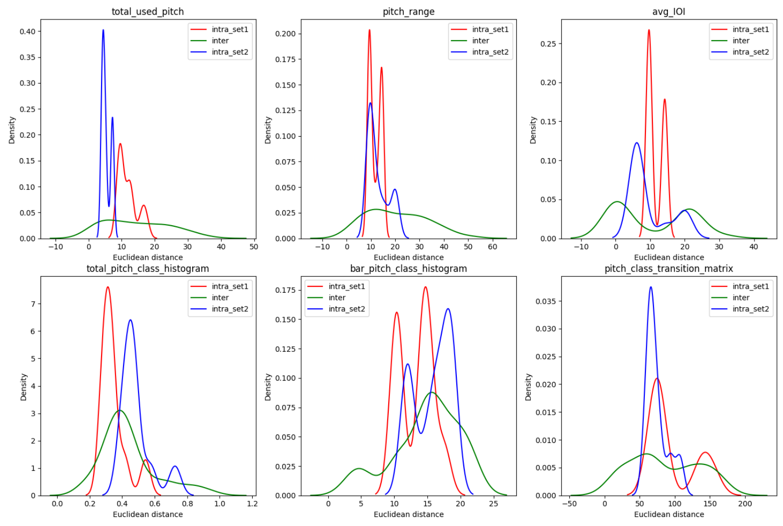

Figure 15. Plots for intra-set and inter-set distances for different metrics for polyphonic music

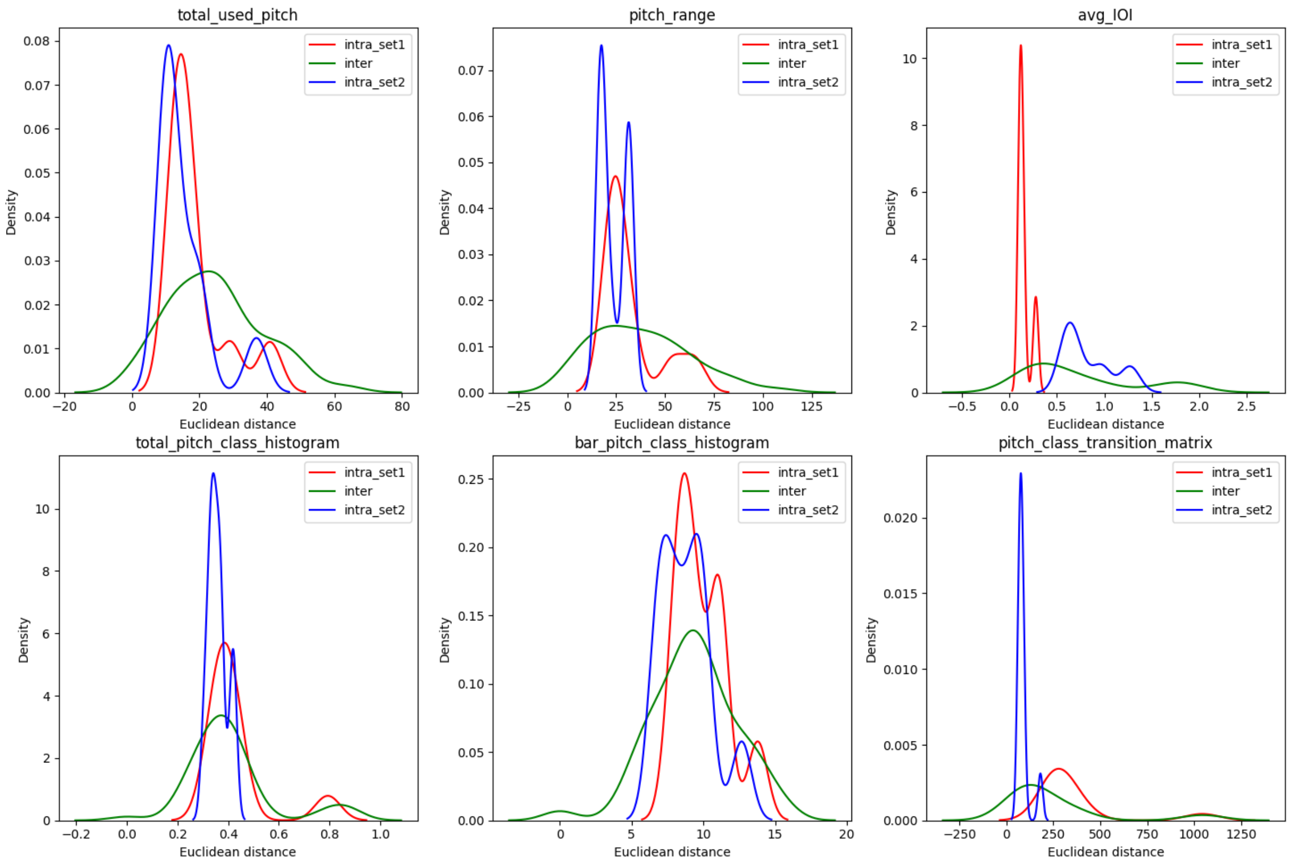

Figure 16. Plots for intra-set and inter-set distances for different metrics for monophonic music