Automatic Drum Accompaniment Generation from Melody

Submitted in partial fulfillment of the requirements
of the degree of

Dual Degree (B.Tech + M.Tech)

by

Rishabh Dahale
(Roll No. 17D070008)

Supervisor:
Prof. Preeti Rao

Department of Electrical Engineering
INDIAN INSTITUTE OF TECHNOLOGY BOMBAY
June 2022

Declaration

Date:

Rishabh Dahale

Roll No. 17D070008

Abstract

Artificial generation of art forms like painting, sketches, music, etc., necessitates intense creativity. In the field of artificial intelligence, art generation is an attractive research area. Music is a very structured yet extremely complicated language. However, when one thinks of composing, one thinks of human intellect, creativity, and emotions. The goal of the sequential method of creation throughout musical history has always been to disrupt music codes. Musicians have created works that are both imaginative and precise. Classical music, for example, is noted for its careful structure and emotional impact.

Transformer and its variants have demonstrated state-of-the-art performance on a variety of NLP tasks and music generation. Despite all of the success of sequential modeling recently, there still exist many issues such as understanding rare words or sparsely occurring events of interest. Another major issue is the presence of biases in the generated output as they mimic the distribution present in the training dataset. In the domain of drum accompaniment, fills and improvisations are important elements of a drum track. They act as cues for the band as well as the audience of an upcoming change in verse. They also help in keeping the audience engaged with the song. These drum fills occur every 8 to 12 bars and hence are highly underrepresented in any dataset. This gives a very unique sequential style to the drum pattern, majorly consisting of a repeating pattern with some minor variations.

In this work, we use the Transformer sequence to sequence model to generate an accompanying drum pattern conditioned on an input melodic accompaniment consisting of notes played by instruments Piano, Guitar, Bass, and Strings. On the manual evaluation of the drum tracks generated by this model, we find that the drum fills and improvisations are largely absent, possibly due to their relatively low representation in the dataset. We propose a novelty function to capture the extent of improvisation in a bar relative to its neighbors. We train a model to predict improvisation locations from the melodic accompaniment tracks. Finally, we use a novel BERT-inspired in-filling architecture, to learn the structure of both the drums and melody to in-fill elements of improvised music, i.e., the fill bars.

Contents

List of Tables

List of Figures

List of Abbreviations

List of Symbols

Chapter 1

Introduction

"If I were not a physicist, I would probably be a musician. I often think in music. I live my daydreams in music. I see my life in terms of music.... I cannot tell if I would have done any creative work of importance in music, but I do know that I get most joy in life out of my violin."
– Albert Einstein

Music has become part and parcel of our lives. It is the art of arranging sounds to produce compositions by varying melody, harmony, rhythm, and timbre. Pitch governs harmony and melody in a more generalized definition, while rhythm is commonly associated with articulation and tempo. The musical part that provides harmonic and rhythmic support for a song's central theme or melody of a song/instrument is known as accompaniment. Different genres and styles of music encompass different types of accompaniments.

One of the most common forms of accompaniment is that of a percussion instrument. This kind of accompaniment mainly provides rhythmic support while maintaining the song's timing structure and overall mood. In modern music, this work is done by a drummer. Drummers need to use their ear and judgment as much as their skills to tone up or tone down their playing to best compliment the song [4]. Bassists and drummers together usually form the rhythm section of a band. A guitarist/vocalist/other musicians can often get away with the odd mistake or bum note, but drummers cannot miss a beat or drop the tempo as such mistakes are easily noticeable. This is in contrast to related generative tasks like image generation, in which a small error in the predicted pixel intensity could be imperceptible to the human eye.

The ambition to learn creativity led to the development of systems with computational creativity on various tasks. Deep learning has revamped the area of generative modeling in recent years. The task of generating music of melodic instruments gained widespread attention in recent times [3, 5, 6, 1, 7, 2]. However, the task of generating an accompanying drum pattern for a given melody is not tackled much, as it is challenging to build a model having computational creativity. A few researchers have tackled this task using Autoencoder (AE) architecture [8, 9]. While such models have shown promising results for generating repetitive drum patterns, they lack creativity in improvisation sections of a song.

Evaluation and assessment of generative systems have proven to be a challenging task. Subjective evaluation of music, while being the ultimate choice, presents its challenges with designing experiments for eliminating bias and controlling other relevant variables. The ever-changing interaction between performance and composition and the abstract meaning of emotion in music makes it even tricky to design objective evaluation metrics.

We try to build a model for generating drum accompaniment for a given melodic accompaniment in this work. We studied various music evaluation methods and proposed some criteria for matching the predicted drum distribution with a professional drummer's distribution. We focus on making a system that can perform a drum fill/improvisation at appropriate locations in the generated drum track.

1.1 Drum Kit & Drumming Style

A collection of percussion instruments like cymbals, which are set up to be played by a single player using drumsticks and feet-operated pedals that control hi-hat cymbal and beater for the bass drum, is known as a drum kit. A standard modern kit contains: a snare drum placed between the players' knees mounter on the stand and played with drum sticks, a bass drum played with pedal, toms of two or more kinds, hi-hat played with sticks and can be opened or closed with foot pedal and one or more cymbal mounted on stands and is played with sticks

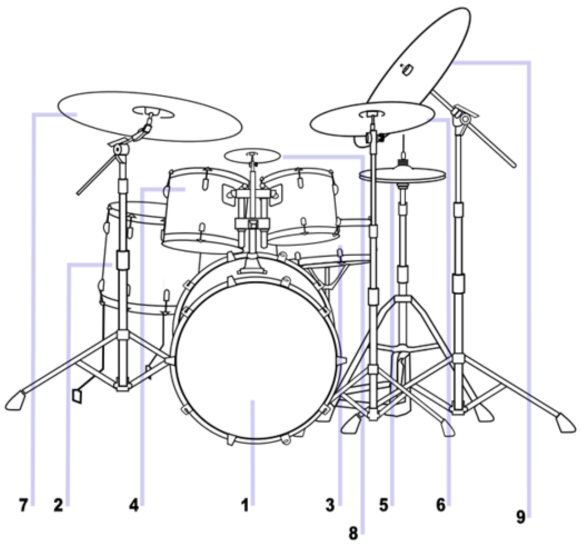

7
2
4
1
3
8
5
6
9
(1) Bass drum (2) Floor tom (3) Snare drum (4) Tom-tom (5) Hi-hat (6) Crash cymbal (7) Ride cymbal (8) Splash cymbal (9) China cymbal

Figure 1.1: Drum Kit

(Figure 1.1).

A musician who is an expert at playing drums is known as a drummer. Western bands that play jazz, rock, or pop include a drummer for keeping time and are also known as "timekeepers." The drummer is known to be the backbone of a band. Their flawless sense of timing provides support and gets the best out of other members of the band. Apart from being the band's timekeeper, they are also required to embellish the song during the performance. This includes accentuating the singer's vocal prowess and the rhythm played by other instruments to make the song expressive.

As the genre of the songs varies the drummer is expected to play in a different style. For e.g., in a bar of rock music, the first beat is also known as the downbeat, and the third beat has similar instrumentation, usually featuring a hi-hat and kick drum onset. Beats 2 and 4 which are commonly referred to as the backbeat, include a hi-hat and snare drum onset. The beat style played by the drummer also influences the overall feel of the song. Generally, either the 8^{th} note pattern or 16^{th} note patterns are played. This means that there are either 8 drum strokes in a bar or 16 drum strokes in a bar respectively. In the 8^{th} note pattern (Figure 1.2a), there are 2 drum strokes in every beat; one at the start of the beat and one in the middle, whereas in the 16^{th} note pattern (Figure 1.2b) there are 4 equally spaced drum strokes in a beat. Since the 1970s, 16th

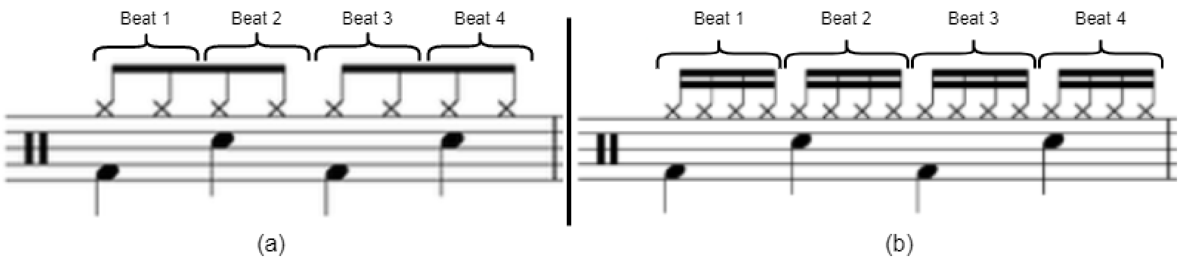


Figure 1.2: (a) 8^{th} note drum pattern where the hi-hat is being played 8 times in a bar (b) 16^{th} note drum pattern; hi-hat being played 16 times in a bar

note subdivisions have been used to produce a funky atmosphere. In ballads, they are frequently played at a slow tempo. The beat of choice for Disco tunes was 16th notes on the Hi-Hat and playing the bass drum on all four beats, which is known as “4 on the floor". The Disco sound and beat persisted in mainstream pop for a while before making a significant comeback with the arrival of club dance music. It has never gone away since then, and it is still the fundamental beat of pop music today.

1.2 Contribution

1. We show that traditional attention-based transformer architectures fail to capture the “improvisation" due to implicit data imbalance.

2. We propose a method to extract drum bars with fills and improvisations by capturing the extent of improvisation in a bar relative to its neighbours

3. We also show that the sampling-based approaches fail to produce a variation of the pattern at the right location. To mitigate this, we show that self-attention-based architectures can extract high quality features, which can be used to predict the location of improvisations.

4. We propose a novel in-filling approach, inspired by BERT that can look at the context of drums and the context of melody and use it to generate the improvised bars.

5. We demonstrate an MLP-based synthesis module for drum improvisation generation from a latent code.

1.3 Outline

Chapter 2 presents an overview of the accompaniment generation methods present in the literature. It also presents a review of the various input and output representation methods and defines the specific task addressed in the present work.

Chapter 3 discusses the dataset used and the details of the preprocessing steps. This chapter also explains our input and output representations and various preprocessing steps used in our work. It also contains the details of the steps taken by us to extract the bars with fills and improvisations and further analysis of these bars. Chapter 4 presents various architectures tested by us for our task. It also provides a sub-module evaluation to get a deeper understanding of the results performed musically.

Chapter 5 contains various musically inclined metrics to evaluate the final samples generated. Finally, chapter 6 contains a summary of the work carried out by us and lays out some of the potential directions for future work.

1.4 Publication

1. Rishabh Dahale, Vaibhav Talwadker, Prateek Verma and Preeti Rao, "Neural Drum Accompaniment Generation From Melody", Int. Society of Music Information Retrieval Conf., Late Breaking Demo Track, Online, 2021

2. Rishabh Dahale, Vaibhav Talwadker, Preeti Rao, and Prateek Verma, "Generating Coherent Drum Accompaniment with Fills and Improvisations", Submitted to Int. Society of Music Information Retrieval Conf., Bangalore, India, 2022

1.5 Acknowledgements

The work in this thesis would not have been possible without the support and contributions received from several other researchers during my association as an undergraduate student with the Digital Audio Processing (DAP) lab in the EE department at IIT Bombay. I extend my foremost acknowledgement to my advisor, Prof. Preeti Rao, for her constant encouragement, enthusiasm, and extremely valuable time-to-time feedback. I would also like to acknowledge Prateek Verma for his timely inputs and guidance with formulating the solution and tackling the issues.

I thank Vaibhav Talwadker for giving insights into a drummer's mindset, and helping to evaluate the drum patterns generated by the model. My sincere thanks to VNS Saketh, the system administrators of DAP lab, who maintained the GPUs which were used for all experiments in this work. Lastly, I would like to thank my family, especially my parents, sister, and my friends for their constant emotional support.

Chapter 2

Overview of Accompaniment Generation

As we now have the knowledge of drum kit and some insights into a drummer's choice of beats, we now turn to the task of conditioned music generation. We aim to generate coherent accompanying drum patterns for given melodic input. In our case, this melodic input consists of the notes played by various instruments. This chapter contains an overview of different methods of music generation present in the literature and a detailed review of selected works. We present various methods of music representation used in literature and also survey recent progress in language modelling as our task is very similar to it.

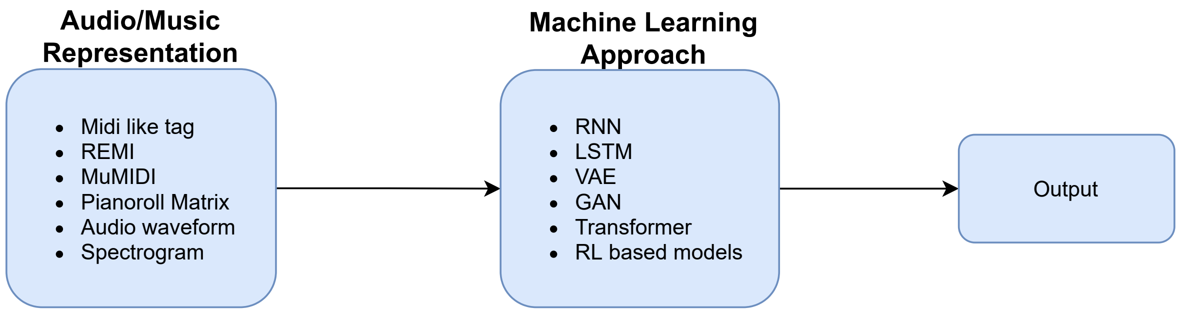


Figure 2.1: General Approach for Music Generation Models

2.1 General Approach and Methods

Figure 2.1 shows the general approach taken for music generation tasks. Audio/Music representation is an important step, as it can help the model abstract musical features easily, making the learning process simple. Several music modeling/representation methods have been studied in the literature. A brief overview of different methods is presented below:

1. **Midi Like Tag:** MIDI is a technical standard that describes a communication protocol that connects a wide range of electronic musical instruments. Instead of storing actual audio samples, high-level information like pitch (MIDI number), loudness (MIDI velocity), the instrument on which it is being played, and the duration of the note is saved. This data is converted to a tag, similar to the one used in the ASR task, which is then fed to the network. Different kinds of tag representation used in literature are:

   (a) **Simple MIDI Tags:** Tags used in this representation are

      i. `SET_VELOCITY<v>` denotes the velocity v of the proceeding `NOTE_ON` event. The velocity of a note can be a value in the range 0-127

      ii. `NOTE_ON<p>` denotes the onset of pitch p

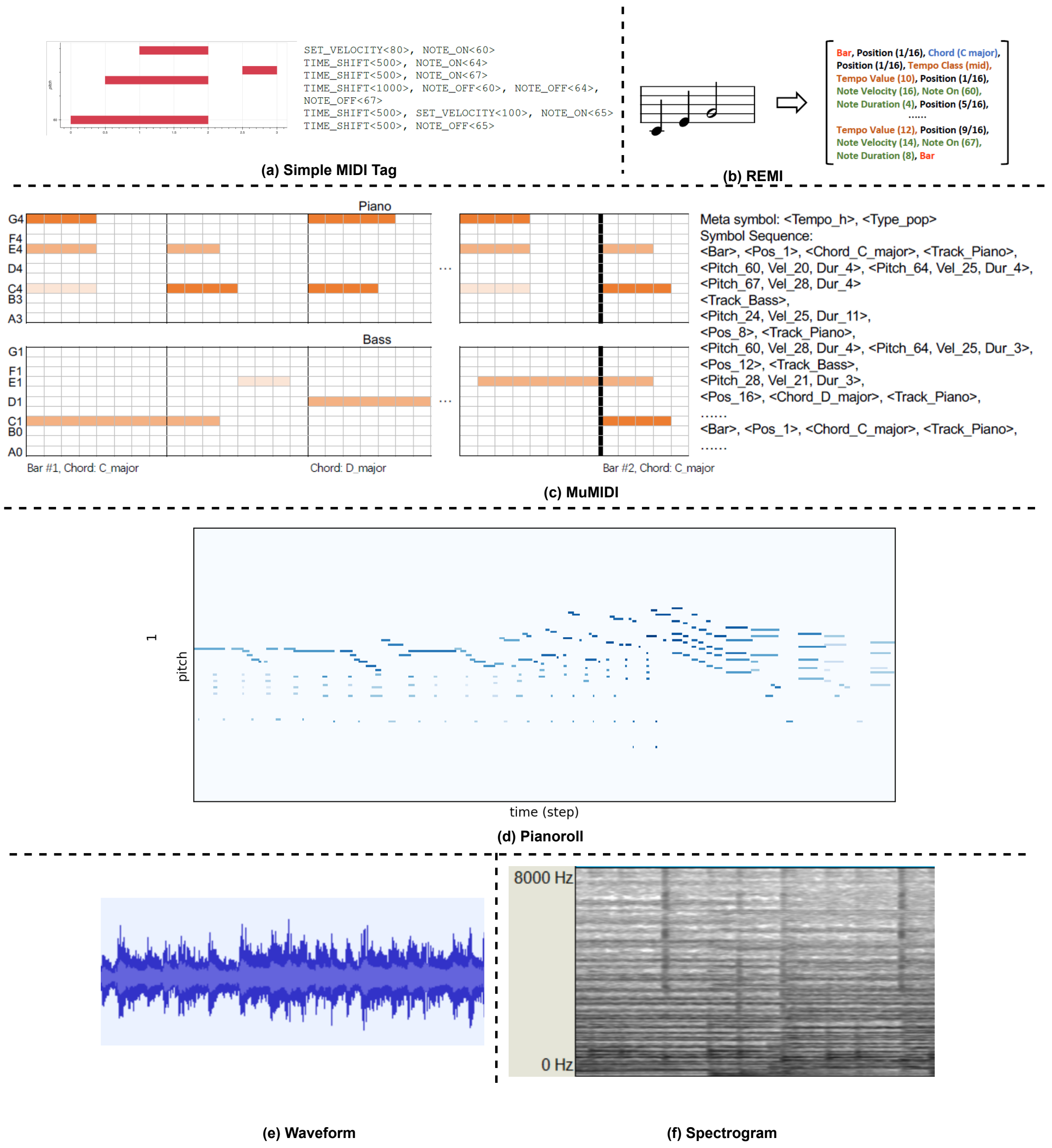
      iii. `NOTE_OFF<p>` denotes the offset of pitch p

      iv. `TIME_SHIFT<m>` denotes shift in time in ms

   (b) **Note Tag:** This notation represents the notes in a quantized beat instead of actual notation. It simply denotes the pitch being played at the given time step, with a different notation for holding a note [7]. E.g., if a beat is quantized in 4 parts and the note C4 is being played, it will be represented as `[C4, C4_hold, C4_hold, C4_hold]`.

   (c) **REMI:** REMI [2] stands for REvamped MIDI. This notation is derived from the simple MIDI tags notation. In this, the timeshift tag is replaced to denote the position in a bar, and the duration of the note is defined in terms of multiple of 32nd note. The chords are extracted to a higher level notation to reduce the burden on the model.

   (d) **MuMIDI:** MuMIDI [3] stands for Multitrack MIDI representation. This is a modified form of REMI to include the instrument information.



```
SET_VELOCITY<80>, NOTE_ON<60>
TIME_SHIFT<500>, NOTE_ON<64>
TIME_SHIFT<500>, NOTE_ON<67>
TIME_SHIFT<1000>, NOTE_OFF<60>, NOTE_OFF<64>,
NOTE_OFF<67>
TIME_SHIFT<500>, SET_VELOCITY<100>, NOTE_ON<65>
TIME_SHIFT<500>, NOTE_OFF<65>
```

```
Bar, Position (1/16), Chord (C major),
Position (1/16), Tempo Class (mid),
Tempo Value (10), Position (1/16),
Note Velocity (16), Note On (60),
Note Duration (4), Position (5/16),
......
Tempo Value (12), Position (9/16),
Note Velocity (14), Note On (67),
Note Duration (8), Bar
```

```
Meta symbol: <Tempo_h>, <Type_pop>
Symbol Sequence:
<Bar>, <Pos_1>, <Chord_C_major>, <Track_Piano>,
<Pitch_60, Vel_20, Dur_4>, <Pitch_64, Vel_25, Dur_4>,
<Pitch_67, Vel_28, Dur_4>
<Track_Bass>,
<Pitch_24, Vel_25, Dur_11>,
<Pos_8>, <Track_Piano>,
<Pitch_60, Vel_28, Dur_4>, <Pitch_64, Vel_25, Dur_3>,
<Pos_12>, <Track_Bass>,
<Pitch_28, Vel_21, Dur_3>,
<Pos_16>, <Chord_D_major>, <Track_Piano>,
......
<Bar>, <Pos_1>, <Chord_C_major>, <Track_Piano>,
......
```

Figure 2.2: (a) Simple MIDI Tags used by [1], (b) REMI representation proposed by [2], (c) MuMIDI representation proposed by [3], (d) Pianoroll representation, (e) Simple Audio waveform, (f) Spectrogram of the audio

2. **Relative Positioning:** In this notation, all the notes are presented in relative position to the previous note played by the instrument. The relative positioning is done based on the number of semi-tones (notes) present between the two notes. For example, a series of notes `[C4, D4, F4, E4]` will be represented by *C4, +2, +3, -1*. This provides an advantage of encoding the relative distance information in the input itself.

3. **Pianoroll Matrix:** The automatic piano inspires this notation. The note events in MIDI are represented as a $N \times T$ matrix, where N is the instrument's pitch range, while T represents the time. Each beat is subdivided into multiple parts to represent smaller duration notes. Each element in the matrix represents the velocity of the note being played.

4. **Audio Waveform:** The waveform is the most direct representation of the audio signal. Audio waveforms are the most simple 1D representation of the signal. Due to recent works on models like WaveNet [10], and NSynth [11], raw audio, even though they are computationally heavy, can now be directly processed.

5. **Spectrogram:** Spectrogram contains much more detailed frequency components of the audio signal. Compared with most traditional manual features used in audio analysis, spectrogram retains more information and has a lower dimension than the original audio.

2.2 Summary

With the recent rise of neural networks, computational music generation has gained renewed interest. Over the past decade, the music generation task has attracted the attention of many researchers. Deep learning algorithms have currently become mainstream in the field of music generation.

For a long time, recurrent neural networks (RNNs) were used to model sequential data. In fact, in 1989, RNNs were used for the first time for the generation of monophonic melody by [12]. Due to the vanishing gradients problem of RNN, storing long historical information was difficult, and LSTMs [13] were developed. Eck et al. [14] demonstrated the use of LSTM for improvising blues music with good rhythm and reasonable structure based on short recordings. Later an RNN-RBM model was proposed by [15] which demonstrated superior results for polyphonic music generation on several datasets. Google brains Magenta team also proposed a Melody RNN model [16] to improve further the ability of RNN to learn long-term structures.

Lately, with the development of complex deep learning architectures like Variational Autoencoders, Generative Adversarial Networks, and Transformers, music generation methods have seen a paradigm shift from RNN to these generative networks. Roberts et al. [19] pro-

Year	**Author**	**Reference**	**Input/Output representation**	**Solution Method**
1989	Todd et al.	[12]	Relative Positioning	Simple RNN
2002	Eck et al.	[14]	Pianoroll	LSTM
2012	Boulanger et al.	[15]	Pianoroll	RNN-RBM
2017	Chia et al.	[17]	Pianoroll	GAN
2018	Dong et al.	[18]	Pianoroll	GAN
2018	Roberts et al.	[19]	Pianoroll	VAE
2019	Lattner et al.	[8]	Onset of Snare and Bass & Beat and Downbeat input	Gated Autoencoder
2019	Wei et al.	[9]	CQT for audio Pianoroll for Drums	Variational Autoencoder (Predicting Drum SSM from melody SSM and then using this to generate drum patterns)
2018	Zhi et al.	[1]	Simple MIDI Tags	Relative Positional Self-Attention Transformer
2019	Donahue et al.	[20]	Simple MIDI Tags	Transformer-XL
2020	Huang et al.	[2]	REMI	Transformer-XL
2020	Thorn et al.	[21]	Drum Machine Control Inputs OR Simple MIDI Tags	Transformer-XL
2020	Huang et al.	[2]	REMI	Transformer-XL
2020	Ren et al.	[3]	MuMIDI	Transformer-XL
2021	Nuttall et al.	[22]	Simple MIDI Tags	Transformer-XL
2017	Yu et al.	[23]	Pianoroll	RL + RNN based GAN
2020	Jian et al.	[7]	Note Tag	RL (Actor-Critic with generalized advantage estimator)

Table 2.1: Overview of previous work on Music Generation

posed a hierarchial VAE model to capture the long-term structure of polyphonic music, which has great interpolation and reconstruction performance. Wei et al. [9] observed that the self-similarity matrix (SSM) of melody and drums show significant correlation and tried to predict the drum SSM from melody SSM using VAE. They used this predicted drum SSM to generate an accompanying drum pattern for the given melody, while [8] proposed a Gated Autoencoder model for the conditional generation of drum pattern while gaining control over the music generation. Dong et al. [18], and Yang et al. [17] used CNN-based GAN architecture to demonstrate their power for music generation. Yu et al. [23] combined reinforcement learning methods to train RNN based GAN for the first time, and [7] have tried to develop a real-time interactive music generation model based on ideas from reinforcement learning. Recently, Transformer models have shown their great potential in music generation [24, 1, 3, 2]. Huang et al. [1] used Transformer architecture for the first time to model long-term architectures to create music. Donahue et al. [20] proposed a multi-instrument input representation for the Transformer model and suggested a pre-training method based on transfer learning, and [2] proposed MIDI-based methods for music representation REMI and used the Transformer XL [25] for sequence modeling to generate piano music.

There has been an increase in interest in the Transformer architecture since it was first introduced [26], particularly for modelling sequential tasks in NLP and music production. The use of an attention mechanism to learn long-term patterns helped it to easily surpass dilated convolution-based methods like WaveNet [10, 27]. They achieve state-of-the-art performance in a variety of natural language problems [28], music/audio understanding [29], and generation tasks [24, 1]. BERT [30], an extension of the Transformer Encoder, was designed to pre-train deep bidirectional representations from an unlabeled text by jointly conditioning on both left and right contexts, causing a huge stir by presenting state-of-the-art results in a wide variety of NLP tasks including Question Answering (SQuAD v1.1) and Natural Language Interface (MNLI). Despite all of the success of sequential modeling recently, there still exist many issues that are relevant to our current work such as understanding rare words or sparsely occurring events of interest [31]. Another major problem is the presence of biases in the generated output, as they mimic the distribution present in the training datasets [32, 33].

Even though there are considerable advances in neural language modeling, the task of finding an optimal decoding strategy remains an open question. Beam search, a method based on the principle of maximum likelihood estimation, is widely used for decoding the text sequence from the predicted distribution. However, it is observed that such decoding often leads to text degeneration even with large state-of-the-art models like GPT [34]. To avoid [35] relied on sampling from the top k most preferred choices for the task of story generation. Although the results of such a sampling-based approach were impressive, they often lead to text that is incoherent and almost unrelated to the context. Holtzman et al. [34] proposed a nucleus sampling approach to overcome this, where instead of selecting the top k most preferred choices, the least number of choices are preferred so that the resulting likelihood is above a threshold. The distribution of these choices is then normalized and sampled. This approach avoids the problem of text degeneration by truncating the tail of the probability distribution.

2.3 Detailed Review of Selected Works

One of the main challenges in drum accompaniment generation tasks is to generate a structurally cohesive sequence. A self-similarity matrix is the graphical representation of a similar sequence in data series. [9] observed that the self-similarity matrix of melody and drums have a correlated structure. They use this observation to generate drum patterns from audio domain music input. The melody is represented by a constant Q-transform spectrogram, whereas a binary pianoroll matrix represents the drums. They use a Variational Autoencoder (VAE) structure to predict a drum SSM matrix. This generated drum SSM is passed through a bar selection module, where top k bars are selected. These selected bars are used to generate the output drums in a bar-by-bar method. This method does not guarantee a rhythmic consistency across the bars.

Another crucial issue in music generation, especially when conditioning on a piece of existing music, is user control. Lattner et al. [8] demonstrate the use of Gated Autoencoder architecture to gain control over the drum patterns by mapping real kick drum onsets to a Standard Gaussian distribution. Samples from this Standard Gaussian distribution are passed through a convolution gated autoencoder to generate drum patterns. The authors introduce an adversarial loss for the model training to make the mapping more constant over time. A further loss that constraints map to have zero mean and unit standard deviation over time is introduced. With the liberty to sample from the learned distribution, the authors present control over the drum generation process.

Huang et al. [1] tried to address the problem of long-term dependencies by proposing an alternate implementation of the relative attention mechanism to reduce the memory requirement of the Transformer from quadratic to linear in sequence length. This allowed them to generate minute-long compositions with a compelling structure. Another approach taken by some researchers [3, 2, 22, 21] is the use of a bigger Transformer-XL [25] model. Huang et al. [2] proposed the REMI notation to exploit the grid structure of music. This representation could be used to represent a single instrument. The authors used it to extend piano samples and demonstrated a better rhythm structure compared to [1]. Building on this work, [3] extended the REMI representation to include a multi-instrument representation and were able to successfully able to generate multi-track songs. Nuttall et al. [22] modified the MIDI tags of nine percussion instruments to represent notes being played by a triplet of pitch, velocity, and start time, and used it with the Transformer-XL model to sequentially generate the drum pattern. Thorn et al. [21] demonstrated three experiments with the Transformer-XL model with varying input and output representation and control. In two of the experiments, the authors tried to control the drum machine while in the third experiment they tried to generate the drum pattern directly. While the Transformer-XL model facilitates the generation of longer duration (musical) sequences, they still suffer from the same issues of biases in dealing with the implicit data imbalance that exists in the training dataset [36, 37].

2.4 Task Definition in the Present work

In an accompaniment generation context, creating a coherent drum pattern with apposite fills and improvisations at proper locations in a song is a challenging task. Drum beats tend to follow a repetitive pattern through stanzas with fills/improvisation at section boundaries which generally occur every 8-12 bars. In this work, we tackle the task of drum pattern generation conditioned on the accompanying music played by four melodic instruments – Piano, Guitar, Bass, and Strings. We use the Transformer sequence to sequence model to generate a basic drum pattern conditioned on the melodic accompaniment to find that improvisation is largely absent, attributed possibly to its expectedly relatively low representation in the training data.

We propose a novelty function to capture the extent of improvisation in a bar relative to its neighbors. We train a model to predict improvisation locations from the melodic accompaniment tracks. Finally, we use a novel BERT-inspired in-filling architecture, to learn the structure of both the drums and melody to in-fill elements of improvised music. To mitigate the issues with audio synthesizing, especially for music which have a rich harmonic dependency, all the work is being carried out in symbolic domain. We use a mixture of MIDI tag representation and pianoroll representation to suit the strengths of individual models.

This page was intentionally left blank.

Chapter 3

Dataset

In this work, we use the Lakh Pianoroll Dataset (LPD) [18, 38], a subset of Lakh Midi Dataset (LMD) [39]. This dataset is a collection of 174,154 songs. It contains multitrack pianorolls stored in a special format for efficient I/O. The songs in LPD are further processed to generate the following subsets of the dataset:

1. **LPD-Matched:** This contains 115,160 multitrack pianorolls derived from the matched version of LMD. These files are matched to entries in the Million Song Dataset (MSD).

2. **LPD-Cleansed:** This subset contains 21,425 multitrack pianorolls, derived from LPD-matched after processing for the following:

    (a) Songs with more than one time-signature change events are removed

    (b) Songs with the time-signature of 4/4 only are selected

    (c) Songs whose first beat do not start at the time zero are removed

    (d) Only one file is kept that has the highest confidence score in matching for each song. The matching scores are the confidence on whether the MIDI file match any entry in MSD

Moreover, there are two versions of the above splits:

1. **LPD-5:** This dataset contains five instruments: Piano, Guitar, Bass, Drums, and Strings.
2. **LPD-17:** This dataset contains seventeen instrument tracks: Drums, Piano, Chromatic Percussion, Organ, Guitar, Bass, Strings, Ensemble, Brass, Reed, Pipe, Synth Lead, Synth Pad, Synth Effects, Ethnic, Percussive, and Sound Effects.

We use the LPD-5 cleansed, which contains 21,425. Among these five instruments, we choose Piano, Guitar, Strings, and Bass as the melodic accompaniment input and drum track as the percussion accompaniment. We condition the percussion accompaniment on the melodic accompaniment. All the songs in this dataset are of 4/4-time signature i.e., all the songs contain 4 beats in a bar and the dimensionality of each bar in this dataset is 128 (pitch) x 96 (time steps) i.e. each beat is divided into 24 parts.

3.1 Data Preprocessing

To compress the input and output representation, we perform the following pre-processing steps:

1. All the songs were trimmed to eliminate the start and end silences in the drum track

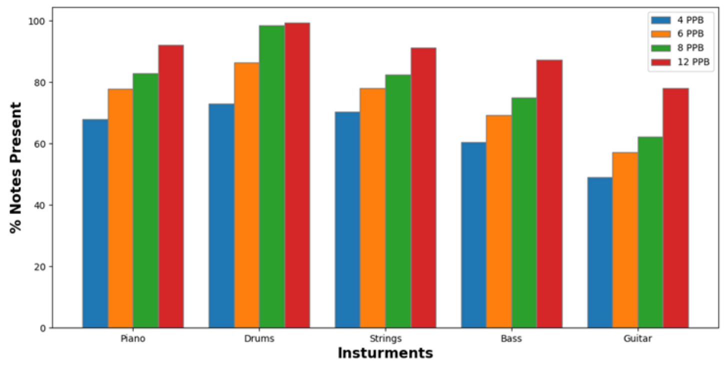


Figure 3.1: Percentage of beats/notes events retained by downsampling each beat to 4, 6, 8 and 12 parts per beat

2. As the dataset has split each beat into 24 parts, a bar of 4 beats requires 96 vectors for representation. This kind of split was done to capture common temporal patterns such

as triplets and 32nd notes. This kind of split, while being detailed, required a massive amount of memory, and hence computation cost increases. We analyzed the percentage of retained beats by downsampling each beat to 4, 6, 8, and 12 parts per beat (PPB). Figure 3.1 shows the percentage of retained beats after doing the above downsampling for all the instruments. It can be seen that while there is a consistent drop in the number of note events retained as the PPB decreases, samples with 8 PPB still retain 98.6% of the drum beats and $\approx 70\%$ of notes played by other instruments while reducing the data size by a factor of 3. Hence we opted for downsampling all the pianorolls from 24 PPB to 8 PPB.

3. Figure 3.2 shows the distribution of usage of various MIDI pitches by all the instruments. As we can see that the majority ($> 95\%$) of the notes being played by Piano, Guitar, Bass, and Strings are in the range of 21-83, we select only these MIDI ranges and discard notes being played outside it. This range corresponds to notes A0 to B5 musically

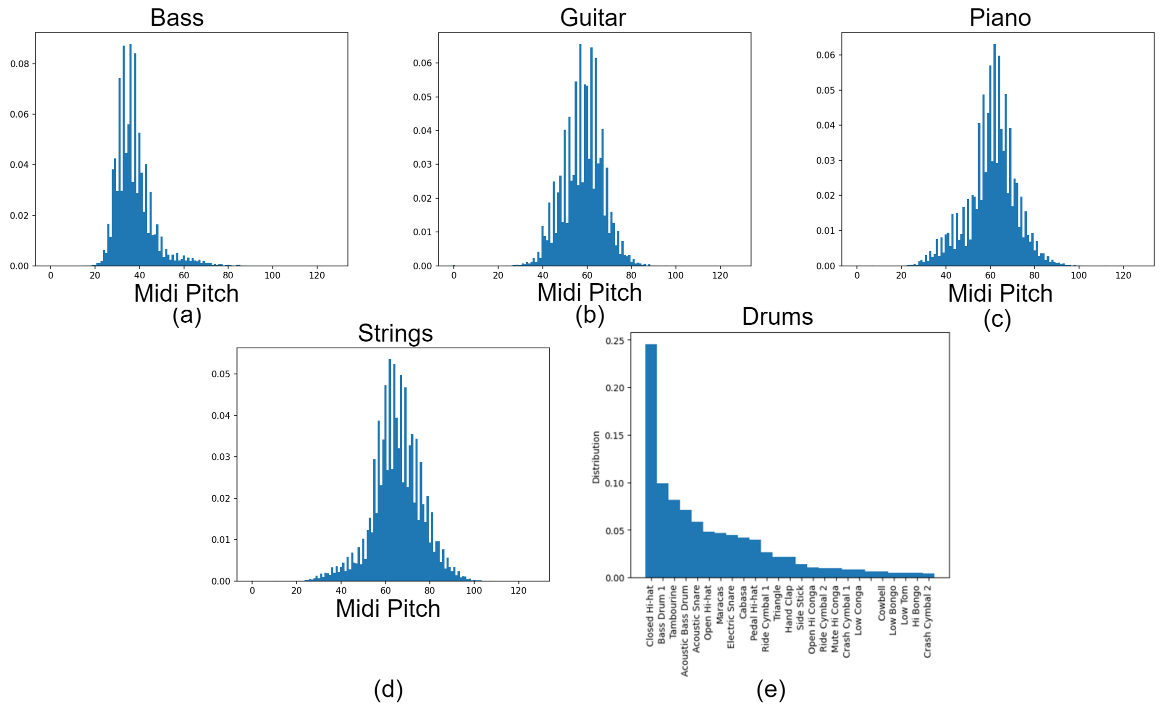


Figure 3.2: Usage of various pitches by different instruments

4. In the MIDI representation of a drum track, there are multiple representations for a single instrument based on different characteristics. We combine such similar instruments together. Table 3.1 gives a list of instruments combined for our task

Instrument Name	MIDI Pitch	MIDI Instrument
Snare Drum	38	Acoustic Snare
	40	Electric Snare
Kick Drum	35	Acoustic Kick Drum
	36	Bass Kick 1
Ride Cymbal	51	Ride Cymbal 1
	59	Ride Cymbal 2
Crash Cymbal	49	Crash Cymbal 1
	57	Crash Cymbal 2

Table 3.1: List of instruments combined in the drum track

5. From figure 3.2, we can see that a majority of the drum strokes are being captured by a selected few instruments. On further analysis it shows that about 85.3% of the drum strokes are being captured by these 16 instruments: snare drum, open hi-hat, close hi-hat, kick drum, ride cymbal, crash cymbal, low-floor tom, high-floor tom, high tom, hi-mid tom, low tom, cowbell, pedal hi-hat, tambourine, cabasa, and maracas. We select only these 16 instruments for our task.

6. To decrease the number of training parameters, we exclude the velocities of the drum track and binarize it. This only keeps the information on whether the instrument is being played or not and discards the loudness of the instrument.

7. As we can expect a fill ever 8-12 bars 1.1, we split all our songs into non-overlapping contiguous 11 bar samples.

Among the 21,425 multitrack songs, 16,832 were used for training, while 4,593 were used for validation. Due to the limited availability of data, we use the validation set as the test set.

3.2 Data Representation

The dataset contains a pianoroll matrix to represent the notes playing in each of the tracks at a given time. The values in the matrix represent the velocity (loudness) of the note being played. As we are working with 11 bar samples, our pianorolls contains $32 \times 11 = 352$ timesteps (sequence length).

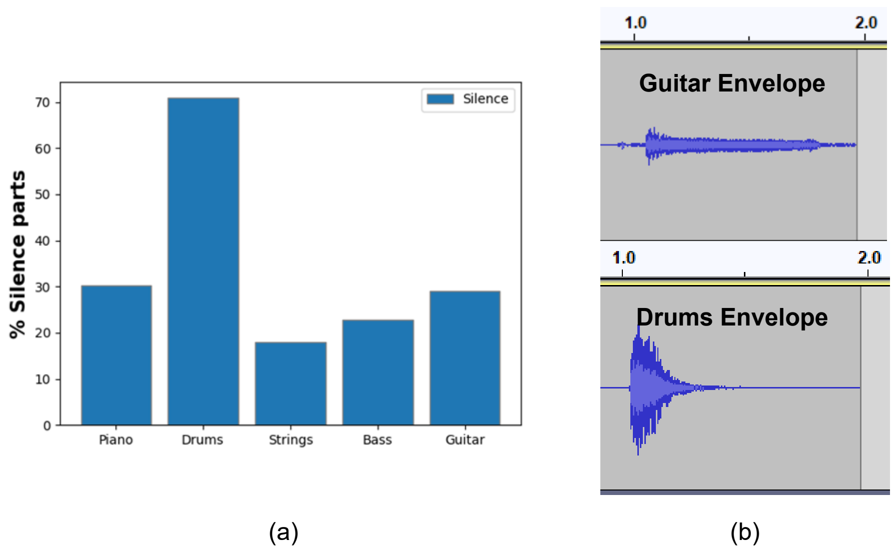


Figure 3.3: (a) Distribution of Silences across instruments in the dataset (b) Envelope of a single guitar pluck v/s that of a single drum stroke

As music is an arrangement of sounds and silences, we analyzed our dataset for the distribution of silences across various instruments. From figure 3.3a, we can see that silences capture a significant portion of any instrument representation. Also, it is interesting to see that drums contain the highest portion of silences ($\approx 70\%$) while strings contain the least amount of silences ($\approx 20\%$). This is because percussion instruments usually decay faster than other instruments like guitar, piano, and strings (figure 3.3b).

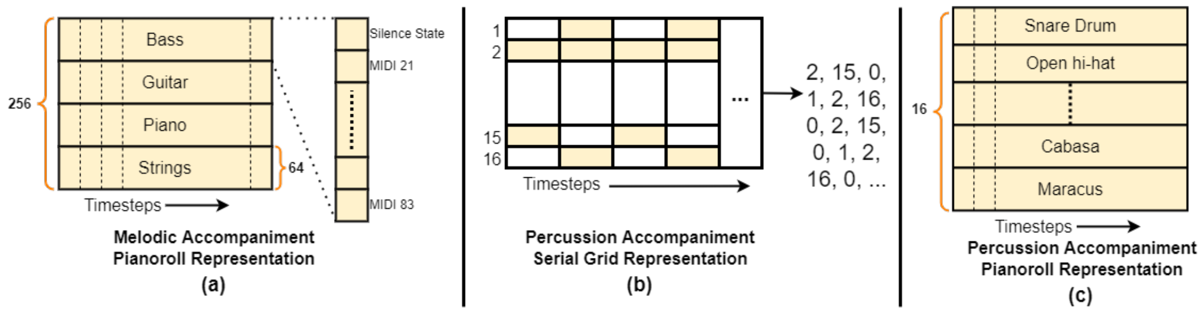


Figure 3.4: Data representation methods used in this work: (a) Melodic accompaniment pianoroll representation; (b) Percussion accompaniment serial grid representation; (c) Percussion track pianoroll representation

As the symbolic domain representations capture the essential high-level feature of music, working in this domain allows us to focus on the musicality of the output instead of worrying about the quality of the actual audio produced. Due to this, we restrict ourselves to symbolic domain inputs and outputs. For the melodic accompaniment input, we use a modified version of the Pianoroll representation. We represent every melodic accompaniment bar in a song by a 256 (feature dimension) $\times$ 32 (timesteps) matrix. The 256 dimensions of this melodic accompaniment are split equally amongst the 4 melodic instruments: Piano, Guitar, Bass, and Strings as shown in Figure 3.4a, giving each of them a 64-dimensional vector. The first dimension in these 64 dimensions is the silence state. This is a binary state which represents whether the instrument under consideration is silent for that timestep. The other 63 dimensions contain the velocities of the MIDI pitches 21-83 being played. A velocity of 0 means that the note is not played. For the percussion accompaniment track, we use 2 different methods for representation depending on the model with which it needs to be used. One of the methods of representation is a pianoroll representation. This is similar to a melodic accompaniment pianoroll. As we have selected only 16 percussion instruments, every bar in the percussion track is represented by a 16 (percussion instruments) $\times$ 32 (timesteps) matrix. This is a binary matrix indicating only the active percussion drum instruments. The other representation used by us for percussion accompaniment is the serial grid representation. This is designed to take the advantage of language modelling tasks. In this representation, only the active percussion instruments which are being played are unfolded into a sequence of tokens. Along with the 16 percussion instruments, we add a silence token and a shift by one token in this representation making a total of 18 tokens. An encoder-decoder transformer structure hugely benefits from the serial grid representation while some other models benefit from pianoroll representation.

3.3 Data Augmentation

The process of applying ways to improve the quantity, diversity, and balance of an insufficient dataset by making realistic changes to existing data samples is known as data augmentation. These variations, which are meant to represent instances that are likely to be encountered during testing, help the model avoid memorising the training dataset's quirks. Modifying the input data samples or the extracted features (adding dropouts in the model) can be used to augment the data. The addition of a little amount of random noise is a simple form of augmentation. This is known to improve the model's robustness by broadening the distribution that it learns. Inspired from the sensor dropout methods in robotics [40] we propose the following data augmentation methods for our task. While common data augmentation methods are used in all the models, we introduce supplementary methods when needed.

3.3.1 Common Data Augmentation Methods

The following data augmentation methods can be applied to each individual input of the model. The model can have multiple inputs but the following methods can be applied to each input individually.

- **Random Instrument Masking:** In the melodic accompaniment, generally a few of the melodic instrument takes a lead in melodic pattern while the other instruments try to complement this with chords, arpeggios, or some complementary melodic pattern. To force the model to consider such variations while making a decision, we randomly mask one of the melodic instruments in 40% of the input samples. The choice of instrument to be masked is being done at random. This is done in an online fashion in every epoch to increase the number of augmented samples.
- **Random Timestep Masking:** At times the musicians can make a mistake leading to disruption of the rhythm of the song. If such a thing happens, they are trained to get back on the timing grid as fast as possible. We try to simulate this by randomly masking 20% of the timesteps from the input.

3.3.2 Supplementary Data Augmentation Methods

In addition to the above augmentation methods, we use the following methods when necessary. The first method can be applied to models with more than one input, while the second method is used to replicate the behaviour of noisy predictions to be used by other models.

- **Input Masking:** For models with more than one input [41], it is important for the model to efficiently work with partial inputs in case some inputs are missing. This also enhances the individual branches of the input to extract more relevant features and forces the model to not to memorize any pattern in input for a particular output. This can be achieved by randomly dropping some inputs while training.

- **Input Random Noise:** As no training is perfect, during the evaluation phase there tends to be some noise in the predictions. If the output of one model is being used as input for another, this noise can disrupt the second model's behaviour. To tackle this, we can introduce noise in the training phase of the second model itself. This helps the model to filter out the noises and make a better prediction from the noisy input.

3.4 Fills and Improvisation Detection

Fills are a short group of notes played as the music transitions from one section to the next, which are typically 8-12 bars long. If a section is supposed to be longer, then a small fill may be introduced in between to keep the audience engaged. These act as an indicator to the audience as well as the band of the upcoming transition in the song. A drummer generally plays a basic repetitive drum pattern with occasional fills and improvisation in the drum pattern to keep the audience engaged. The basic drum pattern can exhibit a different behaviour near a fill as shown in Figure 3.5.

As the focus of this work is on detecting and generating fills and improvisations in the drum track, we need to isolate them from the full drum track. To achieve this, we propose a novelty function to capture the extent of improvisation in a bar compared to its neighbours using a self-distance metric. We perform the following steps (shown in Figure 3.6) to capture the locations of improvisations

1. For any 11 bar sample, we consider the centre bar as the bar of interest and use the 5 bars on either sides as the context bar. We calculate the between two bars: bar_i and bar_j using the following equation:

$$\frac{||bar_i - bar_j||_1 \times k}{||bar_i||_1 + ||bar_j||_1} \tag{3.1}$$

The value k is the hanning window weight. This is based on the relative distance between

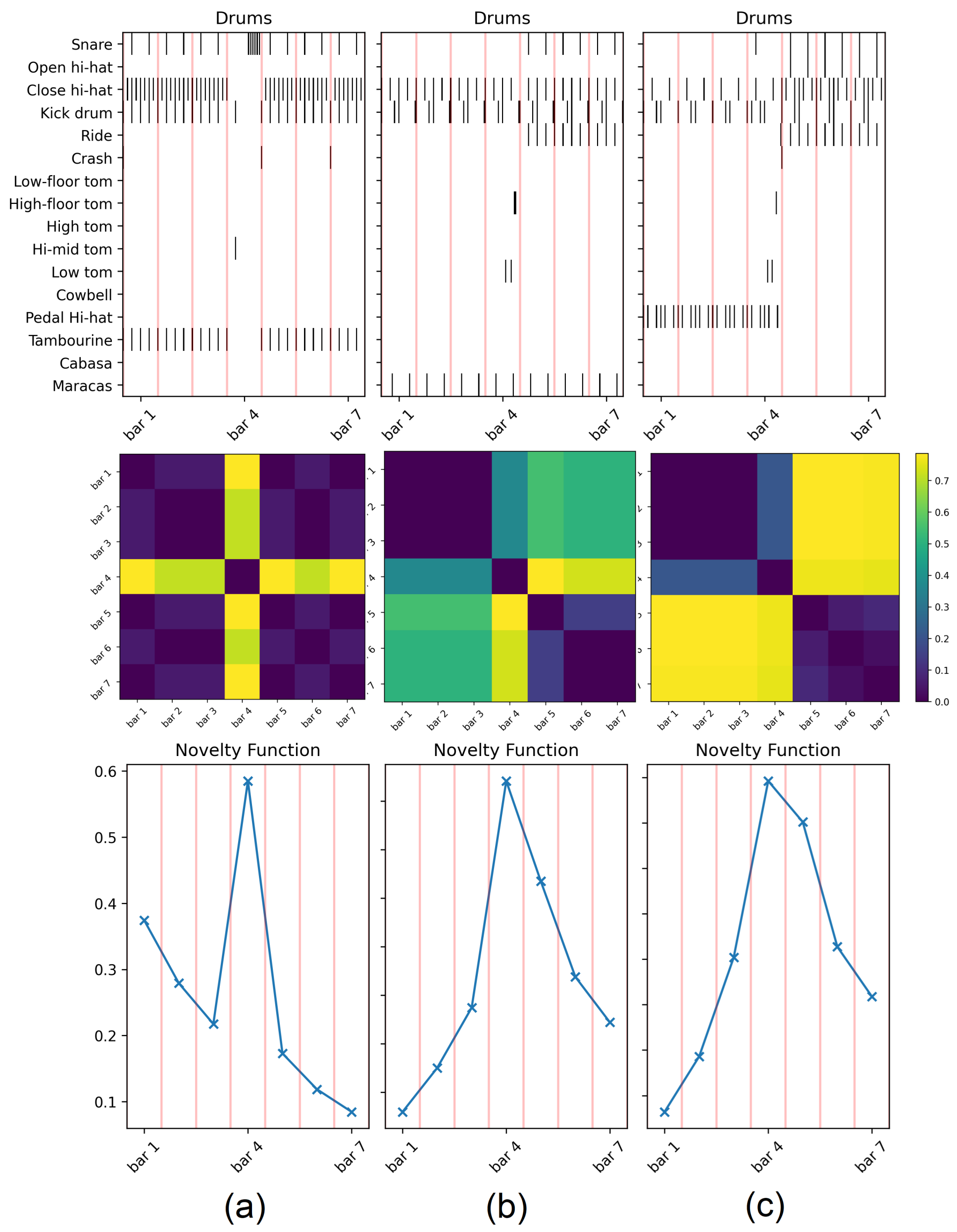


Figure 3.5: Different behavior of basic drum pattern under around fills. The SSM plots for the same and the novelty plot for all the 3 samples (a) The basic pattern remains the same before and after the fill (b) The basic pattern changes slightly after the fill (c) There is a huge change in the basic drum pattern after the fill

bars i and j. As the improvisation is a local phenomenon, we use this weighing parameter to decrease the effect of bars that are far from each other.

2. This calculation is done for all the bars across a track, except the first and the last 5 bars due to the lack of context bars. From 3.7 it can be seen that the novelty function peaks at the bar with a drum fill. These bars are then extracted using a peak picking mechanism.

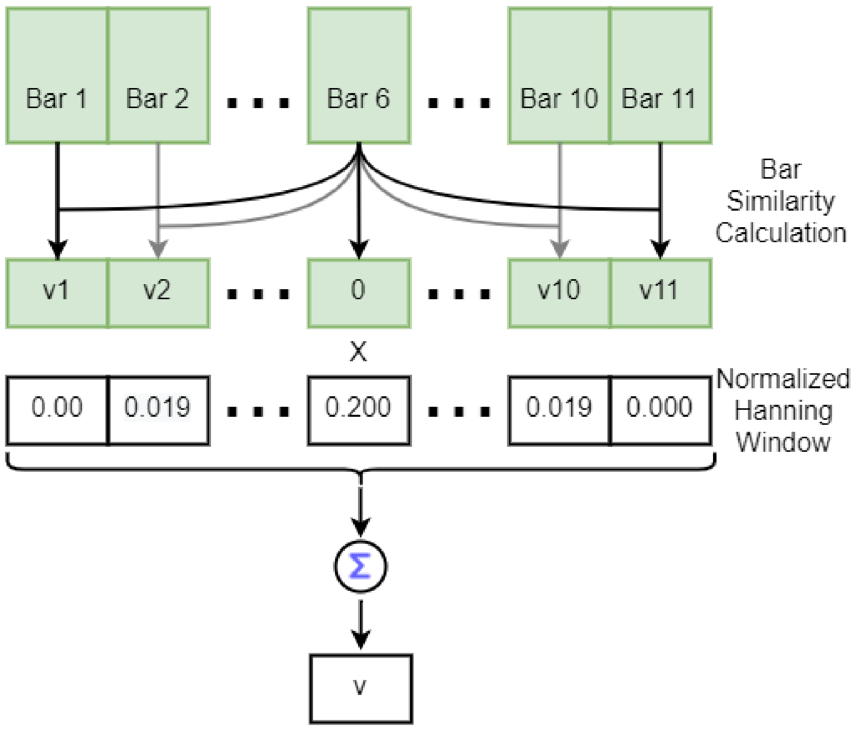


Figure 3.6: Steps for calculating the novelty value of a bar

3. To generate the dataset for our task, we pick the bars with a local maximum as the positive samples. To filter out minor deviations, e.g., bar 18 in 3.7, we put a threshold of 0.1 on the peak's height difference from its neighbours. A maximum of 10% of total bars in a song with these characteristics are selected as the positive samples to filter out weak ones. Same number of bars from the rest of the non-peak regions are selected as negative samples.

Using the procedures outlined above, we were able to identify 146432 bars in the dataset that contained improvisations out of a total of 1779712 bars, accounting for $\approx 8.2\%$ of the total bars.

3.5 Analysis

Figure 3.5 shows different ways in which the basic drum pattern can change after the fill. The basic drum pattern can mainly exhibit one of these 3 changes in its groove pattern:

1. Continue with the same groove after the fill as before. This is usually done in the middle

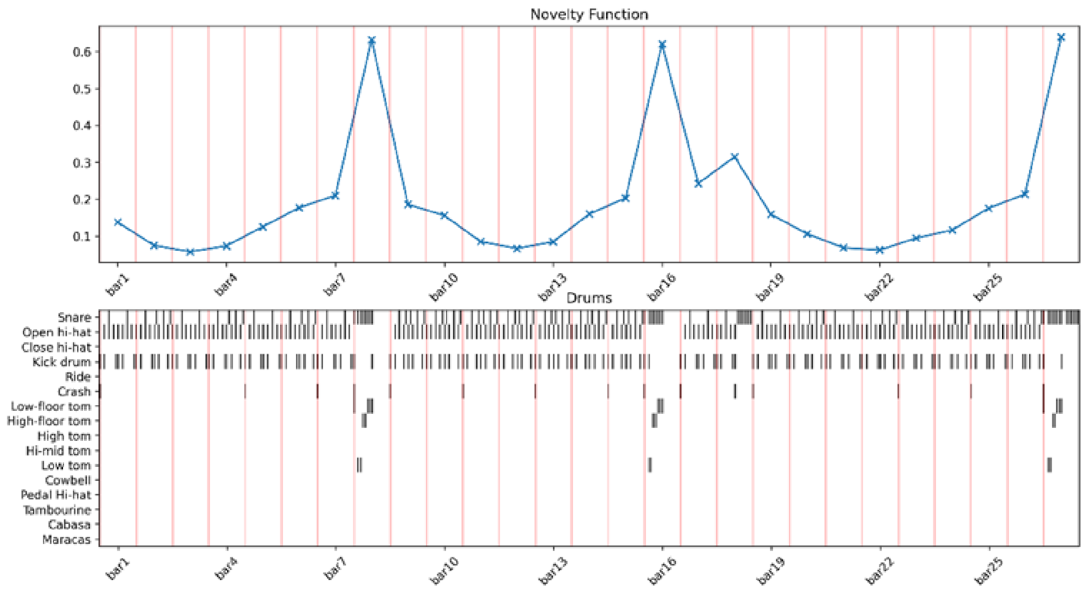


Figure 3.7: Novelty Function Plot for the given Drum Track

of long stanzas.

2. Slightly modify the previous groove pattern. This is usually done when the next stanza exhibits a similar tone as the previous one.

3. Completely modify the previous groove pattern. This is done when the following stanza has a huge tonal difference from the current stanza.

Using Equation 3.1, we can quantify the difference in the basic drum pattern. From the bars identified in the above step, we pick the basic drum pattern bars just before and just after the drum fill bar and calculate the distance between them. Figure 3.8 shows the distribution of these bar similarity values. They are divided into three equal regions corresponding to the three types of changes in the groove. This indicates that the majority of the time (61%) there is a very small change in the drum groove pattern. The groove present in Figure 3.5a, b, and c correspond to the first, second and third region respectively.

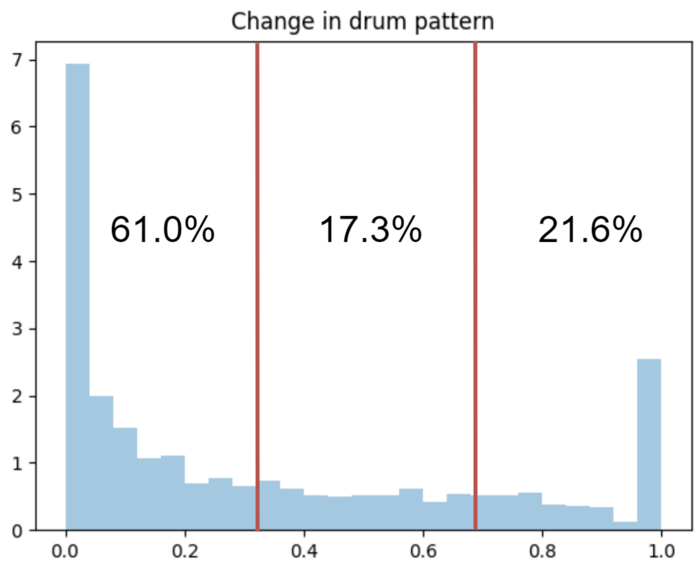
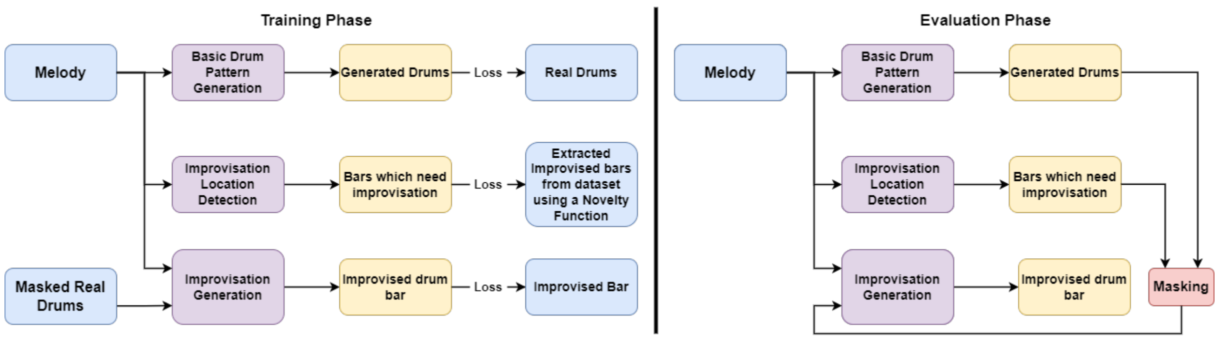


Figure 3.8: Distribution of bar similarity values between the bars just before and just after the bar of improvisation

Chapter 4

Proposed Generation Model

The overview of the proposed system is shown in Figure 4.1. We start by generating a basic drum pattern for the given melodic accompaniment input. This is done using an encoder-decoder based transformer model. After training this model, we find that the model was not able to capture the regions of improvisations and fills and mainly generated a basic repeating drum pattern. To counter this, we develop an improvisation location detection model and an improvisation generation model which takes in the melodic accompaniment and percussion accompaniment generated by the first model to locate and generate a fill or improvisation. We also develop a novelty function to capture the dissimilarity of a drum bar compared to its neighbours



Training Phase
Melody
Basic Drum Pattern Generation
Generated Drums
Loss
Real Drums
Improvisation Location Detection
Bars which need improvisation
Loss
Extracted Improvised bars from dataset using a Novelty Function
Masked Real Drums
Improvisation Generation
Improvised drum bar
Loss
Improvised Bar
Evaluation Phase
Melody
Basic Drum Pattern Generation
Generated Drums
Improvisation Location Detection
Bars which need improvisation
Improvisation Generation
Improvised drum bar
Masking

Figure 4.1: Overview of the proposed system

and use it to locate the improvised bars. Details of each of the models are provided in subsequent sections. Our models are trained for 300 epochs with Adam optimization [42] starting with a learning rate of 1e-4 and decaying it till 1e-6. All the setup was carried out using the Tensorflow [43] framework.

4.1 Common Modules

Before we dive deep into the model architectures, there are a few common modules used by all the models. These modules are being discussed:

4.1.1 Embedding Module

Inputs to our models are either a pianoroll matrix or a series of tokens (serial grid format). Pianoroll representation is a highly sparse representation with only a few MIDI pitches/instruments active at any given timestep. To reduce this sparsity and learn the inter-instrument and inter-note dependencies we use a 2-layer ReLU activated [44] MLP as our embedding module for the pianoroll. For the serial grid input, we use a token embedding layer to learn the token dependencies.

4.1.2 Position Encoder

In the self-attention mechanism of a Transformer, there is no notion of the ordering of input tokens. As all of our models use a Transformer encoder, we need a position encoder module to embed the positional information of different input vectors. We concatenate the sinusoidal positional representations [26] with the input vectors and use a dense layer to project them back into the original dimension space to keep the same input dimension.

4.2 Basic Drum Pattern Generation

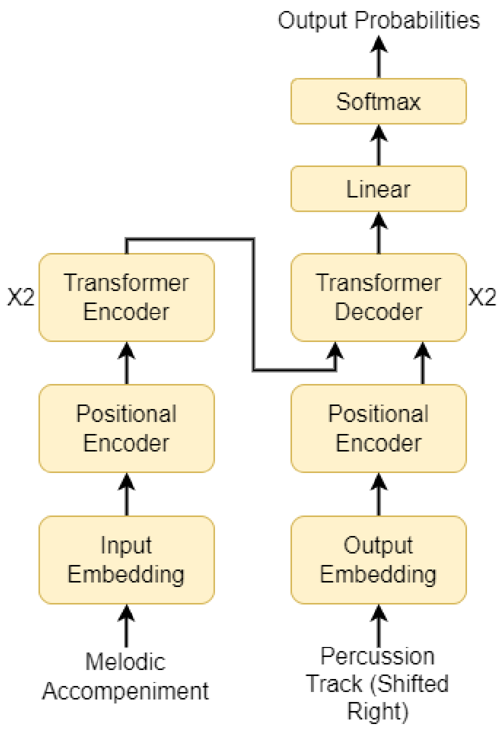


Figure 4.2: Basic Drum Pattern Generation Model

Basic drum pattern refers to a drum pattern without the fills and improvisations. We train a Transformer sequence to sequence model [26] to try to replicate the drum pattern (including the fills and improvisations) (Figure 4.2). For this task, we use the serial grid representation of percussion track. This converts our task to a sequence translation task [45, 46] which allows us to explore the existing literature on it. Multi-state decoding of the output would be required in the case of a pianoroll representation of a percussion recording, which is a separate research area.

For our task, we give the melodic accompaniment as the input to the encoder and the shifted percussion accompaniment tokens to the decoder branch. Both the inputs are first passed through the embedding module (1024 and 128-dimension dense layers) followed by the position encoder. The embedded inputs are passed through 2 layers of encoder/decoder module with 128-dimensional latent space and 8 attention heads. At the output, we have 18 neurons corresponding to the 16 drum instruments, silence token, and shift token. The output decoding can be done in various ways: greedy decoding, simple sampling, temperature sampling, nucleus sampling, and top-k sampling [34]. In our work, we test greedy sampling and simple sampling. With the greedy decoding method, we select the token with the maximum probability at each step of decoding, while with the simple sampling method, we select the tokens from the distribution predicted by the model.

When we inspect the drum samples generated by this model, we find that the model can generate a good repeating drum pattern that maintains the timing structure of the music, but fails to detect and generate improvisations. Also, the model maintains a highly similar drum pattern throughout the 11 bars with very small deviations due to output decoding strategies.

The above model is evaluated with the negative log-likelihood (NLL) values over the train and the validation set. As the model outputs a distribution over the 18 output tokens, there are multiple ways to decode it. We test the greedy method of decoding, where the token with the maximum probability is selected at every step, and the simple sampling method. The model is trained using categorical cross-entropy loss and achieved an NLL of 0.108 and 0.112 on the train and validation split, respectively.

4.3 Improvisation Location Detection

Wei et al. [9], Lettner et al. [47] and Paulus et al. [48] noted that the self-similarity matrix (SSM) of the melody is structurally similar to the SSM of the drums. They used the melody SSM to generate drum SSM which was further used to generate the drum bars. To build on this hypothesis, if just given with melody, we should be able to predict the locations of improvisations and possibly even generate them. We train the following 4 models to test whether we can detect the location of improvisations from melody only:

4.3.1 Model 1: Simple SSM Classifier

This model, is based on the idea that a simple melody SSM should be enough to detect the locations of improvisations. The input to this model is the SSM matrix of the melody pianoroll. The SSM is calculated at bar level making it a 11 (bars) $\times$ 11 (bars) matrix. This is then

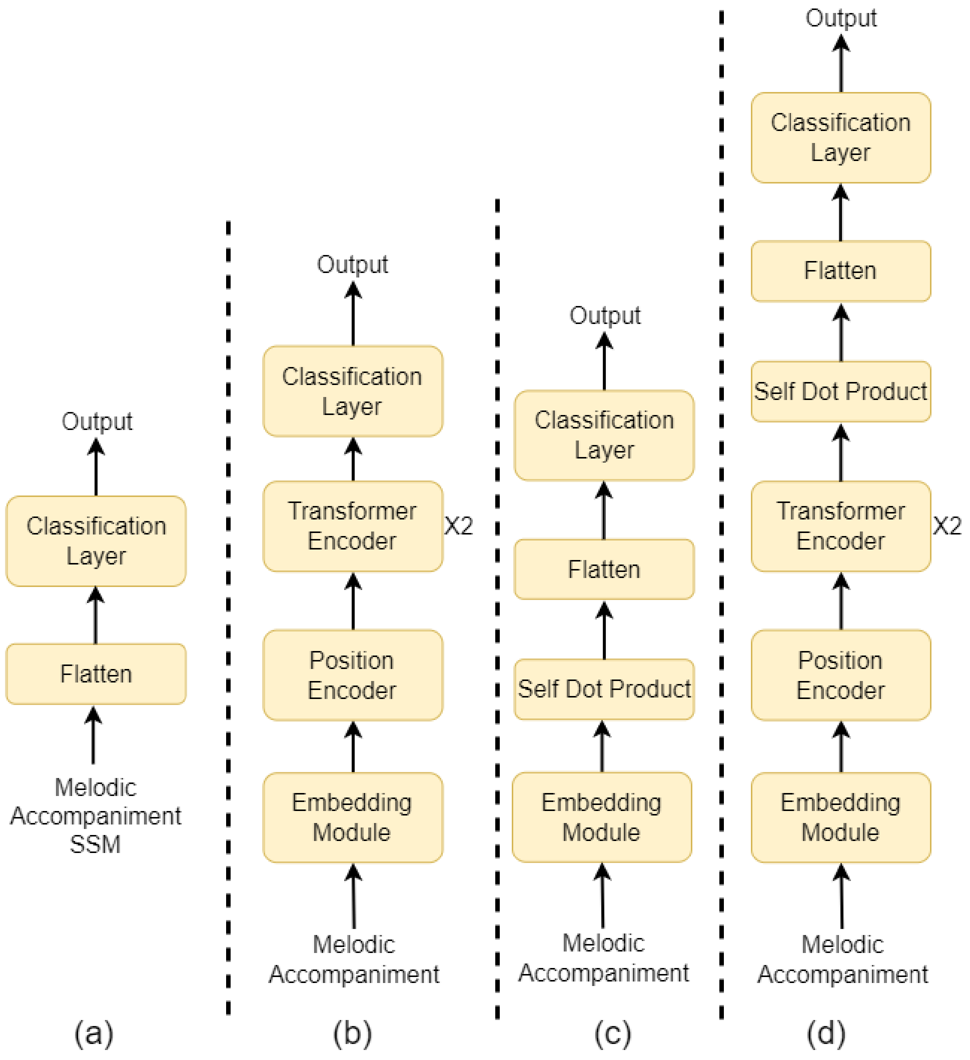


Figure 4.3: (a) Melody SSM Classification Model Architecture; (b) Melody Feature Classifier; (c); (d) MLP Feature Extraction Model (d) Transformer Encoder Feature Extraction Model

flattened and passed through 3 layers MLP with 256-128-2 neurons (Figure 4.3a). The outputs are softmax activated to represent the positive and negative classes. The model is trained on categorical cross-entropy loss using an Adam optimizer for 60 epochs. As this is a classification task, the accuracy, precision, and recall are measured. The results are presented in Table 4.1.

4.3.2 Model 2: Melody Feature Classifier

In this model we extract features from raw melody pianoroll using a Transformer Encoder and pass these features through classification layers to predict the location of improvisations. The self-attention mechanism of the transformers have been shown to extract good high level features from raw data of various modalities [49, 50]. Hence a sub beat level the feature extraction is being done using the transformer encoder blocks with 12 attention heads, 64 dimensions embedding and 512-dimension feedforward layer. The classification layers are 2 layer MLP with 1024-2 dense layers. The output is again softmax activated to represent the positive and negative classes. The model is trained on Huber loss [51] as it's robust to outliers and less sensitive to noise to 300 epochs.

4.3.3 Melody SSM Feature Classifier

On the lines of observation made by [9, 47, 48], if the melody SSM captures the location of improvisations, then it should be able to make the predictions. But from results of Model 1 (Table 4.1), it can be seen that an SSM calculation from raw melody does not help. We test 3 different methods of feature extraction from raw melody for SSM calculation.

1. **Simple MLP Feature Extraction (Model 3a):** We test a simple 3 layer MLP module to extract features from the bar level representation of raw melody (Figure 4.3c). To get a bar level representation, we simply concatenate the vectors of all the timesteps in a bar. This results in a 11×8192 shaped input matrices. The first 2 layers (512-128 dimensions) of this MLP are `ReLU` activated while the third layer (128 dimensions) is `tanh` activated. A dot product of these features is then taken to generate the SSM structure. This is then flattened and passed through the classification layer similar to the ones used in Model 1 4.3.1. Results of this model can be found in table 4.1.

2. **Bar Level Transformer Encoder (Model 3b):** Instead of a simple MLP structure to extract features from melody, we use 2 layers of transformer encoders for this task. The melody representation is first changed to bar level representation as discussed above. This bar level representation is sent to an embedding module with 1024-128 dimensions, followed by a positional encoder block. These position encoded vectors are sent to 2 layers of transformer encoders with 8 attention heads, 128 embedding dimension and 256 feedforward dimensions. The features extracted from these blocks are then passed to another `tanh` activated dense layer with 128 dimension. A dot product of the extracted

features is taken to get an SSM structure which is flattened and passed to the classification module used in Model 1 4.3.1. Results of this model can be found in table 4.1.

Model	**Precision**	**Recall**	**Accuracy**
Model 1	50.0	50.0	50.0
Model 2	79.1	79.4	79.3
Model 3a	50.0	50.0	50.0
Model 3b	82.1	82.1	82.1

Table 4.1: Results of different models for the improvisation location detection task

From the Table 4.1 it can be seen that Model 3b i.e., the Bar level transformer encoder performs the best closely followed by the model 2. Other models do not show any performance improvements compared to a random guessing model. While these two models are based on 2 different intuitions for prediction, they both have a very close performance. This also shows that melody can be used to predict the locations of improvisation in the drum track.

4.4 Improvised Bar Generation

The final step in our system is the generation of improvised bars. To achieve this we use the architecture shown in 4.4. We provide 11 bar melodic accompaniment and percussion accompaniment as the input to the model. During the training phase, the percussion accompaniment is the original percussion track, whereas, during the evaluation/generation phase, the percussion track generated by our first stage model (section 4.2) is used. As these generated drum samples are prone to errors, we simulate this by adding random noise to the input while training 3.3.2. The 6th bar in the percussion sample (middle bar) is the target bar and is masked while giving as the input. To increase the robustness of this model, we use the supplementary data augmentation methods 3.3.2. As there are two different input branches, we mask out one of the inputs randomly 20% of the time.

We generate a summary vector of both the melodic accompaniment and the percussion accompaniment inputs. Both are first passed through an embedding layer followed by a position encoder module. This is then passed through 2 layers of transformer encoders with 128

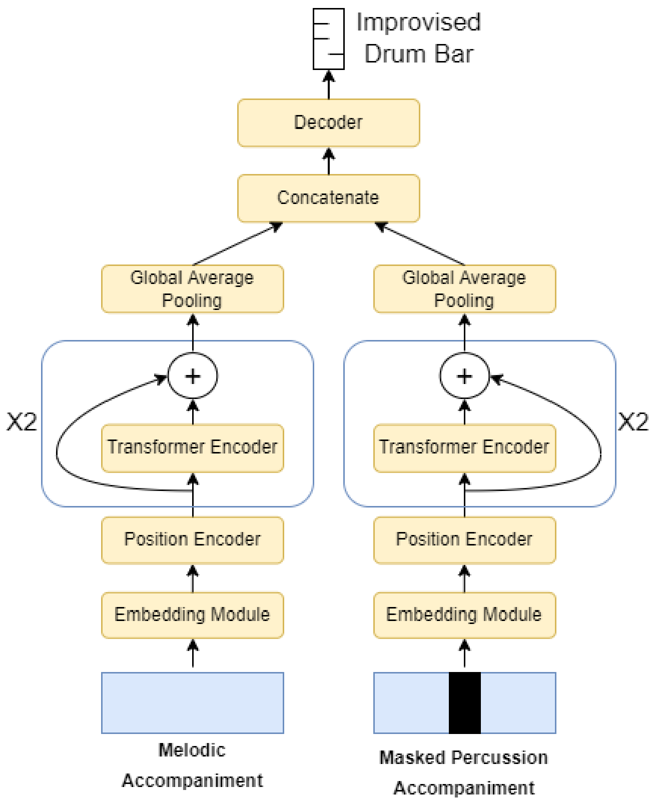


Figure 4.4: Improvised Bar Generation Model Architecture

dimensional latent space and 8 attention heads. We add skip connections to ease the flow of gradients [52]. To generate the summary vector for each input, we use a global averaging technique. These two vectors are concatenated and passed through a decoder structure which looks at the concatenated vector to generate the improvised drum bar. We test the following decoder architectures with our model:

1. **MLP:** A 3-layer dense network with 2048-2048-512 neurons is used. The final layer is sigmoid activated. The outputs are reshaped to 32 (timesteps) $\times$ 16 (percussion instruments) to get the improvised bar.

2. **MLP mixer:** MLP mixers [53] are simple alternatives to convolution and self-attention. They are based on multi-layered perceptrons applied across either temporal dimension or feature dimension. It mixes tokens, or in other words, facilitates communication across different patches in the same channel.

3. **Conv1d:** A simple conv1d architecture with blocks of 2 layers of conv1d followed by upsampling.

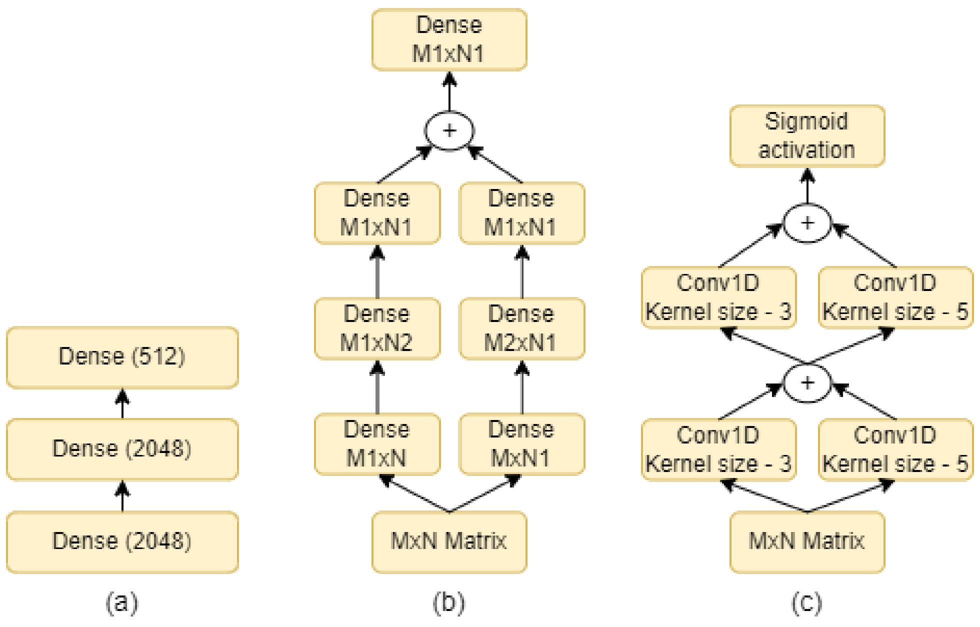


Figure 4.5: (a) MLP decoder architecture (b) MLP Mixer decoder architecture (c) Conv1D architecture

We treat the prediction of the improvised bars as a regression problem. We train it with Huber loss as it is less sensitive to outliers. We do not use cross-entropy loss for generation firstly purely as a design choice, and intuitively each of the time step tokens in the prediction within a bar lack probabilistic interpretation. We monitor and report the accuracy, precision, and recall of the models. As the distribution of 0s and 1s is not uniform in the predicted sample, F1 score provides a better insight in the performance of the models. From Table 4.2 it can be seen that a simple 3 layered MLP decoder can to perform better than the complex MLP mixer and Conv1D architecture.

	Precision	**Recall**	**Accuracy**	**F1 Score**
MLP	82.9	70.3	79.0	76.0
MLP Mixer	83.5	42.1	86.0	56.0
Conv 1D	53.1	70.3	86.1	60.5

Table 4.2: Results of various decoder architectures used in the improvisation generation task

Given a simple drum pattern played by a real drummer, we utilise the following criteria to determine how much musically significant fills/improvisations we can generate (calculated on the test set):

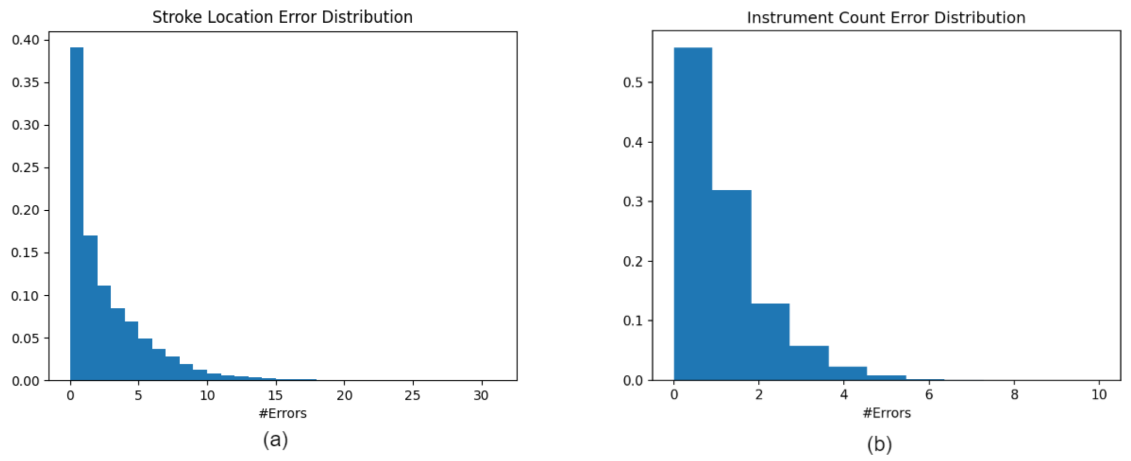


Figure 4.6: (a) Stroke Location Error Distribution (b) Instrument Count Error Distribution

- **Stroke Locations Error:** This checks the number of locations where any instrument onset should have occurred according to a real drum track but were not present in the predicted bar. From Figure 4.6a it can be seen that while a majority of them have zero error, there is still room for improvement.
- **Instrument Count Error:** This checks if our model was able to understand the inter-instrument dependencies and replicate them. From Figure 4.6b it can be seen that about $\approx 80\%$ of the bars have less than 1 instrument difference. This implies that our model was able to understand and replicate the dependencies of various instruments.

Chapter 5

Evaluation of Generated Samples

To evaluate the quality of the generated PA pattern of the proposed system, we conduct both objective and subjective tests with:

1. **O:** Original MIDI drum patterns that are present in the MIDI files in the dataset.
2. **P1:** The basic drum pattern generated by our Basic Drum Pattern Generation model (section 4.2).
3. **P2:** The final drum patterns with fills and improvisations generated by the complete system.

5.1 Basic Filtration of Output

As the output of the Basic Drum Pattern Generation model is decoded using a sampling-based method, it is prone to errors. Sampling a wrong output at any step can lead to degeneration. To reject such samples, we apply the following filters:

1. Getting a complete silence after a couple of drum beats was the most common observed output degeneration pattern. To filter out such samples, we check for the total number of silence bars in the generated outputs and reject all the generated patterns with more than 4 silence bars.

2. While sampling from a distribution, states with lower probability can also be sampled. If an unexpected state is decoded at any given step, it will lead to a deviation of the drum density of future steps too. To reject such samples, we calculate the density of each bar in the 11-bar drum sample and put a cap of 5 on its standard deviation.

After applying the aforesaid filtration to 8192 P1 drum samples generated by simple sampling method, we are left with 3762 samples for further evaluation.

5.2 Objective Evaluation

Evaluation of generative music systems faces harder challenges than that of image generation systems (Briot et al. [54]). To evaluate our models, we design several metrics that can be computed for both the real and the generated data. We compare the basic drum patterns generated by the sequence to sequence Transformer model with the ground truth patterns for rhythmic consistency and to evaluate how good the patterns were musically. We also do a separate evaluation of the generated fill bars to get a better understanding of the system. Finally, we request a few experts to give their insights and comments on the generated drum samples.

5.2.1 Evaluation of Basic Drum Pattern

5.2.1.1 Onset Distribution in a Bar

A basic requirement that any drum pattern generation system must fulfill is the rhythmic positioning of onsets. Generally, in a bar of rock music, the first beat (downbeat) and the third beat have similar instrumentation, featuring a hi-hat and kick drum onset. Beats 2 and 4, commonly referred to as the backbeat, include a hi-hat and snare drum onset. While the quarter notes are accented across the bar, 8th notes are accented within the beat interval. We check the distribution on onsets in a bar to understand if the model was able to learn the importance of this grid structure. From Figure 5.1, we can observe that the model was able to emphasis the start and mid of all the beats, which mainly correspond to the locations of the 8th note. This is particularly interesting because we didn't provide any explicit information regarding bar/beat boundary but the model was still able to extract it from the raw data.

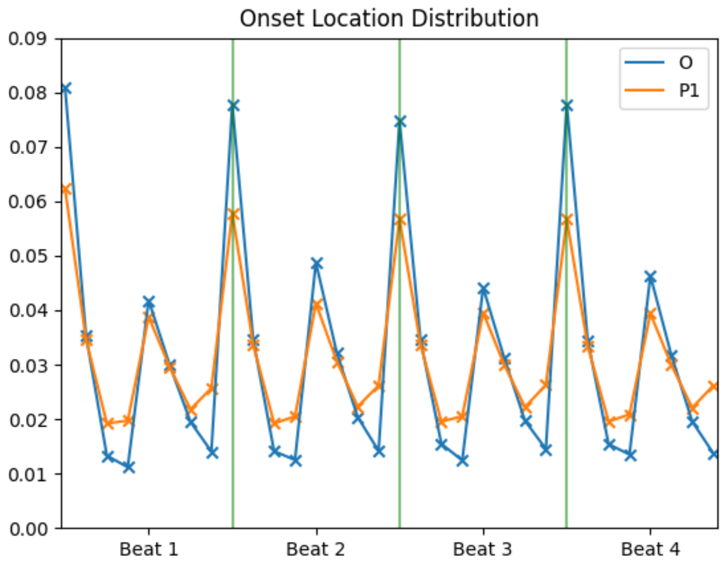


Figure 5.1: Distribution of onset position in a bar in O and P1. Accents at the start and middle of a beat imply an 8th note pattern

5.2.1.2 Pattern Matching

We perform a one-to-one comparison of the P1 outputs against their target drum pattern to understand how closely are the patterns matched with the original drum sample using the following metrics:

1. **Instrument Count (IC) Error:** Instrument count is defined as the total number of distinct instruments used in a bar. To get a better understanding of whether our model can replicate the behavior of multi-instrument dependency, we calculate the error of IC in the generated sample with that present in the real drum track for the same MA. We observe that in about 75.8% of the drum bars, we can replicate the IC, while in 99.3% of the bars our model was off by at most 1 instrument.

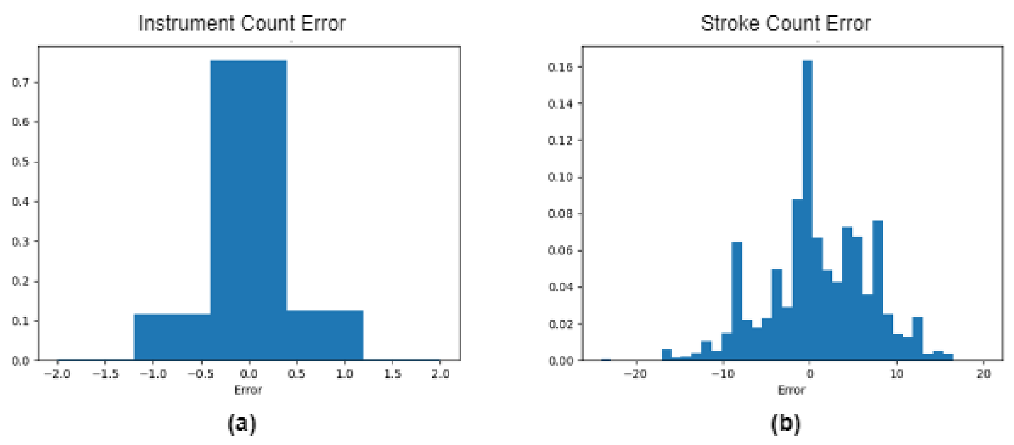


Figure 5.2: (a) Distribution of Instrument Count Error and (b) Distribution of Stroke Count Error in O and P1 samples

2. **Stroke Count (SC) Error:** Stroke count is defined as the total number of timesteps with the onset of any percussion instruments. To understand how well our model was able to replicate the net strokes in a bar, we calculate the error of SC in the generated sample with that present in the real drum track for the same MA. Figure S2 shows the distribution of this error. We can see that while most of the time the model was able to generate a similar stroke count as the ones present in the dataset, some bars had a significantly larger error. The model compensated for an increase in SC for one bar by slightly lowering the SC at some other bar, on average.

5.2.1.3 Rhythmic Consistency

Another important aspect that needs to be considered while generating drum patterns is to have a rhythmic consistency. This means that the generated drum bars should have a consistent drum pattern and there needs to be a consistency in the onset locations across the bars. We evaluate these aspects of the generated drum pattern using the following metrics:

1. **Inter-onset Interval (IOI):** Inter-onset interval in the symbolic domain is the number of timesteps between two consecutive onsets. We measure the IOI on the dataset and the generated samples and compute its distribution and find that the overlapping area of the two is 93.2%.

   Pattern Consistency: For consecutive bar pairs, we calculate the distance between the

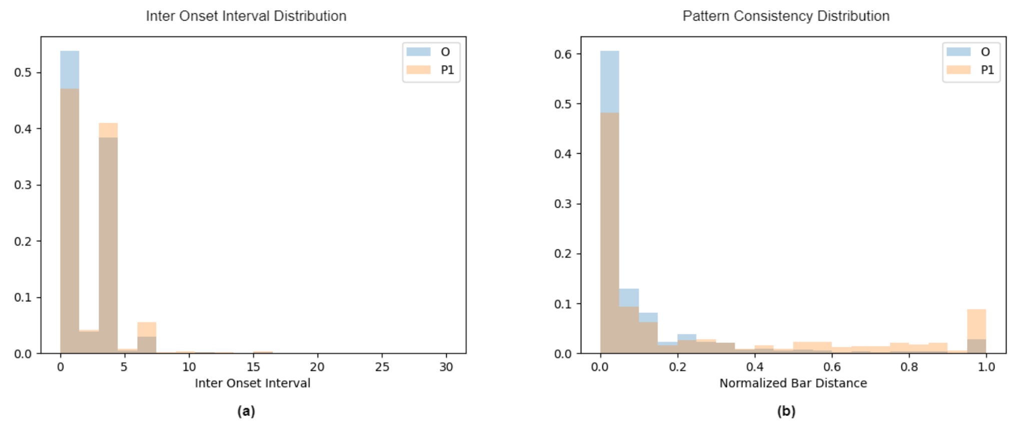


Figure 5.3: (a) Distribution of Inter Onset Interval and (b) Distribution of pattern consistency across the O and P1 drum samples

drum patterns using the equation 3.1. The distribution of the bar distances is shown in Figure 5.4b. We can see that the generated drum bars are more or less similar to each other with some minor deviations due to the sampling decoding method. The overlapping area of the two distributions is 80.4%.

5.2.2 Evaluation of Bars with Fills/Improvisations

To understand how well were our models able to capture the fills/improvisations, we use the following evaluation methods on the improvised O and P2 bars:

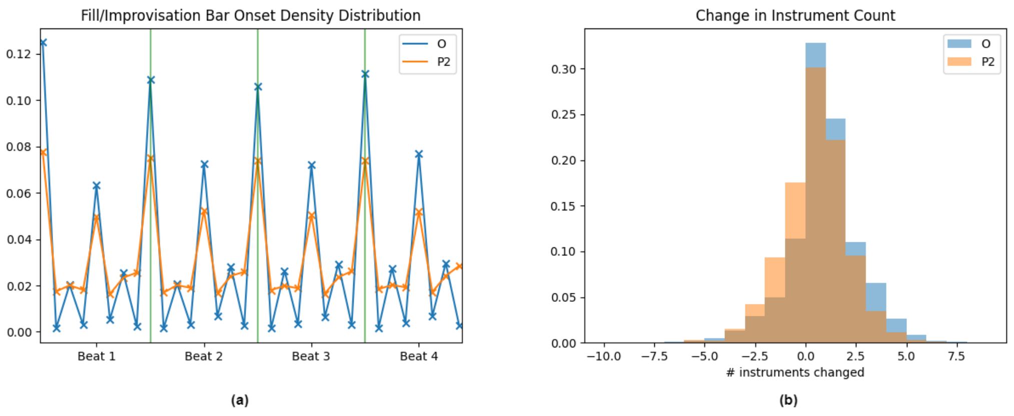


Figure 5.4: (a) Onset position distribution for fill/improvised bars only in O and P2 samples (b) Distribution of change in Instrument Count with respect to the previous bar

1. **Onset Position:** Figure S5 shows the distribution of onset location of percussion instruments across the improvised bar. We observe a slightly higher proportion of 16th note patterns in the improvised O bars as compared to onset distribution across the non-improvised bars seen in Figure S1. We can see that P2 is largely able to capture this behavior as well.

2. **Instrument Count (IC) Change:** Generally during a fill/improvisation, an additional instrument is introduced. IC change is measured as the change in IC of the improvised bar compared to the previous bar. Figure S6 shows the distribution of IC change. The overlapping area of the two distributions is 87.9%, This demonstrates that our model was able to capture the general trend of IC change.

5.3 Subjective Evaluation

To evaluate the perceptual quality of the generated outputs, we present the generated samples to trained musicians. We select 7 MA tracks and their corresponding PA from the 3762 filtered samples. For each of the MA tracks, we have an O, P1, and P2 drum sample. We make 3 pairs i.e. O & P1; P1 & P2; O & P2 for each MA-PA track and present these 21 pairs to two guitarists and a multi-instrumentalist with experience ranging from 5 to 10 years. They were asked to provide detailed comments on the drum pattern in terms of timing, appropriateness of fills, and coherence of the PA with MA. During the comparison, they were asked not to focus on the audio quality or loudness.

In terms of coherence of melody and the generated drum track, while the majority of the time O drum track was preferred over P1, P2 was able to gain significantly more votes over O as compared to P1 over O. The basic drum pattern with no fills/improvisation in P1 gave it a monotonous feel and it was one of the major reason for selecting O over P1. While comparing P1 and P2, in a few samples it was difficult to choose a better version due to the difference of just one bar. Most of the time, P2 was selected as the fills imparted a lively feel to the accompanying drums. However, in a few cases, it was also reported that the fill introduced by P2 felt a bit unnecessary.

5.4 Sample Results

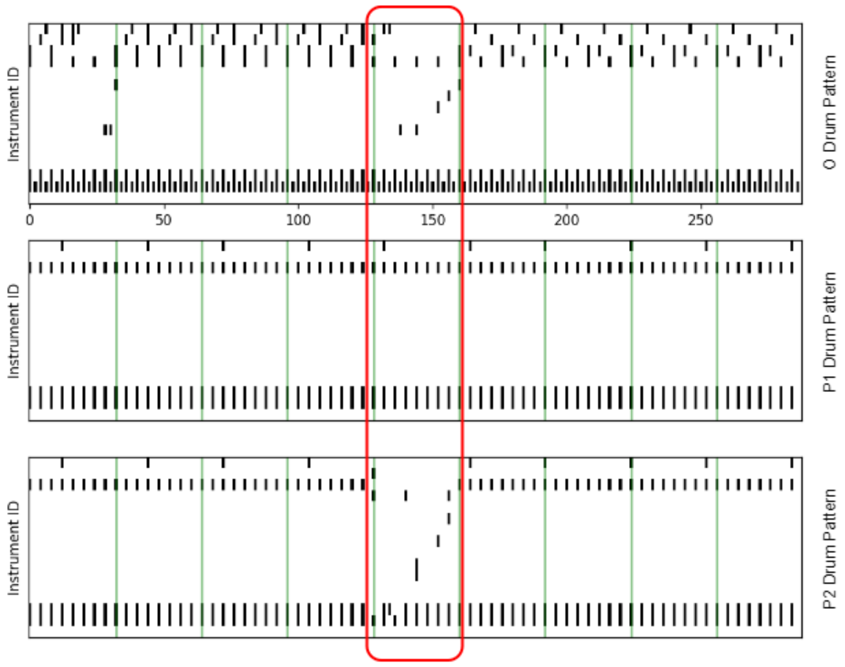


Figure 5.5: Example 1: Pianoroll of O, P1, and P2 drum pattern for the same Melodic Accompaniment

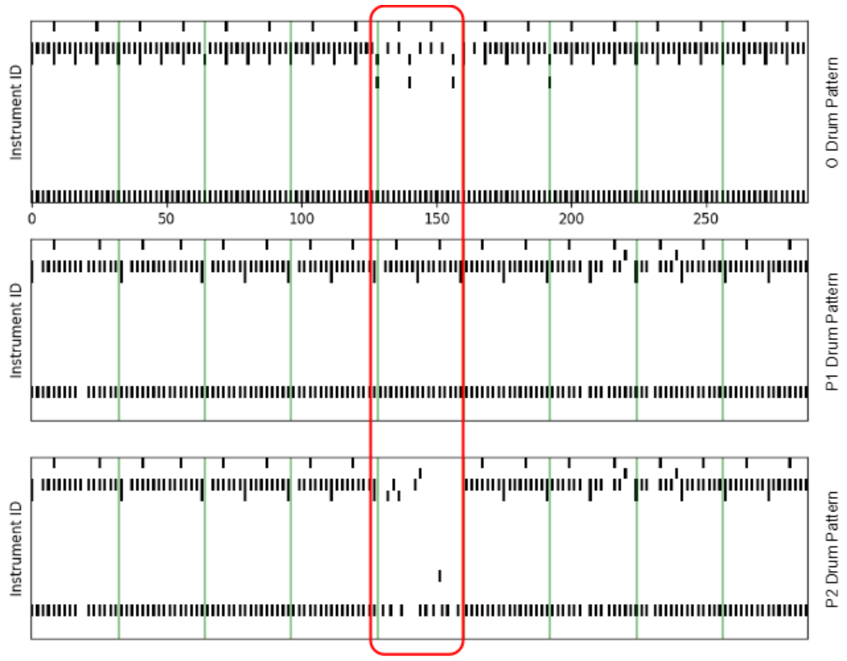


Figure 5.6: Example 2: Pianoroll of O, P1, and P2 drum pattern for the same Melodic Accompaniment

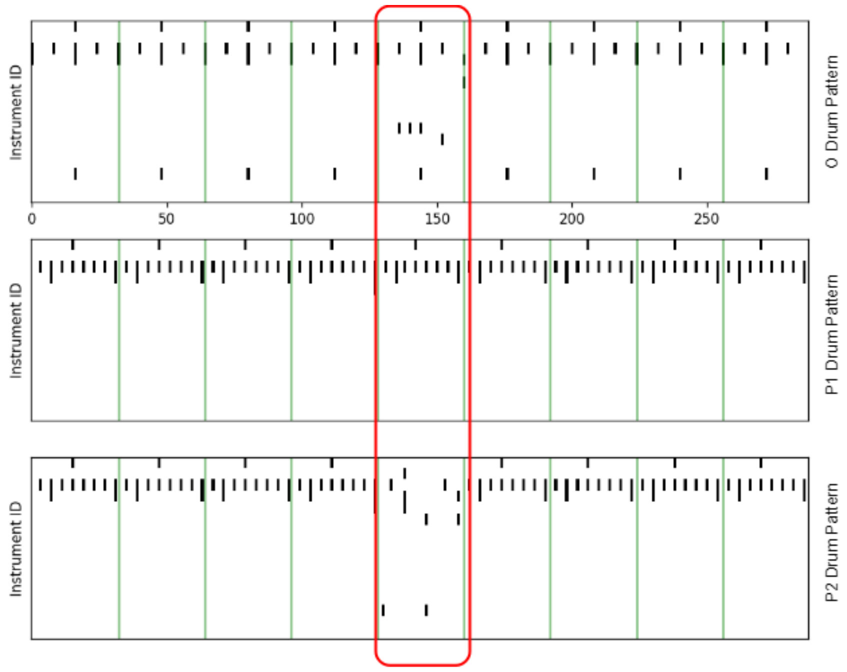


Figure 5.7: Example 3: Pianoroll of O, P1, and P2 drum pattern for the same Melodic Accompaniment

Chapter 6

Summary and Future Works

We have successfully shown a method to produce coherent drums accompaniment with improvised bars by conditioning on a given melodic accompaniment. A novel BERT-inspired infilling architecture is proposed, along with a self-supervised improvisation locator. By learning, where and how to improvise, our evaluations indicate improved generation quality. Thus with a two-step approach, we mitigate the biases and shortcomings that exist with current machine learning architectures. The system can further be improved by learning optimal sampling techniques, which remains an open problem. The findings of our work can yield advances in other domains such as natural language processing and computer vision due to the ubiquity of these architectures. Finally, extending these beyond midi-like events and solving this at scale remains an exciting and challenging problem ahead of us. This work highlights a serious drawback of traditional language-based generators, which have shown promise in a lot of different fields, yet they fail to capture subtle musical signals, where they are often sparsely occurring in otherwise repetitive and common patterns. In the future this work can be extended in the following directions:

- **Increasing Context:** The first and most important extension of this work would be of increasing the context size of melodic and percussion accompaniment tracks.

- **Change in Basic Drum Pattern:** In this work, we have focused on generating fills when the basic drum pattern does not change before and after the fill. From section 3.5, we know that while happens a majority of the time, there is still a large proportion where there is a change in the basic drum pattern. Due to the current step-by-step generation process, we can tackle only the fills where there are no changes in the basic drum patterns.

- **Improving Individual Modules:** We test multiple model architectures for each individual module used in the work. Yet, there are thousands of untested architecture which could give a better performance and bridge the gap in the model's learning.

- **End-to-End Trainable System:** Our current system has 3 different modules, each with a specific purpose. Each of the modules is trained individually by the signals extracted from the dataset. Having a system that can be trained end-to-end to mitigate these biases in the dataset would be of tremendous help even for the NLP tasks.

- **Output sampling strategies:** Decoding from the output distribution is an open challenge even for NLP tasks. While sampling methods have shown promising results, they tend to degenerate after a certain duration and Likelihood maximizing decoding methods cause repetitions and overly generic results [34].

- **Coherence Evaluation:** As any generative model is capable of producing thousands of samples for a given input, it would be interesting to be able to rank them for the user to select from. This is a challenging task as the model needs to be able to understand deeper structures of music and match them. This can be extended to other domains like question answering, text summarization, etc.

- **Loss function formulation:** The loss function plays a vital role in the learning of a model. Having a good loss function formulation that can help eliminate the biases in the dataset is a challenging task.

- **Drum Velocity Representation:** Velocity is the representation of the loudness of the beat. It plays a crucial role in expressing feelings during a song. Our current approach represents only the ON/OFF state of the drum instruments.

References

[1] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer. *arXiv preprint arXiv:1809.04281*, 2018.

[2] Yu-Siang Huang and Yi-Hsuan Yang. Pop music transformer: Beat-based modeling and generation of expressive pop piano compositions. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 1180–1188, 2020.

[3] Yi Ren, Jinzheng He, Xu Tan, Tao Qin, Zhou Zhao, and Tie-Yan Liu. Popmag: Pop music accompaniment generation. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 1198–1206, 2020.

[4] The role of the drummer in modern music. `https://marshall.com/live-for-music/arming-the-artists/the-role-of-the-drummer-in-modern-music`.

[5] Guangyu Xia and Roger B Dannenberg. Duet interaction: learning musicianship for automatic accompaniment. In *NIME*, pages 259–264, 2015.

[6] Guangyu Xia and Roger B Dannenberg. Improvised duet interaction: learning improvisation techniques for automatic accompaniment. In *NIME*, pages 110–114, 2017.

[7] Nan Jiang, Sheng Jin, Zhiyao Duan, and Changshui Zhang. Rl-duet: Online music accompaniment generation using deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 710–718, 2020.

[8] Stefan Lattner and Maarten Grachten. High-level control of drum track generation using learned patterns of rhythmic interaction. In *2019 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 35–39. IEEE, 2019.

[9] I-Chieh Wei, Chih-Wei Wu, and Li Su. Generating structured drum pattern using variational autoencoder and self-similarity matrix. In *ISMIR*, pages 847–854, 2019.

[10] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[11] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Mohammad Norouzi, Douglas Eck, and Karen Simonyan. Neural audio synthesis of musical notes with wavenet autoencoders. In *International Conference on Machine Learning*, pages 1068–1077. PMLR, 2017.

[12] Peter M Todd. A connectionist approach to algorithmic composition. *Computer Music Journal*, 13(4):27–43, 1989.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[14] Douglas Eck and Juergen Schmidhuber. Finding temporal structure in music: Blues improvisation with lstm recurrent networks. In *Proceedings of the 12th IEEE workshop on neural networks for signal processing*, pages 747–756. IEEE, 2002.

[15] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*, 2012.

[16] Elliot Waite et al. Generating long-term structure in songs and stories. *Web blog post. Magenta*, 15(4), 2016.

[17] Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. Midinet: A convolutional generative adversarial network for symbolic-domain music generation. *arXiv preprint arXiv:1703.10847*, 2017.

[18] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[19] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music. In *International conference on machine learning*, pages 4364–4373. PMLR, 2018.

[20] Chris Donahue, Huanru Henry Mao, Yiting Ethan Li, Garrison W Cottrell, and Julian McAuley. Lakhnes: Improving multi-instrumental music generation with cross-domain pre-training. *arXiv preprint arXiv:1907.04868*, 2019.

[21] Oscar Thörn. Ai drummer-using learning to enhancearti cial drummer creativity, 2020.

[22] Thomas Nuttall, Behzad Haki, and Sergi Jorda. Transformer neural networks for automated rhythm generation. 2021.

[23] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[24] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2020.

[25] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[27] Prateek Verma and Chris Chafe. A generative model for raw audio using transformer architectures. *arXiv preprint arXiv:2106.16036*, 2021.

[28] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[29] Prateek Verma and Jonathan Berger. Audio transformers: Transformer architectures for large scale audio understanding. adieu convolutions. *arXiv preprint arXiv:2105.00335*, 2021.

[30] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[31] Timo Schick and Hinrich Schütze. Rare words: A major problem for contextualized embeddings and how to fix it by attentive mimicking. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8766–8774, 2020.

[32] Emily Sheng, Kai-Wei Chang, Premkumar Natarajan, and Nanyun Peng. The woman worked as a babysitter: On biases in language generation. *arXiv preprint arXiv:1909.01326*, 2019.

[33] Kaitlyn Zhou, Kawin Ethayarajh, and Dan Jurafsky. Richer countries and richer representations. *arXiv preprint arXiv:2205.05093*, 2022.

[34] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

[35] Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*, 2018.

[36] Aylin Caliskan, Joanna J Bryson, and Arvind Narayanan. Semantics derived automatically from language corpora contain human-like biases. *Science*, 356(6334):183–186, 2017.

[37] Po-Sen Huang, Huan Zhang, Ray Jiang, Robert Stanforth, Johannes Welbl, Jack Rae, Vishal Maini, Dani Yogatama, and Pushmeet Kohli. Reducing sentiment bias in language models via counterfactual evaluation. *arXiv preprint arXiv:1911.03064*, 2019.

[38] Lakh pianoroll dataset. `https://salu133445.github.io/lakh-pianoroll-dataset/dataset.html`.

[39] Colin Raffel. *Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching*. Columbia University, 2016.

[40] Guan-Horng Liu, Avinash Siravuru, Sai Prabhakar, Manuela Veloso, and George Kantor. Multi-modal deep reinforcement learning with a novel sensor-based dropout.

[41] Relja Arandjelovic and Andrew Zisserman. Look, listen and learn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 609–617, 2017.

[42] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[43] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[44] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[45] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[46] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[47] Stefan Lattner, Maarten Grachten, and Gerhard Widmer. Imposing higher-level structure in polyphonic music generation using convolutional restricted boltzmann machines and constraints. *Journal of Creative Music Systems*, 2:1–31, 2018.

[48] Jouni Paulus, Meinard Müller, and Anssi Klapuri. State of the art report: Audio-based music structure analysis. In *Ismir*, pages 625–636. Utrecht, 2010.

[49] Steffen Schneider, Alexei Baevski, Ronan Collobert, and Michael Auli. wav2vec: Unsupervised pre-training for speech recognition. *arXiv preprint arXiv:1904.05862*, 2019.

[50] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[51] Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics*, pages 492–518. Springer, 1992.

[52] Leonardo Pepino, Pablo Riera, and Luciana Ferrer. Emotion recognition from speech using wav2vec 2.0 embeddings. *arXiv preprint arXiv:2104.03502*, 2021.

[53] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in Neural Information Processing Systems*, 34, 2021.

[54] Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet. Deep learning techniques for music generation–a survey. *arXiv preprint arXiv:1709.01620*, 2017.