NANYANG TECHNOLOGICAL UNIVERSITY

MUSIC GENERATION WITH DEEP LEARNING TECHNIQUES

Paul Solomon Low Si En

Supervisor: Dr. Alexei Sourin

School of Computer Science and Engineering

2024

NANYANG TECHNOLOGICAL UNIVERSITY

SCSE23-0041

MUSIC GENERATION WITH DEEP LEARNING TECHNIQUES

Submitted in Partial Fulfillment of the Requirements
for the Degree of Bachelor of Computer Science and Business
of the Nanyang Technological University

by

Paul Solomon Low Si En

Supervisor: Dr. Alexei Sourin

School of Computer Science and Engineering

2024

1) Abstract

This research paper studies the development and performance of a Text-to-Music Transformer model. The main objective is to investigate the generative potential of the multimodal transformation, where textual input is converted into musical scores in MIDI format. A comprehensive literature review on existing music synthesis methods forms the basis of this study.

This study creates the textual dataset in a novel way by using CLaMP to select the top 30 textual descriptors of the music. A pre-trained RoBERTa model and Octuple tokenizers are used to process the text and musical scores respectively. Thereafter, this music transformer uses neural network architectures with a Fast Transformer base to facilitate the infusion of textual information into generated sequences. Embeddings, linear layers, and cross-entropy loss calculations are used for all 6 musical attributes, with hyperparameter training to promote coherent and varied musical outputs. The generated music was evaluated with a musical analysis and a user study. The results verify that the transformer model can generate music that is either melodious or expresses the textual prompt.

Acknowledgements

I would like to express my gratitude to my FYP supervisor, Professor Alexei Sourin, for his invaluable guidance, support, and recommendations throughout the course of the Final Year Project. His keen insight and vision have been instrumental in helping me grasp the broader implications and applications of the project. From the start, he provided crucial advice and resources that kickstarted my learning and progress.

I would also like to extend my deepest appreciation to my mentor Wang Hanqin. He has been instrumental to the success of this project, with his constant guidance and assistance through the many challenging problems that arose. Thank you for providing me your personal GPUs and for your graciousness in setting aside so much time to meet me.

Also, I would like to thank my family for their support through this whole journey. To Glen, thank you for your encouragements and for reviewing my report drafts. I would also like to thank all the user study participants for their time in sharing their honest feedback.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

2) Introduction

2.1 Background

Technology has always changed how music is generated throughout history. For the piano, the evolution of the harpsichord in the 1600s to the piano in the 1700s added greater dynamic capabilities and nuances to the instrument [1]. The advent of the digital piano in 1980 allowed modern computational technology to reproduce piano sounds digitally [2]. In a similar fashion, the rising popularity of Artificial Intelligence (AI) and ChatGPT has driven research for generative AI and deep learning in music composition, where machine-generated music may become more prevalent than ever before [3]. Through an analysis of 139 recent AI music generation papers, it was evaluated that there is an increased interest in this area of AI research [4]. In addition, more than 70% of the research involves deep learning techniques, with transformers becoming increasingly popular [4].

2.2 Motivation

The motivation for text to music generation by deep learning would be to create a model where musicians can provide textual inputs and receive new generated musical ideas in a MIDI file for their inspiration. This opens new frontiers for musical expression and creative ideas while providing insightful multimodal understanding between text and music.

2.3 Structure of research

To achieve this goal, this research paper will analyze relevant literature reviews of various architecture frameworks and musical generation dimensions. Understanding what has been done will inform the research hypothesis and the project design. The architecture and implementation of the Music Transformer and previous implementations will be discussed. There is followed by a musical and user analysis of the results to determine if the hypothesis has been met. An illustrative example is explored to demonstrate the practical applications of the model.

3) Literature Review

In understanding deep learning techniques for music generation, 3 dimensions can be used for this analysis: Objective, Representation and Architecture. Objective analyzes the content type and usage of the music, Representation involves the musical concepts, formats and encoding while Architecture examines the models used [5].

3.1 Objective

3.1.1 Content Type

In music generation, various content types could be the output of the deep learning model. Some examples range from the melody, which is a sequence of single notes, a polyphony, where multiple notes can be played at the same time, an accompaniment, where other voices harmonizes with the main melody, or a melody with a chordal progression [5]. The differences between the output involves the number of tracks or voices involved, and the number of notes or instruments that can be played simultaneously.

3.1.2 Usage

The usage of the content output could vary depending on the end-user, which also impacts the way the music is represented. A human user would interpret the generated sequence, to allow the human to perform the music. On the other hand, a machine could either play the content through an audio file or process the output through notation software.

3.2 Representation

Representation of music involves how the input and output data of deep learning models are formatted. The options fall into two main categories: Audio Representation and Symbolic Representation which will be explained in the sections below. Encoding is also significant, as it describes how the musical representation is mapped to input variables for the architecture. [5]

3.2.1 Audio Representation

One representation of music is in audio soundwaves, where it can be displayed in its raw audio form or in a transformed mode.

3.2.1.1 Waveform

A waveform is the raw representation of sound, which juxtaposes time with the amplitude of the sound [5]. Time is represented in the x axis while the amplitude is shown in the y axis of the waveform in Figure 3.1. Audio waveforms come in formats such as WAV or MP3 [6].

Figure 3.1 Waveform [5]

3.2.1.2 Chromagram

An example of a transformed audio representation of music is the Chromagram which also maps the notes played in a score to a certain chroma or pitch class. The pitch class is illustrated in diagrams (b) and (d) in Figure 3.2, with the color representing the amplitude.

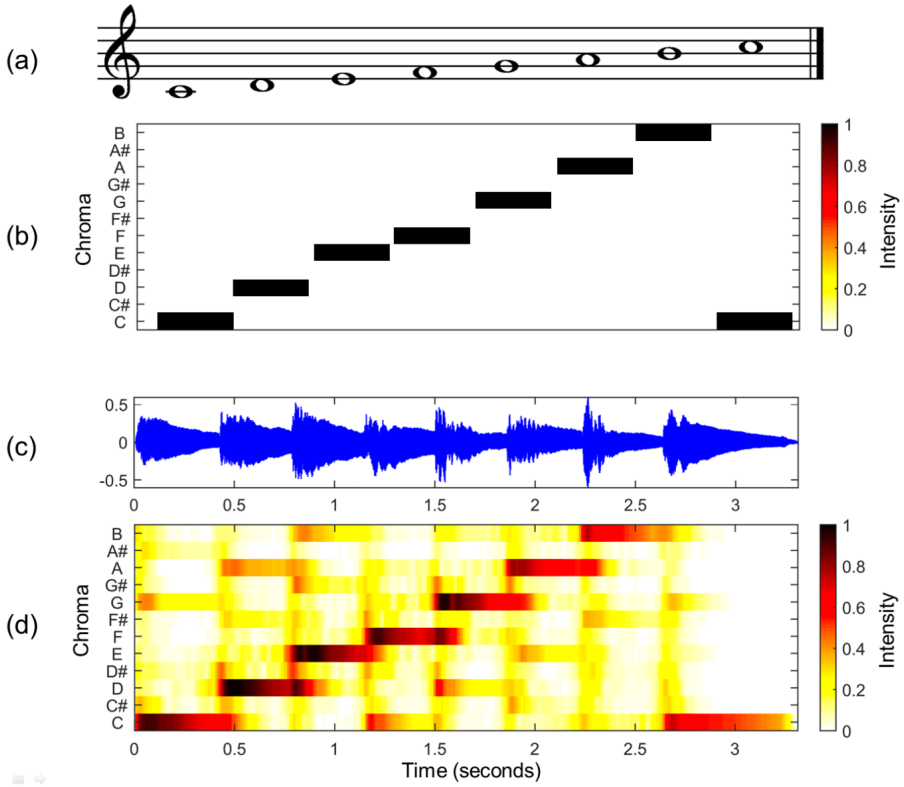

Figure 3.2 Chromagram Diagram [5]

3.2.2 Symbolic Representation

There are various symbolic representations of music that seek to convey various musical concepts in its representation as well. Symbolic Representation can also be described as a bridge between sheet music and audio representation forms [6]. This relationship between the 2 representations and sheet music is shown in Figure 3.3.

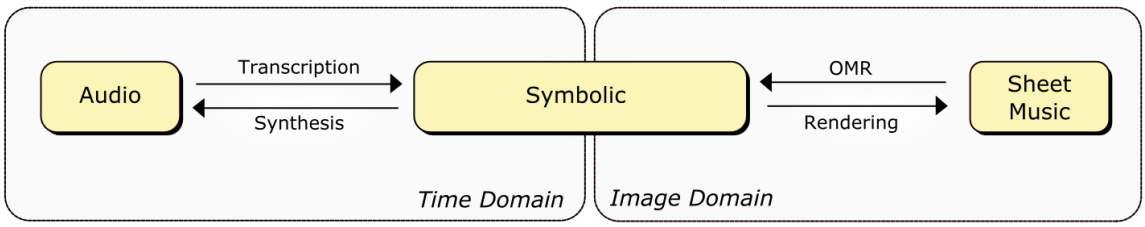

Figure 3.3 Illustration of music representations [6].

3.2.2.1 MIDI

MIDI stands for Musical Instrument Digital Interface, and it is the common benchmark for communication between electronic instruments and software. MIDI conveys information such as Note On to describe that a note is being played, Note Off when the note ends, the note's pitch, time and velocity as well [5].

3.2.3 Encoding

Encoding is the process of converting musical ideas to token sequences that Deep Learning models can use as its inputs. The encoded values can have different variable types and methods.

3.2.3.1 Variable Types

Firstly, variables can be represented as a continuous value encoding such as by defining the frequency of each note in Hertz which results in a real value. As for a discrete version of value encoding, the integer could represent the pitch of each note which falls between 0 and 127. Lastly, values could be represented in a scalar way with 1 for true and 0 for false [5]. Beyond just individual notes, one-hot encoding, many-hot encoding multi many-hot encoding are other techniques used to represent chords at a time slice, or even across different voices.

3.2.3.2 Tokenization Methods

To encode or tokenize music, there are several tokenization methods used that represent various musical information and parameters. MidiTok is a python package that is used for MIDI file tokenization, which provides various tokenization methods such as Midi-Like, REMI, Octuple, MuMIDI, MMM, CPWord [7]. The common tokenization parameters include the pitch_range which encodes the pitch in a discrete value encoding format, nb_velocities which describe the velocity or the note's loudness, beat_res that specifies the time signature of the music. Additional tokens could also be used to encode chords, rests, and the tempo of the music.

3.3 Architecture

The Architecture involves the models used to generate musical output after being trained by a dataset. Through time, some models have grown in popularity and the benefits and disadvantages of the various models will be reviewed.

3.3.1 Generative Adversarial Nets

Generative Adversarial Nets (GAN) is one deep learning model that has been used for music generation. In this framework, the generative model is set against an adversary to improve the generative model's accuracy through competition. GAN is trained with dropout algorithms and backpropagation without Markov chains or approximate inference networks [8].

3.3.2 Convolutional Neural Networks

Apart from GANs, Convolutional Neural Networks (CNNs) are another deep learning architecture applicable to music generation. WaveNet is a CNN used to create audio outputs for text-to-speech and music purposes [9]. It uses hidden causal convolution layers to generate the output from the input as shown in Figure 3.4. After each prediction, the sample would be returned to the network for the next prediction. To reduce computational cost dilated convolutions were used to skip certain input values and cover a wider area.

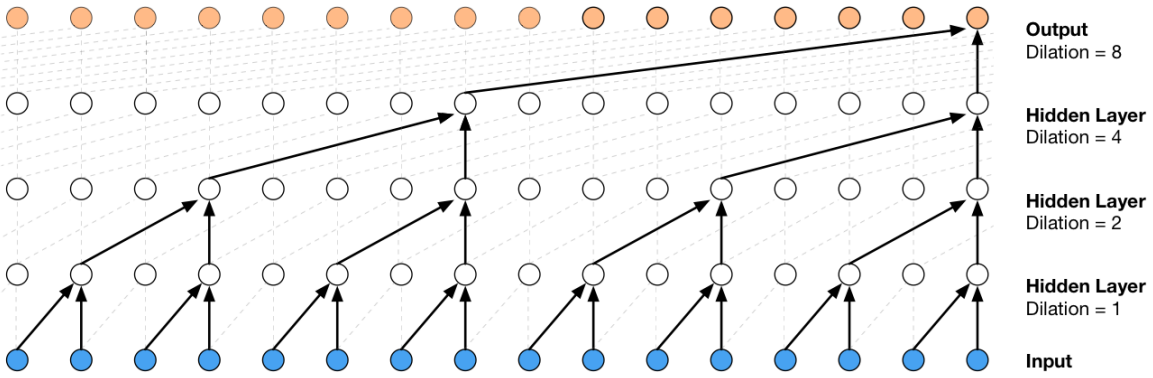

Figure 3.4: Dilated causal convolutional layers of CNN [9]

This model was deemed quicker to train compared to RNNs and achieved better 5-scale MOS tests of 4.21 compared to LSTM-RNN's 3.67 [9].

3.3.3 Convolutional Generative Adversarial Nets

In response to the WaveNet model, a combination of GAN and CNN was used in the implementation of the MidiNet Convolutional Generative Adversarial Nets. It focused on generating melodies originally through a determined chord sequence or based on previous melodies in other bars [10]. Using the concept of GANs, MidiNet likewise uses a generator CNN with random noise inputs, and a discriminator CNN that takes input and predicts if it the input is real or generated. This process trains the model iteratively, with a conditioner CNN being used to consider the previous bar as well, as shown in Figure 3.5.

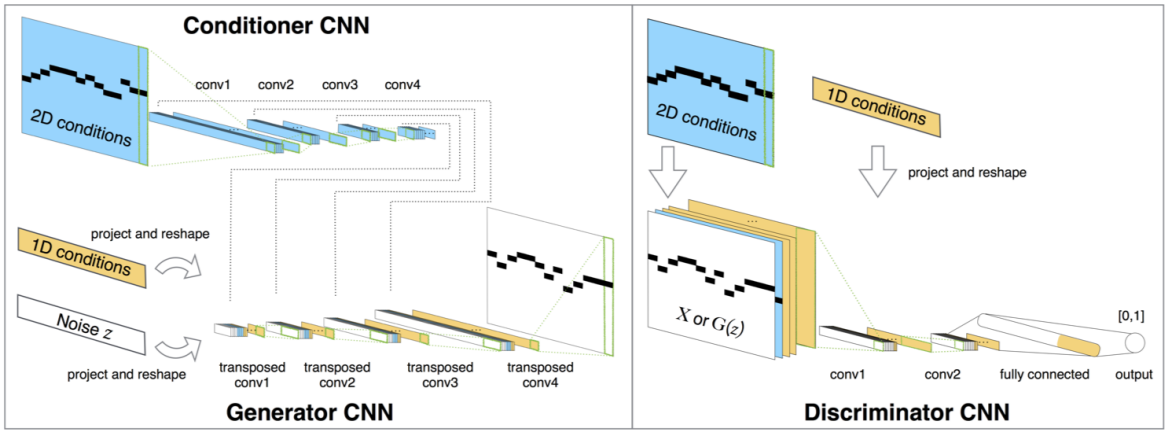

Figure 3.5: System Design of MidiNet Model [10]

The use of these CNNs offer MidiNet the advantage of understanding past sequences and has some creativity due to the use of random noises. It achieved similar musical ratings to MelodyRNN models in terms of realistic music generation even though only the previous bar is considered compared to MelodyRNN's which considers all bars.

3.3.4 Recurrent Neural Network

Various Recurrent Neural Networks (RNNs) exist for music generation purposes and is a more popular option compared to other options such as CNNs [10].

3.3.4.1 Hierarchical Recurrent Neural Network

One of the RNNs that were proposed for music generation was the Hierarchical Recurrent Neural Network (HRNN). The architecture of HRNN comprises of 3 LSTM-based sequence generators: Bar Layer, Beat Layer and Note Layer [11]. Each layers have different functions, with the Bar Layer and Beat Layer creating bar and beat profiles respectively while the Note Layer produces the melody based on the bar profile sequence as shown in Figure 3.6. As seen in the diagram, each layer relied on the sequence output by the higher-level layer. This conditional generation would cause the Bar and Beat layer to learn 16 and 4 times longer temporal structures compared to the Note layer. In addition, the LSTM layers also had 2 hidden layers with 256 neurons per layer each.

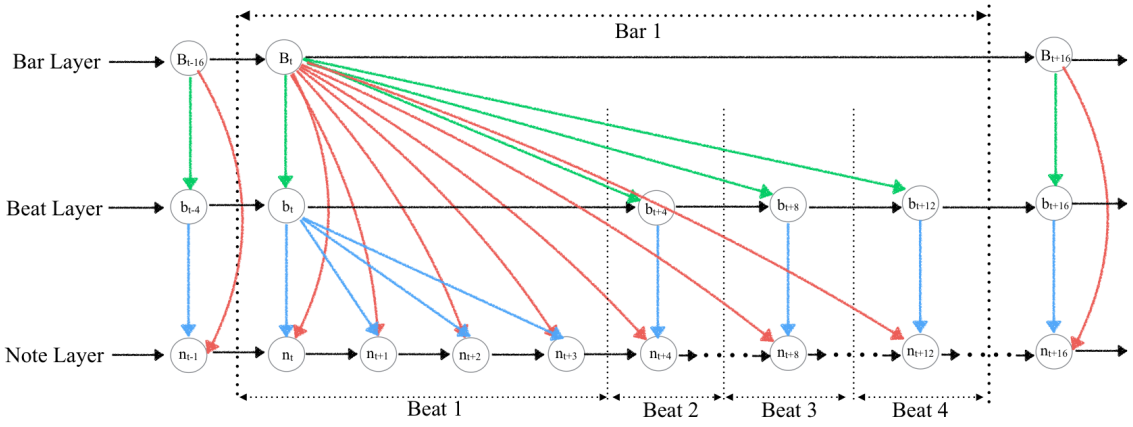

Figure 3.6: Architecture of Hierarchical RNN [11]

3.3.5 Transformers

3.3.5.1 Attention

In "Attention is all you need", Vaswani et al. proposed the transformer architecture that uses self-attention mechanisms without the need for recurrence and convolutions [12]. The Transformers not

only outperformed all other models at the time in text translation tasks but did so with significantly lesser training times. The transformer's architecture can be seen below in Figure 3.7 as it connects the encoder and the decoder with multi-head attention mechanisms and feed-forward networks. The SoftMax function layer at the end also generates output probabilities for prediction.

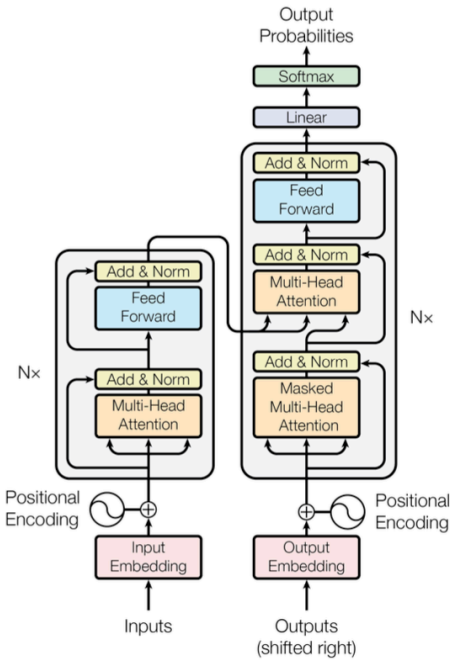

Figure 3.7: Transformer Model Architecture [12]

3.3.5.2 Museformer

In response to the advent of the transformer, researchers sought to apply transformers to music generation as well. Like text, music could also be represented in tokenized sequences, which could utilize the transformer to generate music. One such transformer is the Museformer which is used to generate symbolic musical scores [13]. This Museformer model offers to solve long sequence and music structure modelling to improve the quality of music due to its complex structures compared to text.

Museformer uses fine and coarse-grained attention instead of self-attention for different sections of the music, fine-grained attention for bars related to the music's structures and coarse-grained for the

remaining bars. This allows Museformer to handle longer music sequences better to learn the patterns in music.

3.3.5.3 Compound Word Transformer

Other research articles have also explored alternative methods of generating music through transformers. The compound word transformer uses expansion-compression to convert music to compound words with surrounding REMI tokens [14]. The grey section in Figure 3.9 illustrates how multiple tokens were concatenated, while the transformer uses its attention layers to predict output. This was done using conditional composition where the music was generated given a lead sheet, and unconditional composition where the music was generated from scratch.

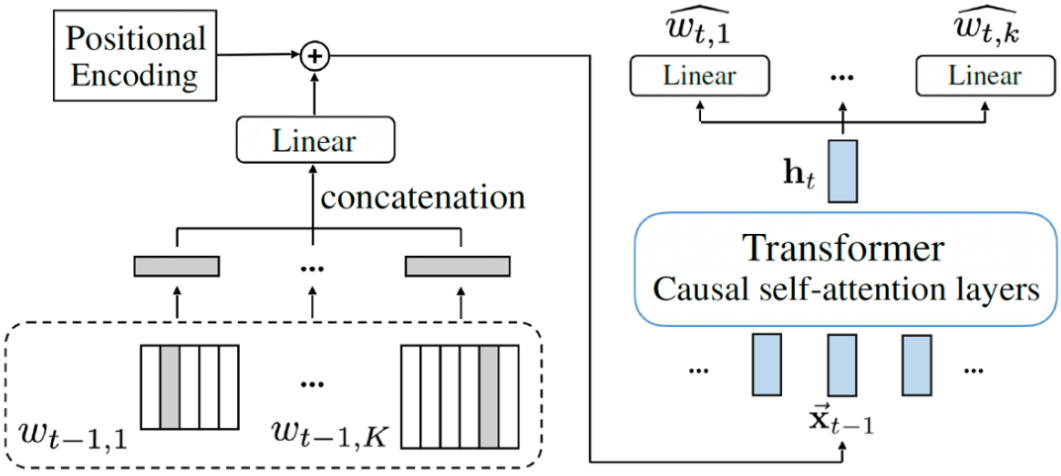

Figure 3.8: Compound Word Transformer Architecture [14]

3.3.6 Natural Language in Music Generation

While the previous areas covered only music generation in isolation, progress has also been made regarding how natural language can associate with music generation. Three of such models are MuseCoco, MuLan and CLaMP.

3.3.6.1 MuseCoco

MuseCoco was designed as a music composition co-pilot to allow musicians to create music from textual prompts. It employs a two-stage architecture of text-to-attribute understanding to extract musical attributes from the text followed by attribute-to-music generation to create symbolic music [15]. MuseCoco uses BERT to determine music attributes from attribute classification and text tokens, before using a transformer to generate the output as shown in Figure 3.9. Each stage is trained independently of each other, with both objective and subjective musical attributes considered.

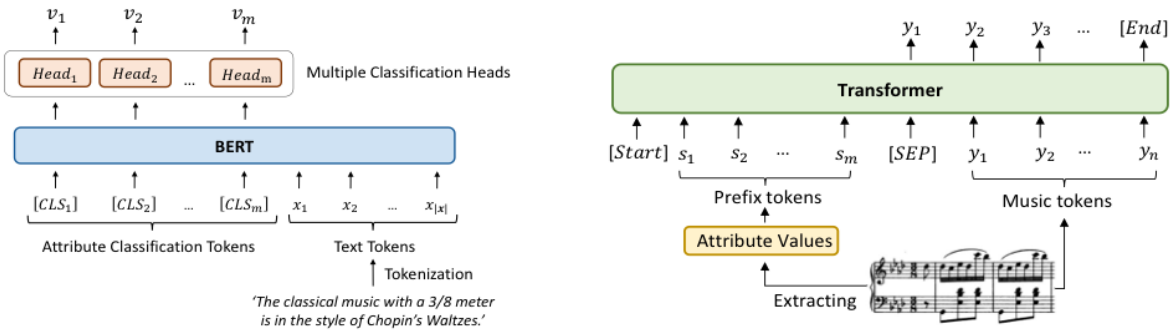

Figure 3.9: Text-to-attribute understanding and Attribute-to-music generation [15]

MuseCoco's music generation had an average accuracy of 77.59% compared to GPT-4's 57.64% and BART's 31.98% based on Musicality and Controllability of the music, showcasing its impressive predictive accuracy.

3.3.6.2 MuLan

MuLan is one of the first attempts to associate music audio to textual music descriptions. With a dataset of music with its musical descriptions, the MuLan model uses a two-tower parallel encoder architecture. This parallel encoder architecture comprises of an audio embedding network that uses Resnet-50 architecture and a text embedding network that uses Bidirectional Encoder Transformer (BERT) to tokenize the text input [16]. Through this process, the usage of a contrastive loss objective creates a shared embedding space between music audio and text. The diagram of the training framework is shown in Figure 3.10.

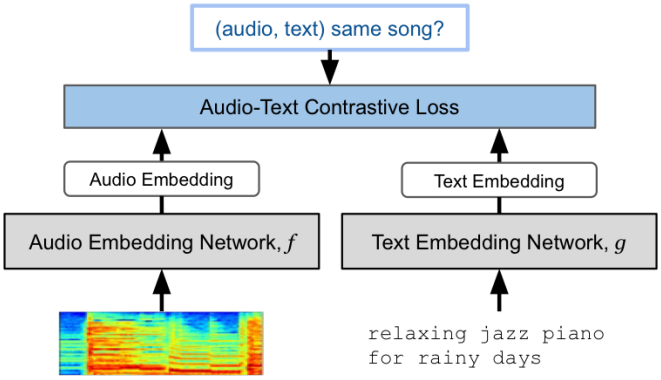

Figure 3.10 Auto embedding of Mulan [16]

3.3.6.3 CLaMP

CLaMP is a model that provides two-way understanding between textual descriptions and sheet music. From text to music, CLaMP can evaluate a textual prompt and generate a musical suggestion that fits the prompt. On the other hand, CLaMP can determine which text descriptions best fits a musical score. This model utilizes a RoBERTa Text encoder and a M3 Music Encoder to train the model to learn cross-modal representations through pre-training and contrastive training, as shown in Figure 3.11 (Wu et al., 2023) [17]. Through a dataset of music-text pairs, the distance between paired music-text examples were minimized while the distance between unpaired examples were maximized with a contrastive loss function. Bar patching and text dropout were also additional techniques used to improve robustness and efficiency of music representation. While CLaMP does not directly generate music it is a helpful model that increases music and textual understanding.

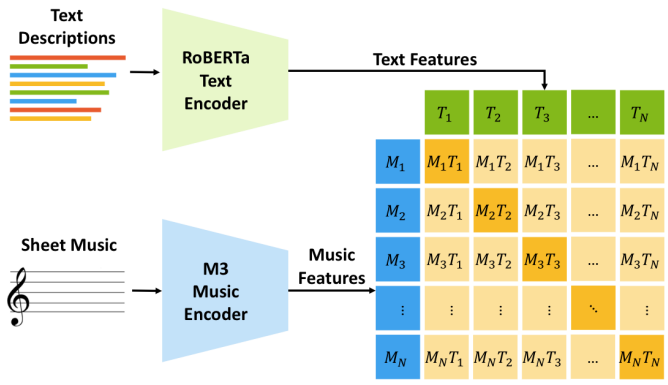

Figure 3.11 Architecture of CLaMP [17]

4) Research Hypothesis

Based on the research, a text-to-music deep learning transformer model will be designed that leverages a two-stage neural network architecture for effective music generation from textual prompts. The initial stage involves a text encoder for text to attribute understanding followed by a transformer to synthesize tokenized musical sequences. It is hypothesized that the use of a transformer would be particularly advantageous, as the attention layers allow for efficient capturing and modelling of patterns within musical sequences [13]. The use of an Octuple as a tokenizer of MIDI files would also be helpful in representing MIDI files with its parameters.

It is hypothesized that the transformer model can generate melodious music that effectively captures the textual prompt used. The representative use of a generated MIDI file as the final output would greatly facilitate user interaction and adoption of the generated music. A MIDI file, being a standard format for musical representation, allows for easy editing of the music to suit their compositional requirements. This would build up a collaborative approach between AI and human composer, to reap the benefits of both parties as the human composer arranges the ideas from the AI generator. The research through a systematic review of AI music generation [3] also supports this approach as the way forward in AI music generation.

5) Design

This section describes the plan to design and create a text to music generation AI model, as well as the model's dimensions of Objective, Representation, and Architecture.

5.1 Project Plan

Year	Month	Plan
2023	August	• Determine project direction and details • Research on past work on deep learning music generation • Start writing Literature Review
	September	• (4 Sep) Submit Project Plan to Supervisor • Establish dataset of music-text pairs • Reimplement open-source models • Finish Literature Review on past work and determine research hypothesis
	October	• Experiment with different architecture approaches • Add improvements to previous models
	November	• Train models with different weights and evaluate the results
	December	• Determine results of the research to evaluate hypothesis • Begin writing Interim Report
2024	January	• Further correction of the model and weights • (29 Jan) Submission of Interim Report
	February	• Finalize Model • Conduct user testing and analyze results
	March	• Complete Final Report • (25 Mar) Submission of Final Report
	April	• (19 Apr) Submission of Amended Final Report • Prepare for Oral Presentation
	May	• (3, 6-8 May) Oral Presentation

Table 5.1: Project Plan

The major goals were accomplished on time in this project plan as shown in Table 5.1. However, the unexpected difficulty in generating an effective transformer that produced versatile and effective

results made it challenging to complete the model according to the project plan. More time was taken to improve the quality of the model and the results.

5.2 Objective

The objective of music generation refers to both the musical content generated as well as the use cases of the model.

5.2.1 Content Type

The form of the musical content being generated depends on the dataset used to train the model. As the dataset uses complete musical scores, model would also be able to output full musical scores. This goes beyond just a simple melody and involve polyphonic sequences with multiple notes. In addition, multiple instruments can be used within the track as well.

5.2.2 Usage

As will be further described in Section 5.3, the output of a MIDI file allows both humans and machines to interact with the content output. A human user can continue editing the MIDI file to further refine and improve the melodic harmonies generated. On the other hand, a machine can interpret the MIDI file to play the audio content or process it through notation software.

5.3 Representation

As seen in Section 3.2, representation normally involves 3 main areas: Audio Representation, Symbolic Representation and Encoding. As this research primarily focused on MIDI file generation, Symbolic Representation is prioritized over Audio Representation. Regardless, audio files can still be synthesized from the symbolic MIDI file.

5.3.1 Symbolic Representation

The Symbolic Representation of a MIDI file is commonly used as a technical standard in music production. Unlike other forms of symbolic representation like ABC notation and Piano rolls, it can store more information and support multi-track and polyphonic harmonies better. The information stored in a MIDI file can then be tokenized for the deep learning architecture to process.

5.3.2 Encoding

To convert musical ideas to token sequences, MidiTok's Octuple was used as the preferred tokenization method. From a MIDI file, the Octuple tokenizer will encode the entire file into a TokSequence of Octuples. Each Octuple contains 6 tokenization parameters: Pitch, Velocity, Duration, Position, Bar and Program to represent each note in the MIDI file [18]. Each of them is discrete, using whole numbers to capture the following information as described in Table 5.2. Tempo and TimeSignature are optional tokens and were not used in this study. The number of token classes is illustrated in Table 5.2, however the Bar Parameter has a variable token class vocabulary size. It was observed that the number of token classes would be 64 when the tokenizer was initialized, but when the tokenizer tokenizes a MIDI file with more bars than the current Bar Parameter classes, the Bar Parameter of the tokenizer would increase accordingly. (Note that some details from Zeng et al.'s paper differ from MidiTok's Python implementation of Octuple)

Parameters	Description	Number of Token Classes
Pitch	Encodes the pitch value of the note	92
Velocity	Encodes the velocity or loudness that the note would be played	36
Duration	Encodes how long the note will be sustained in the piece	68
Position	Encodes the position within the bar that the note is in	36
Bar	Encodes the Bar which the note is in	Default 64 and increases according to MIDI file*
Program	Encodes the Instrument used to perform the note	133

Table 5.2: Encoding Parameter of Octuple

For each of the parameters, 4 values are used to encode PAD_None, BOS_None, EOS_None, and MASK_None respectively. PAD_None is captured in discrete value 0 to allow for padding of smaller tensors to fit the same length of other tensors if required. BOS_None is represented by 1 and would signify the beginning of the musical sequence while EOS_None is represented by 2 and would indicate the end of the musical sequence. Lastly MASK_None is captured by 3 and is used for pre-training.

The choice of using Octuple over other one-dimensional tokenizers include its effectiveness in creating shorter token sequences, especially since music sequences are already quite long. It is up to 4x shorter than REMI sequences which can help to increase efficiency in training (Zeng et al., 2021) [18]. It is also note-centric with each 6-element vector in the octuple representing a note in the MIDI score, making it clearer and more universal as a result. Lastly, Octuple supports multi-track encoding through its Program variable, which enables us to consider multiple instrumental tracks in this project.

5.4 Architecture

With this research project's hypothesis being that a transformer can generate musical sequences from text efficiently, the architecture in Figure 5.1 was designed and used.

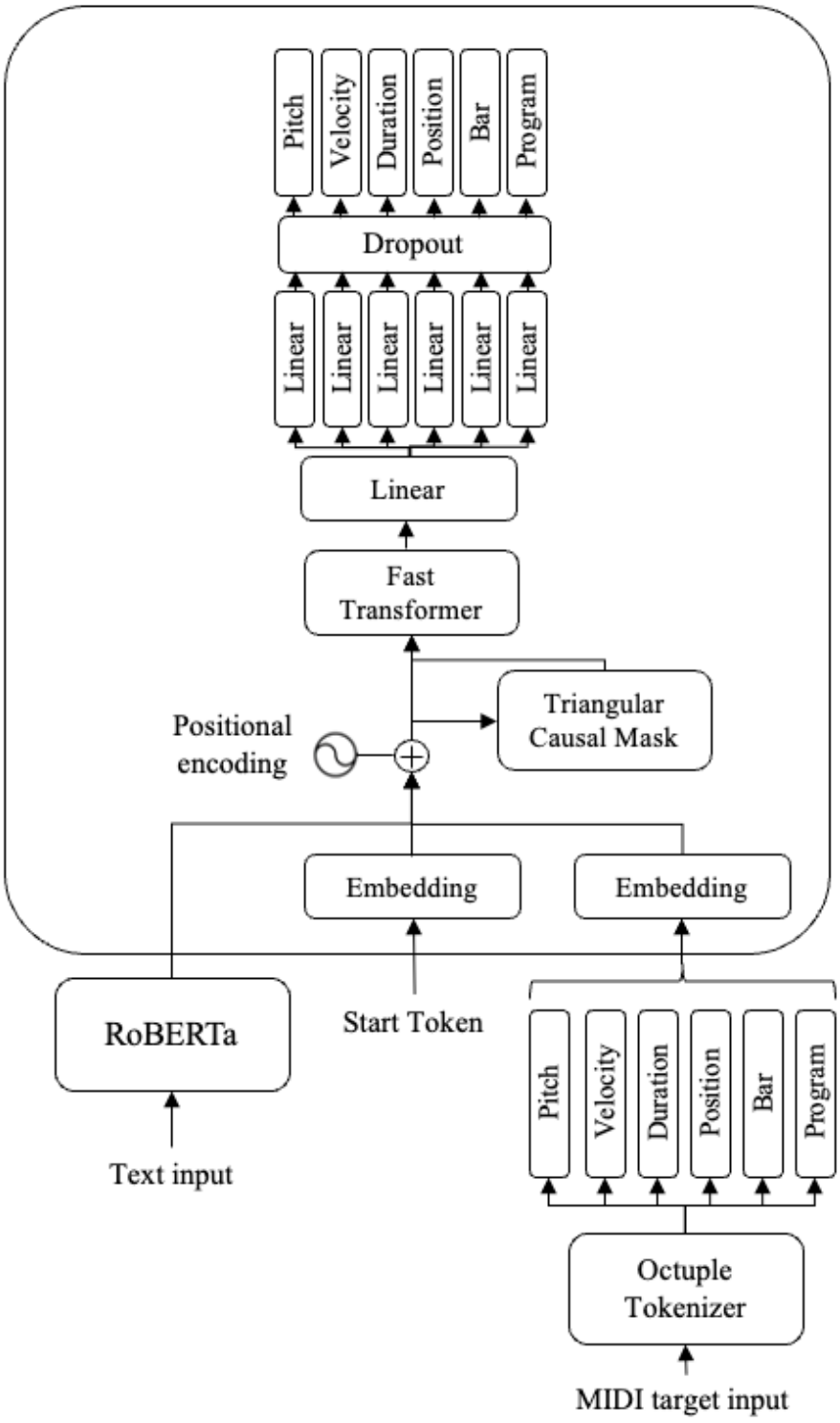

Figure 5.1 Music Transformer Training Architecture

5.4.1 Architecture Overview

The MusicTransformer's architecture uses pretrained RoBERTa, fast transformers and Octuple tokenizers in addition to our designed neural network architecture. The various layers used will be explored in greater depth in the following subsections. On a higher overview, after tokenization by RobERTa and Octuple, an input tensor is created by concatenating the embedded text, embedded start token and the embedded MIDI target. This input tensor is positionally embedded via positional encoding before being converted to an attention mask. Both the positional encoded input and the attention mask would be passed to the fast transformer. The output from the hidden layer would then be passed to an initial linear layer to convert dimensions of the outputs from 768 to 256. Next, each parameter will have their separate linear layer before reaching the final dropout layer. This whole process is illustrated in Figure 5.1.

Subsequent designs of the training loops process and evaluations are covered in the implementation in Section 6.2.3.

5.4.2 RoBERTa

RoBERTa is a language model which robustly optimizes BERT models and through training on text data, with improved generalization and robustness compared to BERT. The model is better able to capture nuanced languages and accomplishes state-of-the-art results on various GLUE, RACE and SQuAD benchmarks [19]. This pretrained model was used to process and understand the textual input given by the user to produce 2-dimensional tensor sequences of variable lengths. These text tensor sequences could then be appended with the embedded start token and the embedded MIDI token subsequently.

5.4.3 Start Token and Embedding Layer

The Octuple Start Token had a shape of (batch_size, 6) with each integer 1 representing the start token for each of the 6 elements in an Octuple. This start token is used to signal the start of the sequence for the model, so that model has a clear indication of where the sequence generation should start from.

This start token would then be embedded through 6 different embedding layers, each one specific to each parameter of the octuple. Each embedding layer had a dimension of 128 but had different number of embeddings depending on the vocab of the Octuple tokenizer. The vocabulary size of the Octuple tokenizer is reflected in Table 5.2. These embedding layers are used to transform the varied input dimensions into consistent dimensions of 128 for all the parameters. In addition, embedding also converts the categorical values of the octuple classes into continuous vector representations. Each token is associated with a unique embedding vector, which captures the relationships between tokens.

5.4.4 Positional Encoding Layer

Positional encoding is a technique in transformer-based models to introduce the positional information of the musical sequences. This is necessary as input sequences are normally treated with no inherent order, while musical sequences require this positional information due to its sequential nature. Rather than using indexes to encode the position, which have issues for long or variable sequences, the indexes are converted to vectors through sine and cosine functions. [20] This sine and cosine functions can be illustrated in Figure 5.2's formulas which would encode the indexes and ultimately result in a matrix output for the positional encoding layer. Through this, the model will gain a better understanding of the positional layout of the inputs.

$$P(k, 2i) = \sin\left(\frac{k}{n^{2i/d}}\right)$$

$$P(k, 2i+1) = \cos\left(\frac{k}{n^{2i/d}}\right)$$

Figure 5.2 Positional encoding sine and cosine functions [20]

In this formula in Figure 5.2, P(k, j) is the position function to map the input to the matrix, k refers to the object's position within the input sequence and d refers to the output embedding's dimension. n is a scalar value that is commonly defined at 10,000 from Vaswani et al.'s Attention is All you Need [12]. Lastly, i is used to map indexes to the matrix's columns, as each value of i is used to map token indexes to both cosine and sine columns.

5.4.5 Triangular Causal Mask

Triangular Causal Mask is an attention masking implementation in the fast transformers library [21]. It is used to ensure that each token in the sequence only looks at previous tokens or tokens in its same position. It is implemented as a square matrix, with every index above the main diagonal masked, to hide the subsequent sequences from the model's current computation [22]. This enables the model to make predictions with only the current information it has, to better guide the learning process of the model. An illustration of how the triangular causal mask works is shown in Figure 5.3, where each cell is the attention weight and the grey cells on the right is when the Triangular Causal Mask is applied.

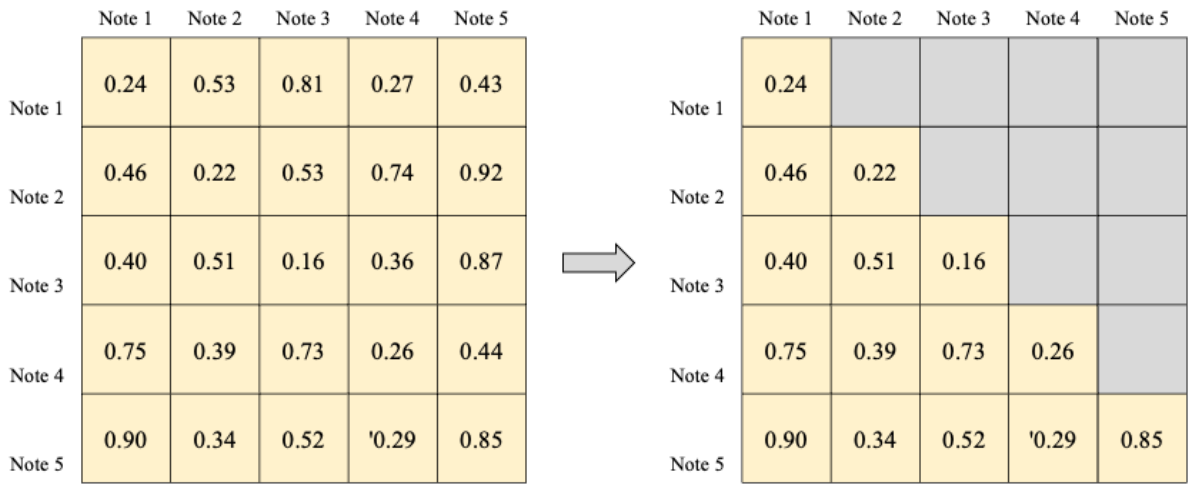

Figure 5.3 Triangular Causal Mask Application

5.4.6 Fast Transformers

The Fast Transformer model is an efficient transformer architecture that is used to speed up the transformer's prediction. Through the Fast Transformer architecture proposed in Katharopoulos et al.'s "Transformers are RNNs" paper, [23] the model obtained similar performance to traditional transformers, but with 4000x the speed. This is achieved through kernel feature maps and matrix products which improve the quadratic time complexity to a linear complexity.

This fast transformer is integrated into the entire MusicTransformer and is used in the MusicTransformer's forward method to process the data. In the forward method, the transformer takes in the positional encoding output and triangular causal attention mask before generating prediction outputs. Additional information on the library of Fast Transformers is written in Section 5.5.3 and information on the implementation of the Fast Transformers Builder is written in Section 6.2.2.1.

5.4.7 Hidden Layers

Hidden layers are the intermediate layers that capture dependencies in the sequential data. Latent vectors, which are the intermediate representations would undergo transformations and aggregations to produce the output.

5.4.8 Linear Layers

Linear layers are used in neural networks as a linear transformation to the input data. This helps to decrease the number of dimensions between an input and its output while saving the relevant information and complex patterns within the data. In this MusicTransformer, there are 2 separate linear layers. The first is a linear transformation from 768 to 256, with the input dimension of 768 is based on the feed forward dimension of the fast transformer.

The music transformer also has linear layers for each of the separate parameters. These specific linear layers would further reduce the dimensionalities to the vocabulary size of each of the octuple

parameters. These outputs after linearization are called logits, which are the unnormalized scores produced by the linear transformation.

5.4.9 Dropout

Finally, the dropout layer is a regularization technique in neural networks to reduce overfitting and aid the model in generalizing its predictions. It works by randomly dropping a percentage of the neurons in a layer during training, preventing their propagation further down the transformer. This is controlled by probability p, where p would be the dropout rate, or the number of neurons dropped in the layer during training. Through the dropout layer, this adds more noise into the network, which would prevent the model from over relying on certain patterns or features.

5.5 Programming Language and Libraries

5.5.1 Python

Python is also considered one of the most powerful languages used for deep learning. It has various advantages such as having concise syntax, making it easier to learn and read [24]. In addition, unlike other languages like Java and C, Python has numerous open source packages and libraries [25] that can be imported to reduce boilerplate code. This wide community support for the libraries also makes Python more accessible and suitable to be used for deep learning.

5.5.2 PyTorch

PyTorch is a popular Python library that is used as a framework for deep learning modelling [26]. It is a powerful tool that allows for efficient computation of tensors with high accuracy results. It is considered an expressive language which makes it easier to code in and provide adaptability.

5.5.2.1 Neural Networks

The foundation of the neural networks architecture is defined by the nn or neural network class [27]. PyTorch provides the framework for all the various layers used in a neural network, from linear to embedding layers and utility functions from dropout to Softmax. Modules of neural networks include the forward step which defines the forward pass computation of the model, data parallelization which allows for distributed processing, and the definition of the loss function used [26]. Additional details of these features used in the architecture can be found in Section 5.4.

5.5.2.2 CUDA

Torch also provides support for interfacing with CUDA, that allows NVIDIA GPUs to be used for deep learning purposes. CUDA allows GPUs to be utilized instead of CPUs, resulting in synchronized and accelerated computation [28]. CUDA has its specific tensor operations to be done and provides its own functions for managing devices and memory. With the complexity of computation needed, greater efficiency is needed to speed up the process. The usage of DataParallelization would further enable the synchronized usage of multiple GPUs in model training.

5.5.2.3 Dataset

Another important class in PyTorch is Dataset. Dataset is used to store the collection of data, including both input and target data [29]. These data are used by the DataLoader to iterate through during training and evaluation. A Dataset class has 3 main functions: __init__ to initialize the dataset, __len__ to denote the total amount of data and __getitem__ to generate the next sample of data. The implementation of this custom Dataset class will be addressed in Section 6.1.4.

5.5.2.4 DataLoader

The DataLoader is a PyTorch class that helps to load data from the Dataset class before iteratively feeding it to the deep learning model. It preprocesses the data and applies batching to group the data into specific sizes. A collate function can further define how data is collated into batches suitable for

the model. Shuffling is also used to ensure data order doesn't affect the results, and ensure robustness. [29]. The implementation of this DataLoader will be addressed in Section 6.1.5.

5.5.3 Fast_transformers

Fast_transformers is a package used to implement an accelerated autoregressive transformer that has improved performance for longer sequences of [23]. The use of clustered attention also helps in addressing quadratic complexity issues that arose through computing attention [30]. This package's linear transformers can achieve similar performance of other transformers but with up to 4000x the speed. To build a Transformer Encoder, a TransformerEncoderBuilder class is typically used, and various attributes of the transformer can be defined with its attention_type, layers, heads, forward and query dimensions. Similar processes are done to create a TransformerDecoder, RecurrentTransformerEncoder or RecurrentTransformerDecoder.

In addition, this package also includes support for masking, with various implementations of masking from FullMask to LengthMask and TriangularCausalMask. Due to this project using an autoregressive transformer, TriangularCausalMask was the mask used.

6) Implementation

This section would address the developmental details, particularly focusing on the training and evaluation process.

6.1 Dataset Preparation

The Dataset for the Music Transformer required both text files as well as MIDI files. Even after obtaining the MIDI files, each MIDI file had to be paired with a relevant text file. Thereafter, the files had to be processed to form the dataset for the Transformer.

6.1.1 Creation of MIDI files

6.1.1.1 WikiMusicText Dataset

The CLaMP project also introduced the WikiMusicText Dataset, which consists of 1010 MusicXML lead sheet files [17]. This resource was created to support music classification and taken from publicly available sources of music. To use this dataset in this research, the MusicXML files were converted to MIDI files, so that they could be tokenized. An example of a MIDI file after conversion can be seen in Figure 6.1.

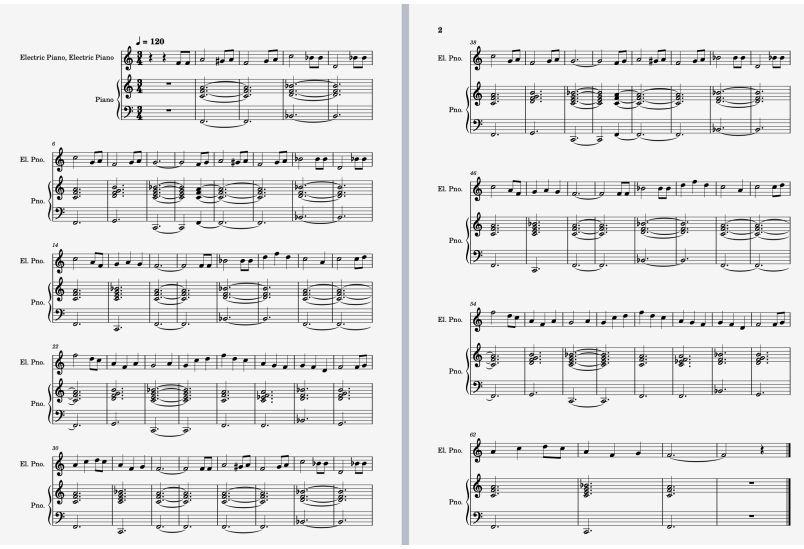

Figure 6.1: MIDI file example

The first few iterations of the model only used the WikiMusicText dataset and had 1010 entries in the dataset to train the model. The WikiMusicText dataset did not encounter errors in Octuple Tokenization and had much shorter sequences. As such many of the dataset processing efforts described in Figure 6.6 for Lakh MIDI and MetaMIDI were not necessary for this dataset.

6.1.1.2 Lakh MIDI Datasets and MetaMidi Dataset

To further supplement the WikiMusicText-Dataset, subsets of the Lakh MIDI Dataset [31] [32] and the MetaMidi Dataset [33] were used to increase the dataset size from 1010 files to 5000 files. Lakh MIDI Dataset collates 176,581 unique MIDI files, with LMD-matched being a portion of 45,129 files that align with music from the Million Song Dataset. MetaMidi Dataset is another extensive 436,631 MIDI files with metadata regarding artists, titles, and genres.

6.1.1.3 Processing of Lakh and MetaMidi Datasets

Due to the size of these datasets, more processing had to be done to refine the MIDI files to suit this project's purposes. Many MIDI files especially in the Lakh MIDI dataset were very long, and when tokenized, some of them could reach a sequence length of up to 28,011. This affected setup time significantly by slowing down the Octuple's tokenization of long MIDI files. It could take up to an hour to tokenize all 5,000 files. As such, a limit of 2,000 sequence length was used to filter the MIDI files. This filter removed approximately 90% of files from the Lakh MIDI dataset and 49% of the MetaMIDI dataset. This improved tokenization time for all 5000 files from 55 mins to 5.36 mins. 1990 Lakh MIDI files were used and 2000 MetaMIDI files were used subsequently.

In this processing format, some MIDI files failed to be tokenized by the Octuple or could not be effectively converted to a MusicXML format for CLaMP. These defective files were removed from the dataset and replaced by other short MIDI files.

6.1.1.4 Transposition of MIDI files

To further improve the accuracy of the model, the MIDI files were also standardized to be of the same key or its relative minor. By nature, most of the MIDI files in the dataset were originally of different keys, which made training of the MIDI files more challenging, as even the same song but in a different key would have a completely different octuple pitch variables. As such the generated pitches during evaluation were initially more random and harder to converge. To improve this, each of the MIDI files were analyzed to find their key signatures. If the music score was in a major key it was transposed to C major while if the music score was in a minor key it was transposed to A minor (which is C major's relative minor). C major was used as it doesn't have a key signature which reduces the complexity of the music piece. This can be illustrated by the Circle of Fifths in Figure 6.2 with shows all the possible keys and their relationship with C major and A minor by transposition.

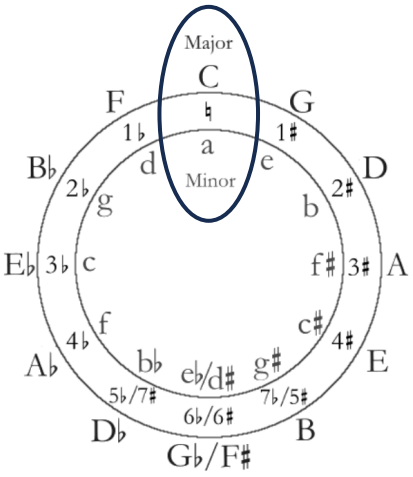

Figure 6.2: Circle of Fifths [34]

This process was done using music21's library to identify the key signature and prettyMIDI to transpose the MIDI files for better fidelity of the music. For example, Figure 6.3 shows an excerpt from a MIDI file in the dataset: 'Deed I Do. When transposed to C major, prettyMIDI can convert it to C major accurately and keep the structure of the piano and voice tracks intact in Figure 6.5. On the other hand, a similar code structure by music21 as shown in Figure 6.4 sometimes fails to transpose

the MIDI file accurately and merges the different tracks into 1 score which shows its poorer fidelity in transposition.

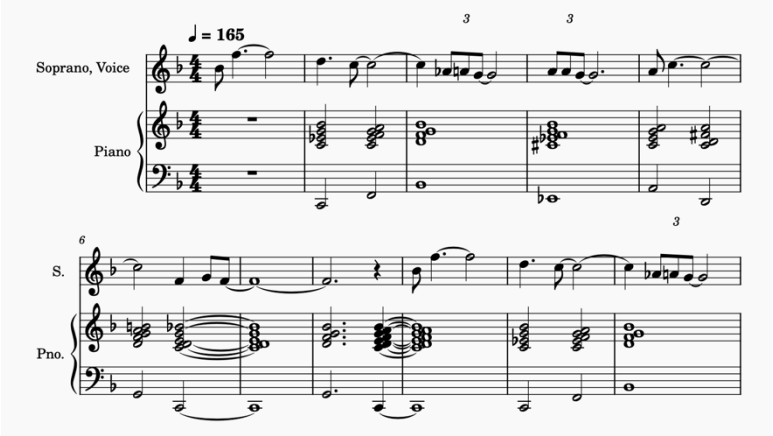

Figure 6.3: Excerpt from initial 'Deed I Do.mid file in F major

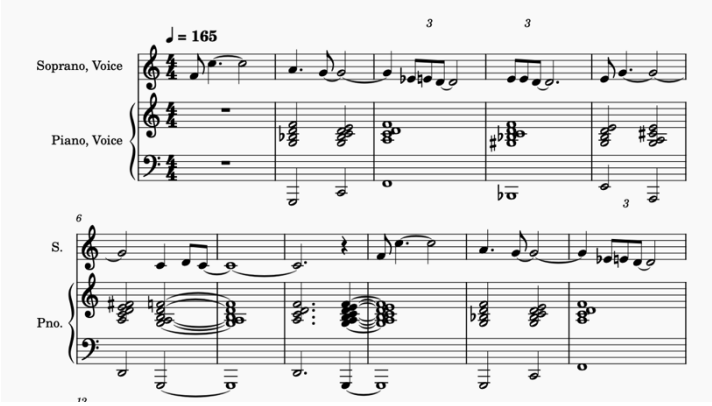

Figure 6.5: Transposition to C Major by prettyMIDI

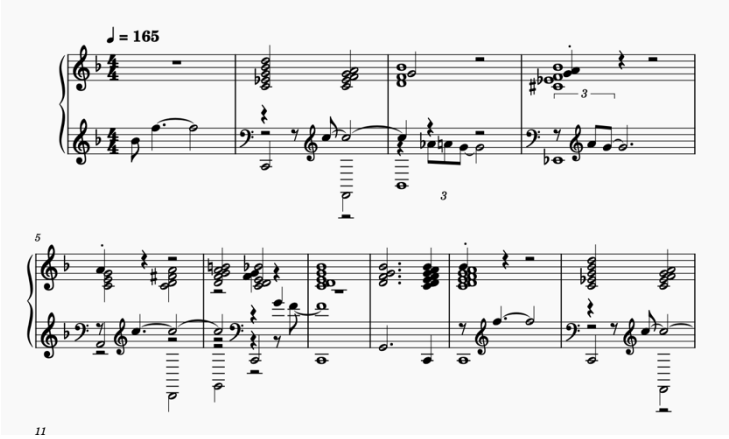

Figure 6.4: Transposition by music21 with issues

6.1.1.5 MIDI file generation Overview

To summarize the MIDI generation setup process, the MIDI files used to generate the dataset was taken from 3 sources as described in Table 6.1 to allow for a more diverse range of MIDI files from various sources.

	MIDI Files used	Dataset Size	Descriptions
WikiMusicText Dataset	1010	1010	All short sequences
Lakh MIDI Dataset	1990	176,581	90% files greater than 2000 sequence length
Meta MIDI dataset	2000	436,631	49% files greater than 2000 sequence length

Table 6.1: Summary table of Datasets

A process flow of the dataset generation process can be shown in Figure 6.6 for each of the 3 datasets. There are multiple processes needed to ensure that the music sequences could be tokenized by the octuple, were short enough for quicker processing, could be transposed by prettyMIDI to C major or A minor, and could be converted to musicXML. Lakh Dataset and MMD Dataset had files that failed at various points in this pipeline process. These files had to be removed and this whole process had to be repeated for the new file. At all points, it was crucial to ensure that files were not duplicated, and each MIDI file had a correspondingly named text file.

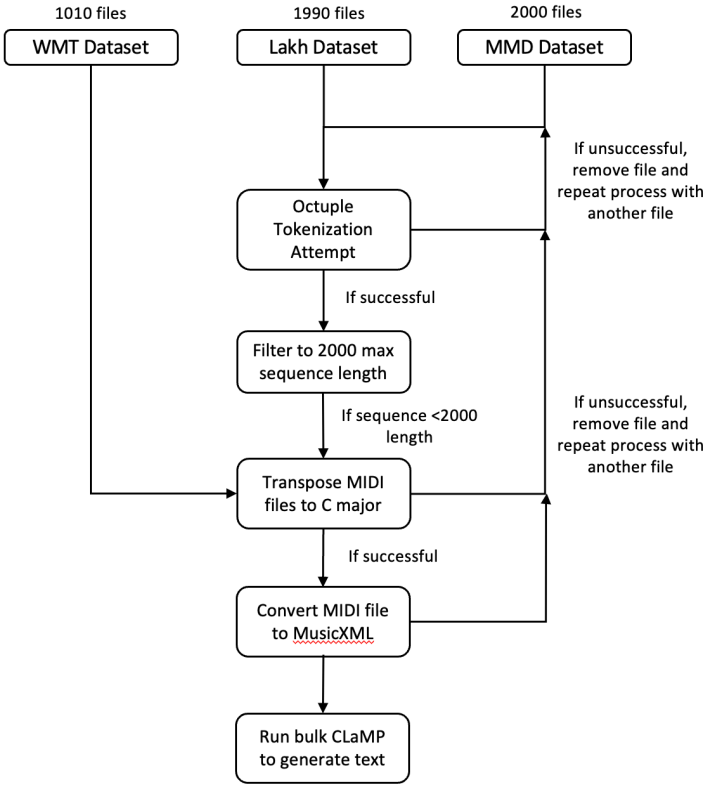

Figure 6.6: Flowchart of Dataset Generation Process

6.1.2 Creation of text files

6.1.2.1 WikiMusicText Dataset

Even though the WikiMusicText Dataset contained text files as descriptors for each MusicXML file, they were taken from the Wikipedia articles such as in Figure 6.7. Such text files contained more

information about the history of the music rather than a musical description of what the music is about, which would be less relevant for use in training the model.

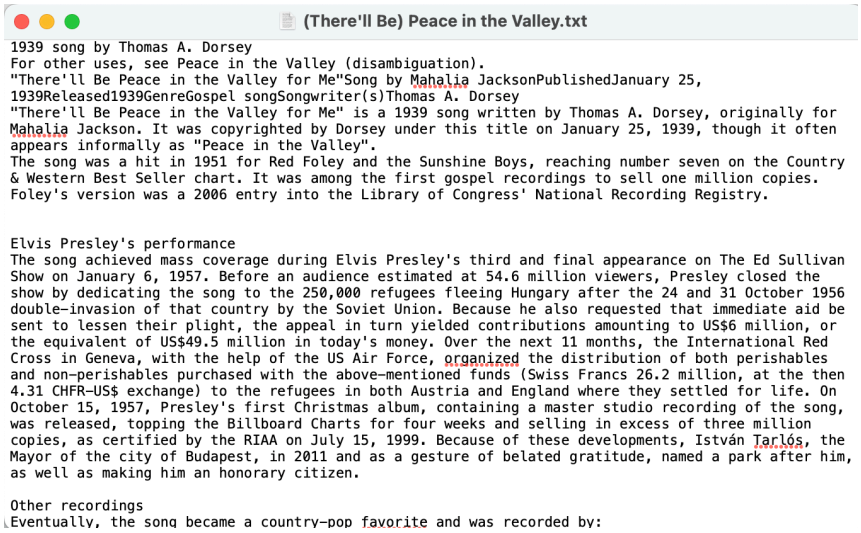

Figure 6.7: WikiMusicText Dataset Text Files

6.1.2.2 Creating Music Descriptors for Dataset

As such to generate text files with more musical descriptions for the dataset, the CLaMP model was utilized. As introduced in Section 3.3.6.3, CLaMP can process a MusicXML file and select the top n descriptors that best described the piece of music [17]. As such, a list of 113 music descriptors was created to serve as the foundation for describing each piece.

To create this list of 113 music descriptors, the musicLM dataset of descriptors was referenced [35]. This dataset contained 5,521 music audio files with text captions and an aspect list for each of the audio files. This aspect list was extracted from the dataset and ranked in accordance to how often each aspect descriptor was used in the dataset. A snapshot of the results before further processing can be seen in Figure 6.8. This list had to be further filtered to remove descriptors of audio quality and singers as they were not relevant to our MIDI files dataset. Repetitive (medium tempo and moderate tempo) or very similar descriptors (bass guitar, electric guitar, acoustic guitar) were also removed at this stage.

	Value	Count
0	low quality	1220
1	instrumental	908
2	emotional	636
3	noisy	628
4	passionate	610
5	medium tempo	608
6	energetic	599
7	amateur recording	518
8	live performance	477
9	slow tempo	476
10	acoustic drums	394
11	fast tempo	383
12	mono	358
13	groovy	321
14	bass guitar	297
15	electric guitar	297
16	acoustic guitar	292
17	instrumental music	260
18	uptempo	247
19	piano	240
20	punchy kick	239
21	poor audio quality	236
22	punchy snare	235
23	groovy bass	229
24	happy	228
25	male voice	224
26	moderate tempo	224
27	e-bass	222
28	shimmering hi hats	217
29	male vocal	208
30	rock	207

Figure 6.8: 30 Most common Aspects used in musicLM dataset before processing

From the remaining aspects, the top 100 most common aspects were extracted from the dataset and sorted according to which musical category the descriptors fell under. Among these top 100 common aspects, it was observed that a huge portion of the descriptors were mostly describing the mood, genre, tempo or instrumentation of the music, rather than commenting on other musical parameters of the MIDI file such as texture, dynamics and rhythm. As such, aspects that described musical parameters had to be added, to give a more holistic picture of the music piece. With a better understanding of how music could be described [36], 10 categories of music descriptors were created to describe the music: Mood, Genre, Tempo, Dynamics, Rhythm, Form, Instrumentation, Texture, Timbre and Melody/Bass Line. Some mood and less helpful aspects were removed and replaced with aspects from other sections. The list of all 113 music descriptors and the definition of each category are depicted in Table 6.2.

Music Descriptors		
Category	Explanation of category [36]	Descriptors
Mood (31)	Emotional quality or atmosphere	Exciting, vibrant, joyful, playful, buoyant, animated, uplifting, inspiring, passionate, energetic, groovy,

	conveyed by the music	aggressive, funky, epic, romantic, sentimental, melancholic, soulful, nostalgic, heartfelt, relaxing, melodic, pleasant, chill, dreamy, meditative, engaging, suspenseful, dramatic, distorted, eerie
Genre (11)	Distinct styles of music, determined by musical characteristics and culture	Rock genre, electronic genre, ambient genre, hip hop genre, pop song, film, music genre, jazz genre, love song, classical genre, country genre, folk music genre
Tempo (7)	Descriptors of the music's speed and tempo	Upbeat tempo, driving tempo, leisurely tempo, measured tempo, languid tempo, flexible tempo, rubato
Dynamics (10)	Difference in velocity or loudness throughout the piece	Loud, soft, lively, soothing, intense, sudden dynamic shifts, gradual build up, steady dynamics, big range of dynamics, smooth dynamic contours
Rhythm (7)	Pattern of beats and pulses over time	Steady rhythm, syncopated rhythm, offbeat rhythm, swinging rhythm, irregular rhythm, regular rhythm, pulsating rhythm
Form (8)	Musical structure of the piece, its organization of section and themes	Repeated theme, verse and chorus structure, binary structure, ternary structure, theme and variations, complex structure, linear structure, cyclical structure
Instrumentation (8)	Selection of musical instruments used	Piano, percussion, bass, strings, orchestra, brass, multiple instruments, single instrument
Texture (10)	Density and layering of melodies and harmonies	Simple harmony, complex harmony, polyphonic texture, homophonic texture, sparse texture, dense texture, arpeggiated accompaniment, chordal accompaniment, grand texture, dissonant harmony
Timbre (7)	Temperature or color of the music and how it sounds	Reverb, mellow tone, harsh tone, warm sound, cold sound, dark music, clean music
Melody / Bass Line (14)	Melody refers to the sequence of musical notes that describe the main theme, while bass is the lowest pitched music section.	Sustained melody, catchy melody, percussive bass line, flowing melody, haunting melody, contemplative melody, walking bass line, sustained bass line, rapid succession of notes, dramatic pause, extended musical lines, brief musical segments, staccato, legato

Table 6.2: Music Descriptors for CLaMP model

To use CLaMP, a script was used to convert each MIDI files to MusicXML files first as shown in Figure 6.6. Thereafter, CLaMP was run using these 113 descriptors to find the top 30 descriptors for

each MIDI file. These 20 words were then used as the text descriptors for each music file as seen in Figure 6.9.

Figure 6.9: CLaMP generated music descriptors for dataset

6.1.3 Automation Scripts for Dataset Creation Process

6.1.3.1 Jupyter Notebook

As many files were being processed at any point in time, automation scripts were created to facilitate this process. A Jupyter notebook was used for the tokenization, filtering of music files and handling of errors.

6.1.3.2 MIDI to MusicXML script

Various methods were tested to find the most optimal and effective method of converting MIDI files to MusicXML. Music21 [37] was a possible package to be used for this purpose as a common toolkit for understanding and processing music. However, the fidelity of Music21's conversion was not as

effective as Musescore's save file as MusicXML function. As such, a python script was used to utilize Musescore and repeatedly open each MIDI file and save it as a MusicXML file. This process would take approximately 2.5 hours for the 5000 files.

6.1.3.3 Edited CLaMP script

The original CLaMP model could only be used for one file at a single time. The script had to be edited to iterate efficiently through all the files in the dataset to generate the output text and save the output in a .txt file.

The python prompt and instructions to generate the top 30 text descriptors is included in the Appendix 11.1 and the script runs at an average rate of 0.354 seconds per file.

6.1.3.4 Transpose dataset script

As mentioned in Section 6.1.3.4, this script was used to transpose the MIDI dataset to a standardized C major and its relative minor. This script was run on all the MIDI files in the MIDI file folder to generate a transposed MIDI folder. The files were not renamed so that they could have a matching text file.

6.1.4 Train and Validation Dataset Classes

As mentioned in Section 5.5.2.3, a Dataset class is used to store both the input and the target data for the model. However, each MIDI piece has different number of bars, causing each target data tensor to have variable lengths as well. As the default Dataset class was unable to accept data of variable length, customizing the class to meet the project's needs had to be implemented.

The Dataset classes when initialized would obtain both the text and MIDI files and create the Roberta text tokenizer and the Octuple MIDI tokenizer. Each time a new item was needed, the text file would be read, processed, and tokenized by Roberta. On the other hand, the MIDI file would be created,

tokenized by Octuple, and an End of Sequence token would be appended to the end of the sequence. Both these processed tensors would be returned subsequently to the DataLoader.

To conduct both training and validation, 90% of the dataset would be used for training and 10% for validation of the trained data. The dataset classes were separately defined as Train_Dataset and Val_Dataset for these 2 purposes.

6.1.5 DataLoader

As mentioned in Section 5.5.2.4, a DataLoader is used to load and process data accordingly. For both the Train_Dataset and the Val_Dataset, the data was loaded by batch size, with shuffling and drop last features. There was also a collate_fn to pad the text and music values according to the longest sequence in the batch with zeroes. This allows for uniform sizes between the tensors in a batch which is necessary for deep learning.

6.2 Training Implementation

6.2.1 Training Setup

Before the training section could be run, setup code and time was needed. In this section, it was important to set up all the necessary train, validation, and test dataset classes, the DataLoader and the MusicTransformer class efficiently. A long setup time would slow the project significantly as the training process had to be repeated multiple times for modular improvements. This setup would also be costly when running on a cloud GPU with cost per hourly usage considerations.

As the Octuple tokenizer has a variable class length for the Bar parameter as described in Section 5.3.2, it had to tokenize the MIDI file with the highest number of bars during the setup time. If the tokenizer started off the training process otherwise, there would be errors with the embedding and accuracy calculation of the model. The setup initially tokenized all sequences at the start, but this was edited to only tokenize the MIDI file with the greatest number of bars for greater efficiency. This led to the entire setup time to be reduced to just 2.16 seconds on a local 4 RTX2080 Ti machine, which improves the efficiency of training, and led to cost savings when training on a cloud GPU.

6.2.2 Model Architecture Implementation

While Section 5.4 describes the theoretical design of the various layers in the architecture, this section would detail the specific parameter and design choices made for the music transformer.

6.2.2.1 Transformer Builder Parameters

As introduced in Section 5.5.3, the Fast Transformer's Transformer Encoder Builders was used to initialize the transformer. Parameters involved in the transformer builder included n_layers, n_heads, query_dimensions, value_dimensions, feed_forward_dimensions, activation, dropout and attention_type. The values chosen to run the model on 4 RTX2080 Ti machines are described in Table 6.3.

Final Music Transformer Builder	
Parameter	**Variable**
Layers	6
Heads	6
Query Dimensions	768
Value Dimensions	128
Feed Forward Dimensions	768
Activation	Gelu
Dropout	0.1
Attention Type	causal-linear

Table 6.3: Music Transformer Builder Parameters

N_layers specify the amount of transformer layers in the model. Increasing this parameter would allow the model to better capture complex relationships but would add to the model's complexity and potentially lead to overfitting. Heads refer to the attention heads in the multi-head attention mechanism of the transformer. More attention heads allow for richer attention calculations to encode more relationships for each output [38]. The query and value dimension parameters determines the dimensionality of query and value vectors, which are used for the computation of attention scores. Activation refers to the activation function used in the network, while dropout is a regularization method to avoid overfitting. Feed Forward Dimensions determines the dimensions of the hidden layer and attention type determines the attention mechanism used in the layers.

The choice of layers and heads were limited by the memory capacity of the machine used. Each of the RTX2080 Ti GPUs had a usable memory of 11264 MiB which was maximized with this configuration. Increasing the layers and heads would further improve performance should the GPU have sufficient RAM to conduct the training. 768 is also a common dimension for many transformer-based models, which was adopted in this model. Lastly, the causal-linear attention type was also used

instead of full attention. Similarly to compound word transformer, causal-linear attention was needed especially for the triangular causal mask implemented in the architecture [14].

The model was also tested on a cloud RTX A60000 with 48GB VRAM. Where layers and heads were increased to 10 and 12 respectively, with value dimensions dropping to a 64 to accommodate the additional heads. This configuration was the maximum the RTX could handle without having out of memory errors on a 10-epoch training cycle.

6.2.2.2 Data Parallelization

As the workstation had access to 4 RTX2080 Ti GPUs, data parallelization was used to distribute the workload over all 4 GPUs. This was done to increase efficiency, allowing for a more complex model to be used with a higher batch size as the batches can be distributed to each of the GPUs. The gradients are then calculated independently before being aggregated to update the model's parameters. Consequently, each epoch would also have fewer iterations due to the increased batch size. For the configuration on the RTX A60000, data parallelization was not used as only one GPU was rented.

6.2.2.3 Batch Size

Due to the memory limitations, the batch size used was 4 which led to 1 batch per GPU on the RTX2080 Ti. On the RTX A60000, the batch size used was 2 on the single machine, as any further would induce out of memory errors for the GPU. As an overview of the Final model's values, the configurations in in Table 6.4 was used.

Music Transformer Parameters	
Parameter	**Variable**
Batch size	4
Num GPUs	4

Table 6.4: Batch size and num GPUs of Final Trained Model

6.2.3 Training Loop Implementation

This subsection talks about the overall structure of the training loop and its implementation while other concepts will be further explored in Section 6.2.4 onwards.

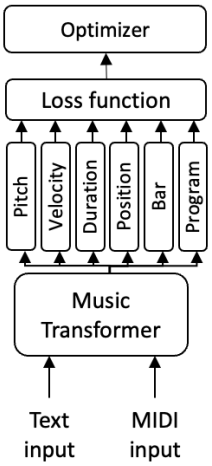

Figure 6.10: Training Loop Process

The Music Transformer was trained for a series of 10 epochs, or training loops. For each epoch, all the train DataLoader text files would be iterated through to be the inputs for the transformer. An overview of the training process can be seen in Figure 6.10. The outputs of the transformer will be generated in the form of probabilities for each of the 6 parameters' classes. Each probability per parameter represents the model's predicted likelihood of that parameter's class being the next output. The loss function will calculate the cross-entropy loss for each parameter, and the average of the 6 parameters will form an average loss.

This process is repeated for all 4000 training MIDI files, over 10 epochs. On average on a RTX2080 Ti, 1 epoch would take an average of 979.88 seconds per epoch which is 16.3 minutes /epoch.

On the RTX A60000, it was an average of 1272.25 seconds per epoch which is equivalent to 21.2 minutes /epoch.

6.2.4 Optimizer and Loss Functions

6.2.4.1 Optimizer and Learning Rate

In the training of a neural network model, an optimizer is used over the stochastic gradient descent to find the optimal weights of the neural network. In this project, an Adam Optimizer was used due to its ease in implementations and its ability to utilize the strengths of both the Adaptive Gradient Algorithm and Root Mean Square Propagation [39]. It uses only a single learning rate for weight updates as a hyperparameter to adjust the rate of model learning. This implementation defines the Adam Optimizer's learning rate at 0.0001 and is multiplied by 0.8 after every epoch. This configuration of the learning rate decay to decrease the learning rate in a linear fashion [40] allows for a smoother convergence towards a solution, and improve the generalization capabilities of the model.

6.2.4.2 Optimizer Functions

To update the model parameters through the computed gradients, optimizer.zero_grad(), optimizer.step() and loss.backward() are the important functions used during each training step. Optimizer.zero_grad() is called at the start of the training iteration to clear the gradients of optimized parameters in the Music Transformer model, to prevent previous iterations from changing the current parameter updates. Loss.backward() is used after the calculation of the loss for each batch, to compute the loss gradient and backpropagate the gradients recursively. Finally, optimizer.step() is used to conduct the parameter update on the gradients through the Adam optimizer algorithm to minimize the loss function. The learning rate and parameters will also be updated accordingly.

6.2.5 Validation

During training, validation is also needed to assess the model's performance during training and is commonly used to detect if overfitting has occurred. This is crucial as the model needs to be able to generalize unseen musical sequences, and validation sets can test the model periodically without updating the model.

To validate the results in this transformer, a validation step will be done after every 8 training steps. The validation DataLoader is used to generate the validation text and MIDI files for the Music Transformer. The corresponding outputs will likewise be permuted and have their loss values calculated by cross entropy loss for each parameter. In addition, 6 different Torchmetric's MulticlassAccuracy metrics are built and used to calculate the accuracy of each parameter. Unlike the training steps, no optimizing and loss.backward() steps are called in the validation process.

A checkpoint of the model will be saved whenever a lower validation loss score is achieved, allowing the last saved checkpoint to be the final trained model with the best validation loss value. Across 10 epochs, the validation loss typically showed substantial improvements at first, with subsequent epochs yielding smaller incremental gains.

6.3 Evaluation

After saving the trained model, the final model can be used in the evaluation process to produce MIDI scores, as illustrated in Figure 6.11. Firstly, the text input would be the input for the music transformer. While the training architecture concatenates the text embedding, start token and the MIDI target embedding as the input tensor, the evaluation architecture only concatenates the text and the start token initially. After Music Transformer produces logits of class probabilities for each parameter, the results go through a Softmax layer with temperature. This Softmax function normalizes the logits and converts them to probabilities accordingly. The temperature parameter is used to further adjust the randomness and distribution of probabilities. A higher temperature leads to a flatter probability distribution with more entropy while a lower temperature brings about a sharper probability distribution with lesser entropy. [41] Next, if a threshold probability p parameter is provided, nucleus sampling is done, while weighted sampling is done otherwise. Nucleus sampling encourages proportional diversity [42] where tokens are selected until the cumulative probability reaches the threshold probability p. On the other hand, weighted sampling simply selects from the probability

distribution which each probability serving as a weight. This process referenced the compound word transformer implementation of sampling methods [14].

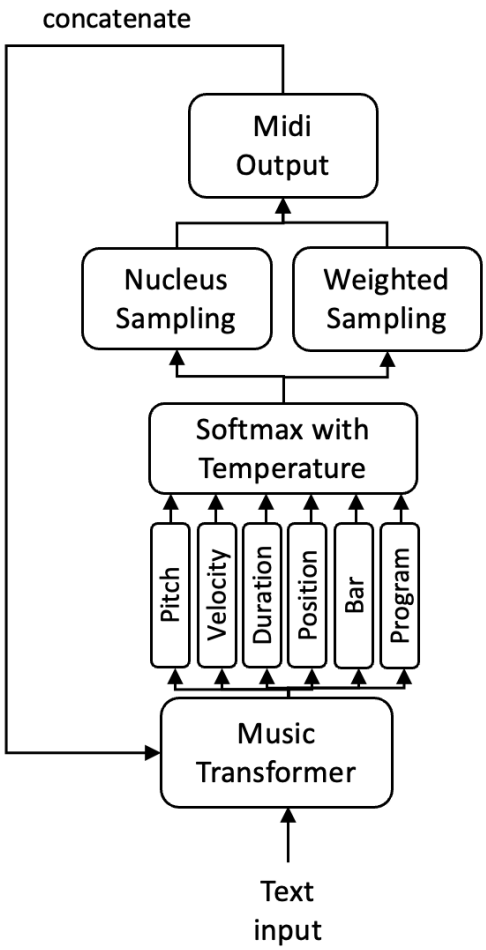

Figure 6.11: Evaluation process

In summary, this evaluation process ensures diversity in the selection of outputs where the model doesn't keep selecting the highest value output. The randomness and entropy in the model facilitate the model's creativity in composing music.

6.4 Hyperparameter Testing

As mentioned in Section 6.3, hyperparameters p and t can be changed for each parameter to change the entropy of the outputs. There were two methods used in this project to find the optimal hyperparameters for a model and they will be discussed below.

6.4.1.1 Automated Hyperparameter testing

To automate the finding of optimal parameters, the test dataset can be used to test each combination of hyperparameters. The accuracy of the hyperparameters can be tested by comparing the generated music against the test dataset MIDI file to check the accuracy and loss values. However, this can be a very computationally expensive process. In a brute force approach where every hyper parameter combination is tested, if 5 values are used per hyper parameter, this would be 5^{12} combinations for each MIDI file in the test dataset. For all 1,000 files in the test dataset, this would be 244 billion music generations to go through the dataset.

To speed up the hyper parameter testing, an adapted process was used where only one octuple parameter was varied at a time. Each of the parameters were tested this way on a much smaller subset of the dataset to conduct the tests quicker. Even though this process would not guarantee an optimal set of hyper parameters, it would yield a better perspective of how the hyper parameters impact the results.

Subsequent sections highlight the impact of a slight change in hyperparameters. This can be seen in Figure 7.8 and Figure 7. 9 in Section 7.2.1 where there can be huge changes in outcomes with slight hyperparameter tweaks.

6.4.1.2 Manual Hyperparameter testing

While the automated testing allows for faster generation and comparison of results, it would still be difficult to qualitatively evaluate the results just from the loss and accuracy values. Elements such as musicality, tonality, chord sequences and more are hard to define purely from just accuracy values. In addition, since only a few hyperparameter values could be used in the automated testing process, finer adjustments needed to be done to decide on a preferred value.

In this approach, the octuple list was printed out after evaluation, and several heuristics were used to compare the efficacy of the results from just the Octuples. The more significant ones are listed below in Table 6.5. These heuristics offer helpful guides to determine if just from the octuple, the music is likely to be an acceptable MIDI file.

Heuristic	**Explanation**
The Pitch values are not identical	An early problem was that the evaluation output would always generate outputs with low variance in pitch values. Constantly repeating pitch values would not be a musical piece.
The Bar values are not identical	With a repetitive Bar value, this will mean that the piece is only 1 bar long.
The Bar values should be gradually increasing in a stepwise fashion	To create a continuous piece, the Bar values should increase in a smooth and gradual progression, with multiple repeats and an increase of just 1 value each time. This would prevent the music from having too much empty space in the middle of the score.
The Program values are not random	An overly random Program values would result in too many tracks from different instruments making it more likely to be noisy. An ideal program output should vary between being identical (single track) or one main instrument and other instruments.

Table 6.5: Significant heuristics for Hyperparameter testing

Next, if the octuple result looked satisfactory, the MIDI file was created, listened to, and evaluated qualitatively. The adherence of the music output to the initial textual prompt was also observed. In this process, 189 MIDI files were created and manually checked during this hyper parameter testing, with about an additional 100 generations at the octuple level.

It should be noted that while the global optima can’t be reached by this method, achieving a local optimum was sufficient for the purposes of this project. This whole process is also only specific to each trained version of the model and had to be repeated when a new version of the model was trained.

6.4.1.3 Overview of Hyperparameter testing

From this process, the hyperparameters chosen for the trained model is listed as follows in Table 6.6.

Final Music Transformer Hyperparameters		
Parameter	**Temperature**	**Probability**
Pitch	1.2	0.5
Velocity	2	None
Duration	1.25	0.5
Position	1.5	0.5
Bar	0.6	None
Program	1.25	0.45

Table 6.6: Selected hyperparameters for Music Transformer

The hyperparameter for pitch yielded results that were consistently non identical unlike past iterations of the model. It had a good balance of both creative pitches but wasn't overly random as well compared to past model outputs. The velocity parameter could afford more variations to simulate human-like dynamics and intonations. Duration and Position were calibrated to allow more variation without being overly fixed or rigid. It was observed that a lower entropy on these values made the music sound more robotic and repetitive which wasn't ideal. Bar needed less randomness as it had to follow a gradual increase as described in the heuristic. However, too low a temperature often yielded identical bar values for the entire piece. Program was also calibrated such that the output would sometimes be identical throughout, which resulted in a single-track instrument, or vary to create multi track music. This variation was ideal to create more diverse music compared to taking the highest probability result which only yielded piano programs.

7) Previous Implementations and Results

In the process of making the current Music Transformer implementation, there were many other architectures and designs that were attempted. However, the results from these implementations were unsuccessful. The architectural designs and results will be explained in this section.

7.1 Stepwise Music Transformer

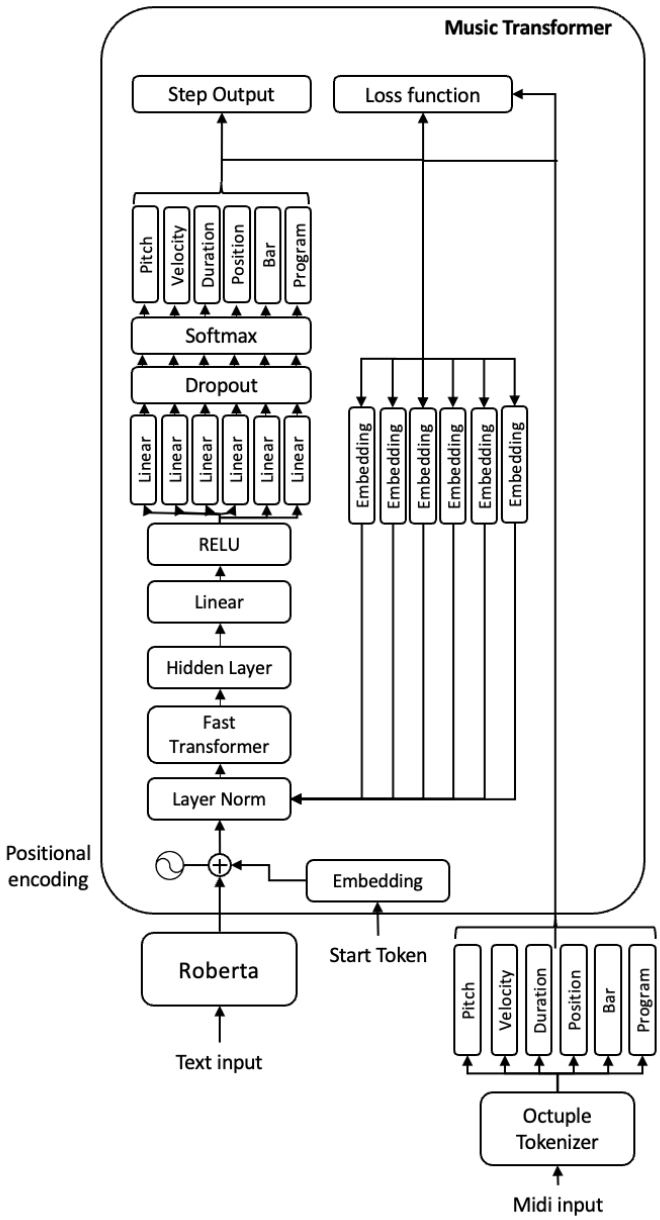

Figure 7.1: Stepwise Music Transformer Architecture Diagram

7.1.1 Overview

This version of MusicTransformer was the main design for the transformer before the entire architecture was redesigned. The architecture diagram can be seen in Figure 7.1. In this design, the text input along with a start token was passed through a forward step to generate the first octuple output. At each forward step, the cross-entropy loss function will be called to calculate the loss between the predicted output and the target MIDI data. Through teacher forcing, either the previous outputs or the target octuple will be used as the embedded inputs to the fast transformer. This process continues until the length of the MIDI target is reached, by which the generated music sequence would be completed.

Unlike the current MusicTransformer which generates the entire musical sequence before calculating the loss, this version calculated the loss at each octuple. This architecture also did not use triangular causal masking due to the stepwise implementation. However, other layers and techniques that were used in this implementation will be described below.

This architecture took significantly longer to run, with each batch taking a rough average of 114 seconds, which when run on a 4000-size train dataset would take 31.67 hours just for 1 epoch even after data parallelization. This is astronomically longer compared to the new model's 16.3 mins per epoch. A table to compare the differences between the two implementations can be found in Table 7.1.

Differences		
Areas	**Current MusicTransformer**	**Stepwise MusicTransformer**
Architecture design	Predicts	Predicts step output
Training time per epoch	16.3 mins / epoch	31.67 hours / epoch
Performance	Varied output, coherence in melodies	Always predicted same output

Table 7.1: Differences between the 2 transformer designs

7.1.2 RELU

RELU stands for Rectified Linear Unit and was applied to activate the hidden layers of the transformer to learn complex relationships. This was a simple function to replace negative values in the input tensor with zero to introduce nonlinearity to the model to better learn complex patterns and representations in the data.

7.1.3 Softmax

Softmax was also used as an activation layer to convert the logits after the dropout layer to probability distributions for classification prediction. This would be used to normalize the results to ensure that all probabilities sum up to 1, before prediction occurs.

This wasn't used in the current implementation because it was later discovered that performance improved significantly upon removal of the Softmax layer. This is so as the CrossEntropyLoss already includes a Softmax operation internally and only expects raw logits as the input.

7.1.4 Loss calculation

The Loss was chosen to be calculated at each octuple step instead of at the end mainly due to memory constraints. With each output being calculated in a stepwise fashion, it would be too memory demanding as the stochastic gradients would need to be calculated and stored for each octuple in the model. This was infeasible for longer sequences without incremental computation to reduce memory usage.

It is hypothesized that the backpropagation of losses at each step could have led to a decrease in the performance as the rest of the context might not have been considered in this way.

7.1.5 Teacher Forcing

One technique that was used in this architecture was teacher forcing for sequence generation. Teacher forcing is implemented by a probability, which would alternate between providing the neural network with the target sequence as the input in each step instead of the model's output [43]. This is a method to guide the model during training, which has large benefits in improving convergence time and reducing the error propagation. Through teacher forcing, early mistakes by the model wouldn't be repeated and exacerbated through the rest of the sequence generation. Nevertheless, it risks making the model over dependent on the target data and struggle with generalization. As such, the teacher forcing probability would decrease gradually over the training process, so that the model can rely on its predictions more.

7.1.6 Progressive Results

7.1.6.1 Results of Initial Architecture Design

Before teacher forcing was implemented, the model only generated one note each time at the end of the piece as shown in Figure 7.2. This was regardless of the prompt and after various attempts. It was evident that changes had to be made and teacher forcing was then implemented to improve the training process.

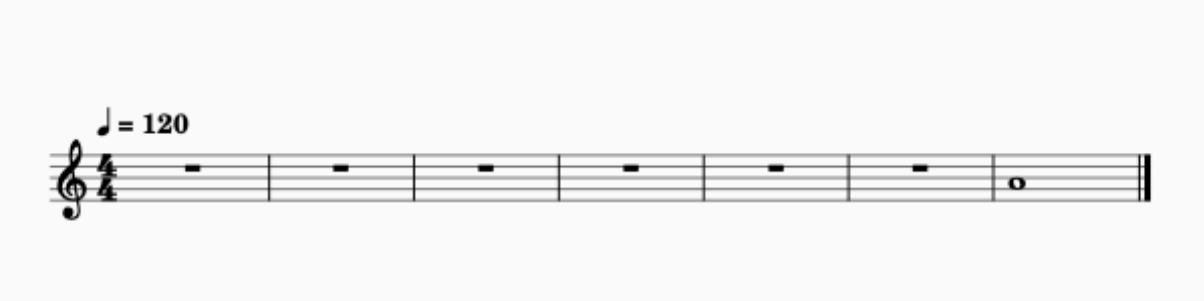

Figure 7.2: Initial design MIDI outputs

7.1.6.2 Results after teacher forcing

After teacher forcing was implemented, the model generated more results in the evaluation section. However, it created very repetitive results and converged quickly to certain pitches. An example of the

MIDI results can be seen in Figure 7.3, and the mp3 file can be found at: https://tinyurl.com/FypInitialTeacherForcing. The trained model converged to G4 and A4 for pitch very frequently as that was the predicted output that reduced losses. This happened in the training after just 55 batches in the then 90 MIDI trained dataset. In addition, the bar attribute was constant through most of the output which led to fewer notes appearing on the MIDI file. The similar minims generated on the first beat each bar also points to similar values in both position and duration parameters. Likewise, changing the prompt did little to change the outcome, where apart for a few anomaly pitches, the rest would follow a similar pattern.

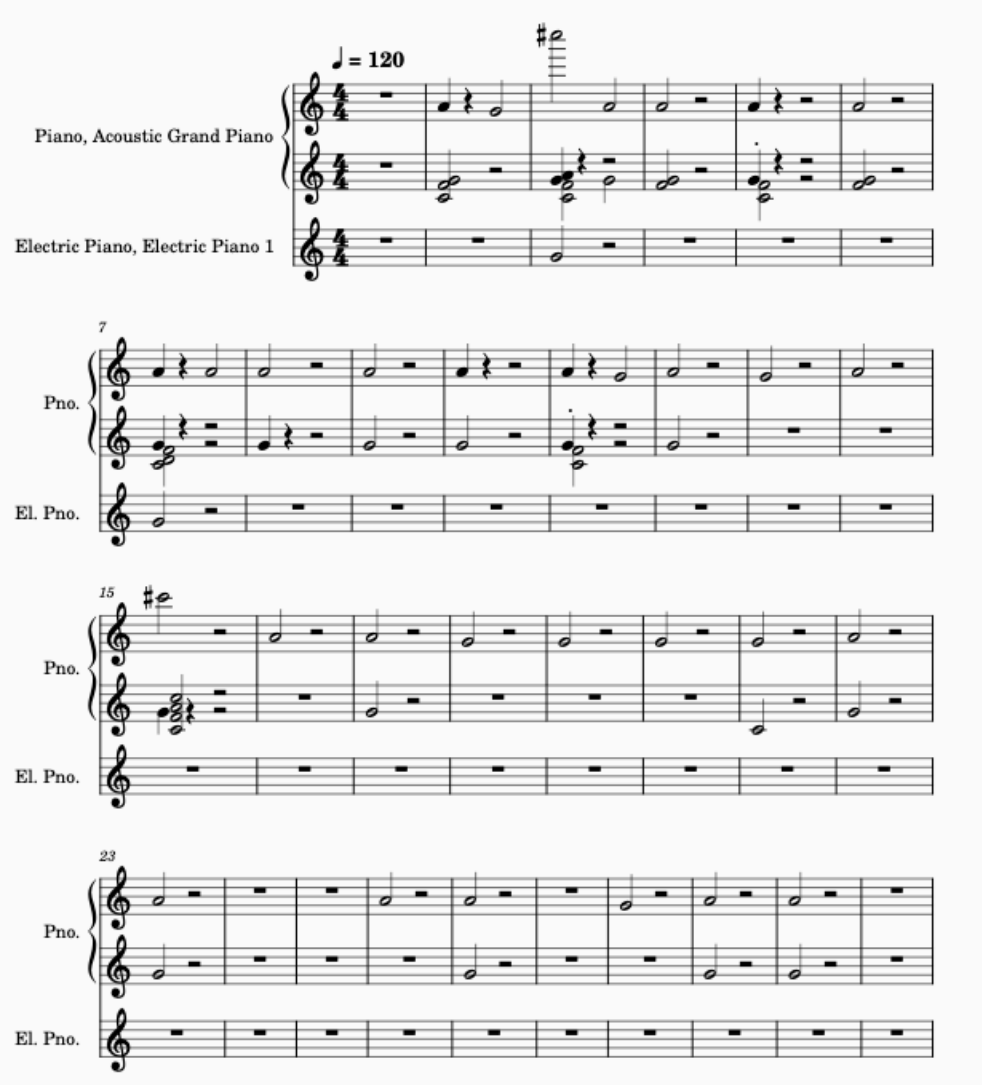

Figure 7.3: MIDI output after teacher forcing

7.1.6.3 Results after Additional Layers and Experimentation

It was clear from the training and evaluation results that the model converged too quickly and overfitted to the values. As such, various combinations of techniques were used to achieve better results. New layers such as layer normalization was used, different number of heads, dropout values, learning rates were all attempted in this process. These techniques allowed for more varied outputs and seemed to address the previous issue of overfitting to 2 main pitch values. However, the pitch results became more random rather than cohesive. As can be seen in Figure 7.4, there are many high notes, and multiple notes on the same beat and position, especially the notes in Bar 19. The MP3 file for this MIDI piece can be found at: https://tinyurl.com/FypInitialLayerNorm.

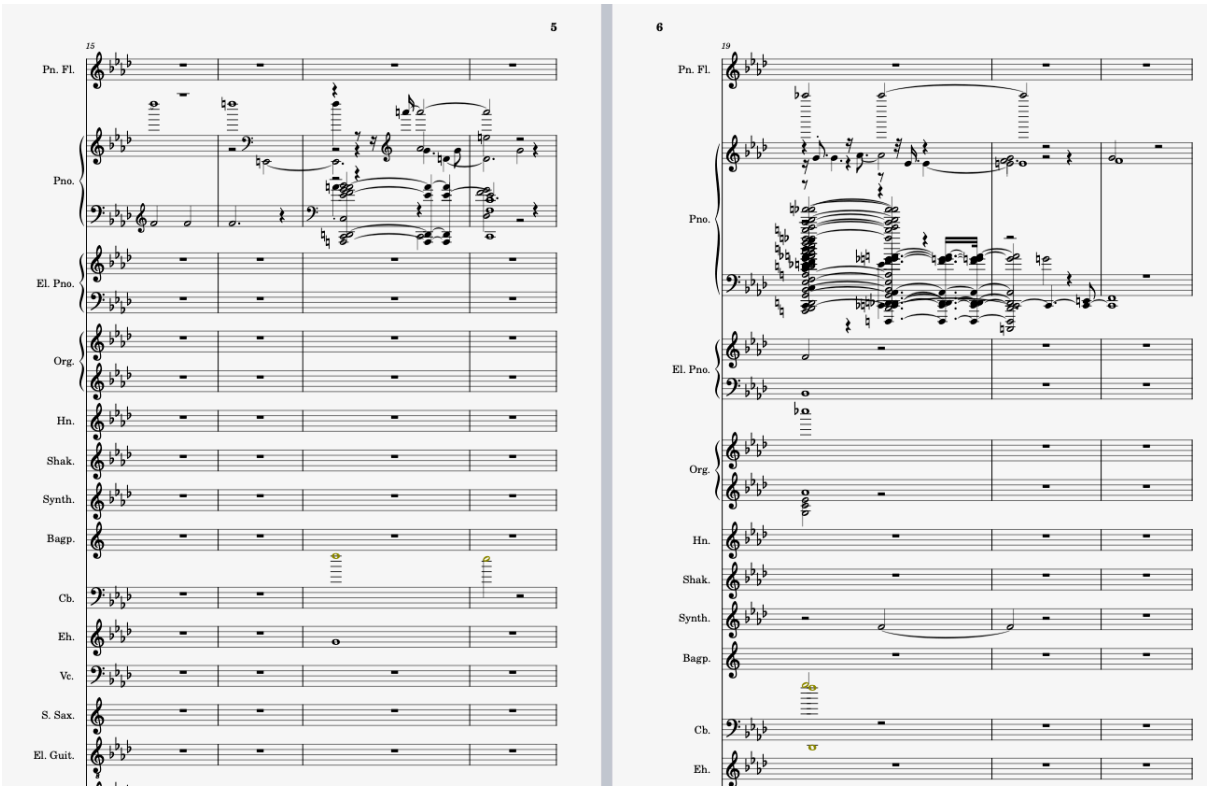

Figure 7.4: MIDI output with Additional Layers

Additionally, while there are more varied notes, the same core notes of G4 and A4 still come up quite often in the piece as can be seen in an Figure 7.5, which is an earlier section of the same piece. For example, the clusters of G♭4, G4, F4, E♭4 in the red circle of Bar 7 would sound very clashing. It is also a result of the model still overfitting to specific values.

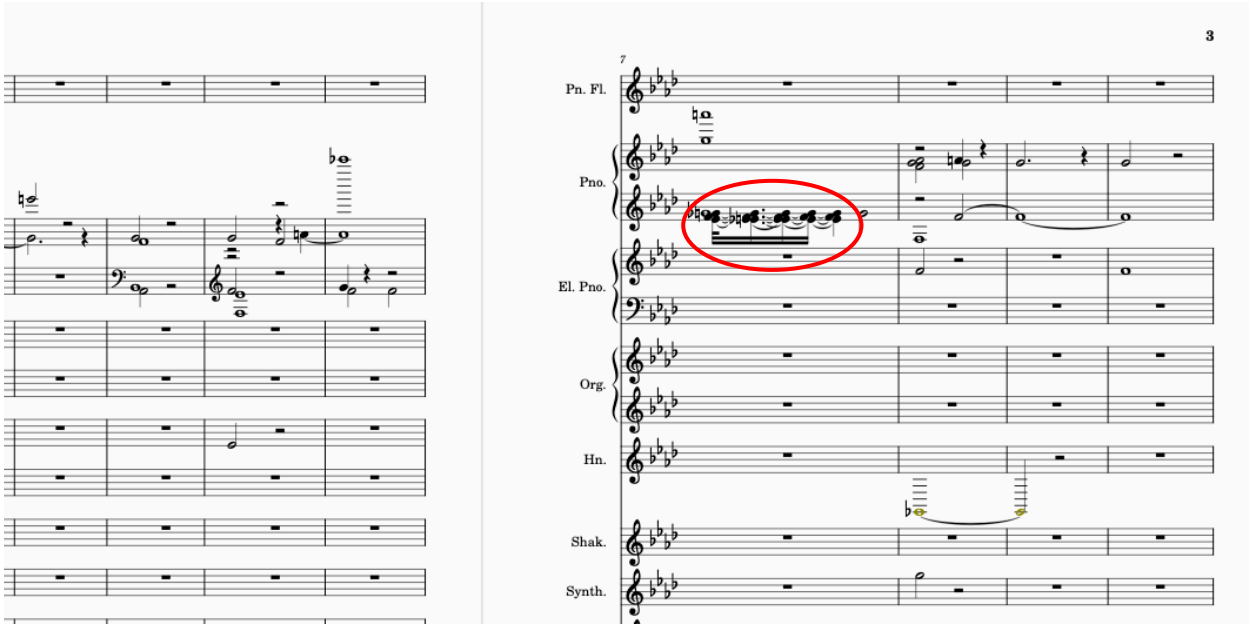

Figure 7.5: MIDI output with Additional Layers

7.1.6.4 Results after GAN on old model

To solve this, we tried to use different ways of calculating loss but that wasn't very effective. We also considered using a Generative Adversarial Nets (GAN), which would help to discern between generated MIDI files and target MIDI files, but likewise that implementation did not improve the results significantly. An example of the similar outputs can be seen in Figure 7.6, and the audio file can be found at https://tinyurl.com/FypInitialGAN.

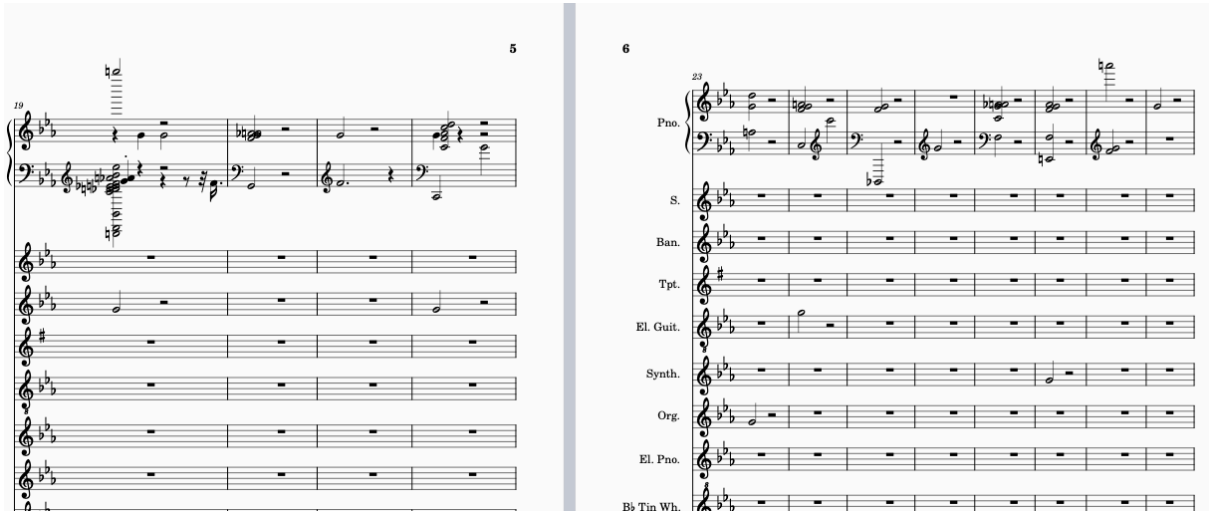

Figure 7.6: MIDI output with GAN implementation

7.2 Previous Results of Current Architecture

7.2.1 Initial results of Final Transformer Architecture

Before the final trained model was obtained, some significant changes had to be made to the current architecture to achieve results. This section will document the progress that was made.

When the model architecture was first used, it still had Softmax layers for the training loop segment, the dataset was not transposed and only had 1000 files. In addition, the method of evaluation was different and did not use hyperparameters or Softmax with temperature. As such, the initial generation is shown in Figure 7.7, where the most probable result was always chosen. While this led to quite tonal and pleasant music, it was more repetitive and lacked creativity as it was largely similar chords that were played. The audio file for the MIDI file can be found at: https://tinyurl.com/FypInitial.

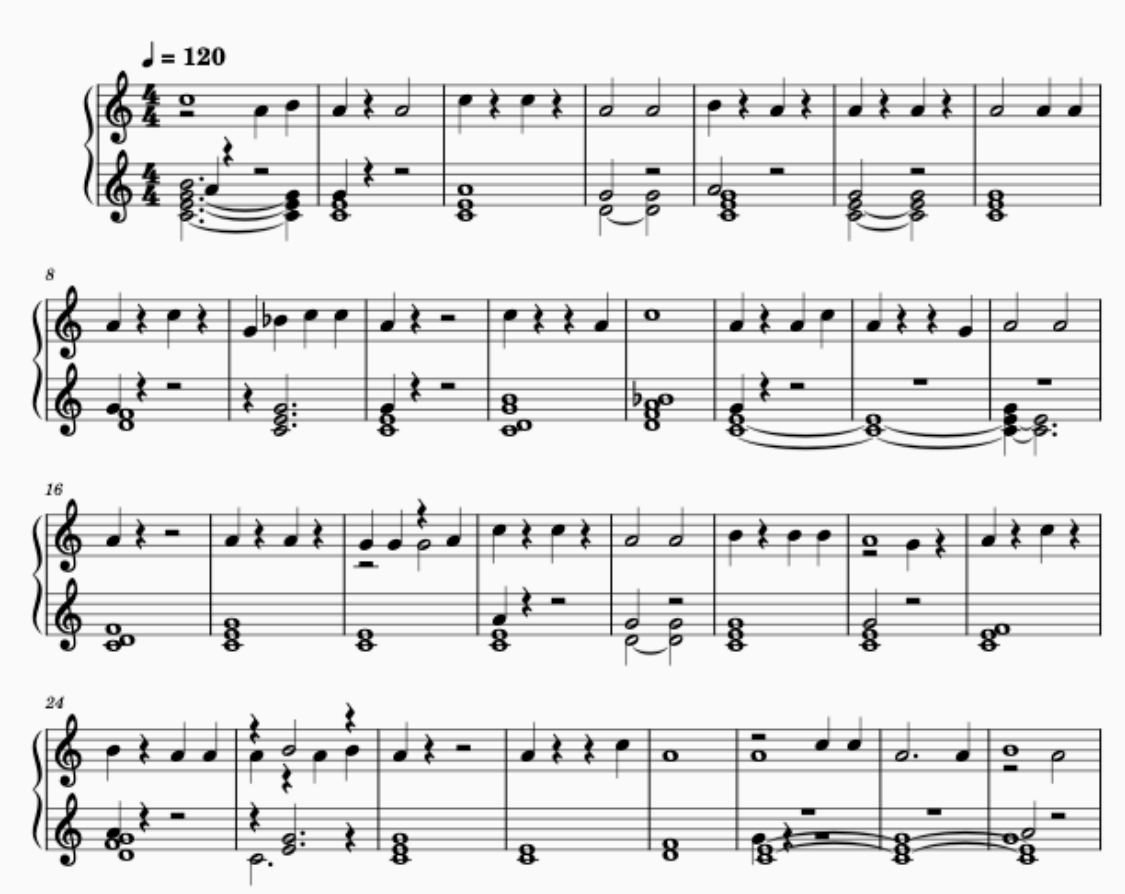

Figure 7.7: Initial Generated Music before changes

To generate more diverse results, the evaluation process was revamped, and included the Softmax with temperature and Sampling functions as addressed in Section 6.3. As a result, the model was able to generate varied results through tuning the hyperparameters of t and p. However, it was quite challenging to conduct this process as the evaluated results could alternate between random music and repetitive music quickly with slight changes to the hyperparameters. The methods of automatic hyper tuning as described in Section 6.4.1.1 and the manual hyper tuning of Section 6.4.1.2 were both used, to explore the impacts of changing each parameter accordingly.

For example, with the prompt: “Classical love song, upbeat tempo and complex harmony”, and hyper parameters as shown in Table 7.2 , the music file as shown in Figure 7.8 was generated. The audio file for the generated MIDI file can be found at: https://tinyurl.com/FypSoftmaxT23. This indeed had a complex harmony, although it generated very big and clashing chords as in bar 11.

Figure 7.8 Hyper Parameters		
Parameter	**Temperature**	**Probability**
Pitch	2.3	0.9
Velocity	3	None
Duration	2	0.9
Position	1	None
Bar	1	None
Program	0.5	None

Table 7.2: Hyperparameters for Figure 7.8

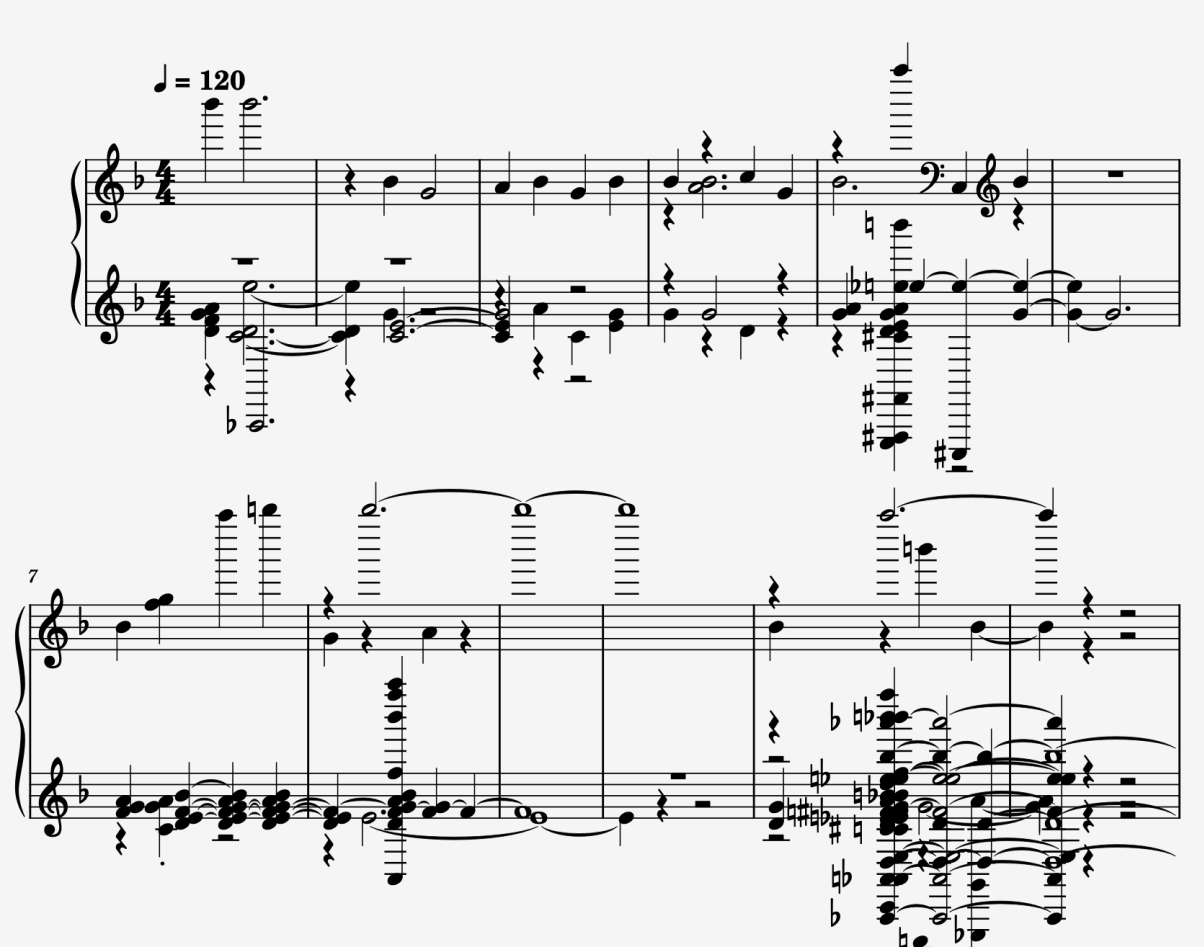

Figure 7.8: Prompt for "Classical love song" Generated Music after Softmax with Temperature 2.3

However, when just the temperature of pitch was changed to 2.2 instead of 2.3, the model would generate very repetitive music as shown in Figure 7.8 with the A4 notes being repeated often. The audio file for the generated MIDI file can be found at: https://tinyurl.com/FypSoftmaxT22.

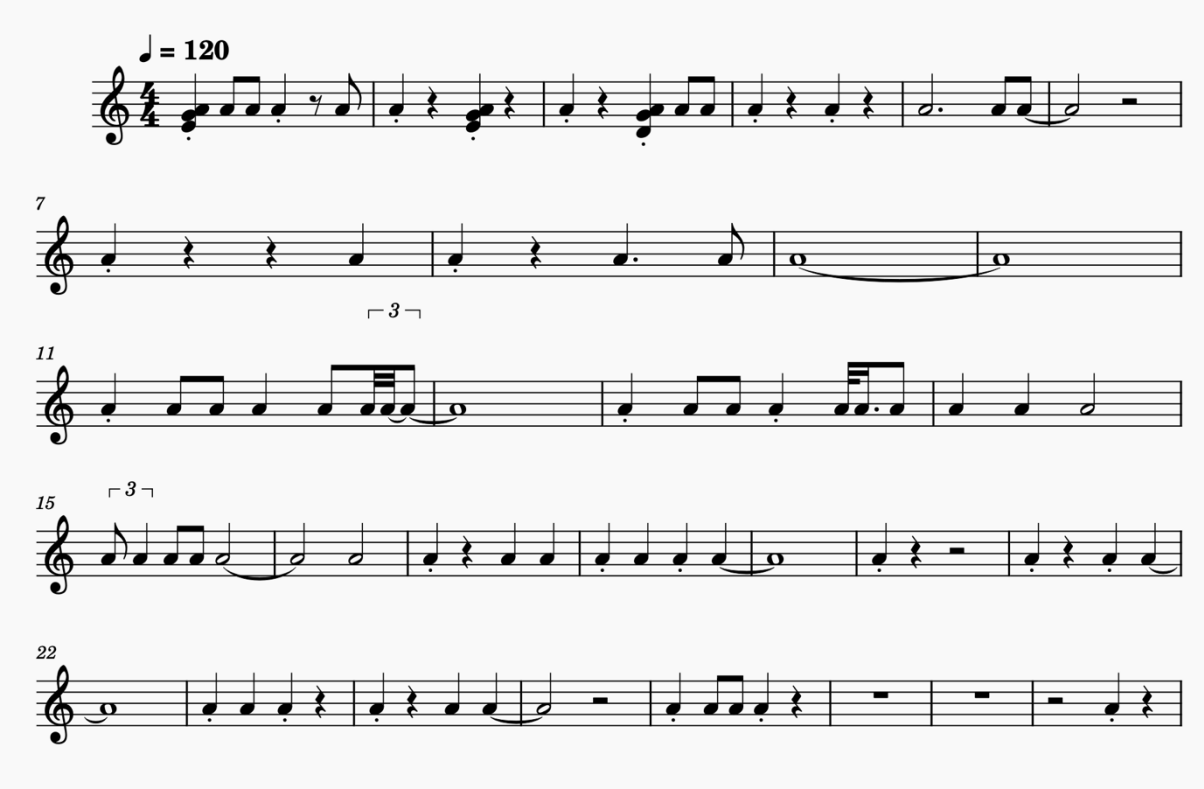

Figure 7. 9: Prompt for "Classical love song" Generated Music with pitch t = 2.2

As such, this made it challenging to hyper tune the parameters when a small change in parameters could result in quite different outputs. This volatility in output where the model alternates between repetitive music and random music was not ideal, as an in-between music was desired. While some creativity is needed and can be expressed through randomness, the random outputs such as in Figure 7.8 can be quite extreme and sound like noise to most listeners. On the other hand, while stability and repeated themes are important fundamentals in music, the over repetition of a single note such as in Figure 7. 9 would sound dull and plain. It was also likewise a sign of over fitting to a single value and had to be addressed.

7.2.2 Overview of Challenges

To summarize the common challenges, it was clear that there were a couple of problems that usually came out in this MusicTransformer model:

1. Repetition: The music generated could often be repetitive to a fault, resulting in uninspired music even though it might sound more tonal. This can be a result of overfitting or of taking the most probable result all the time.
2. Randomness: On the other hand, the music could also turn to the other extreme of being too random, where there aren't identifiable patterns in the music.
3. Little links to the prompt: The above two problems further manifest their issues by resulting in music that have minimal links to the prompt. This would defeat the purpose of a text-to-MIDI AI model, if the text has little impact on the music.

7.2.3 Final changes made to Music Transformer

Before the final trained model was achieved, there were 3 major changes that improved the outputs significantly:

1. The increase of the dataset from 1010 files in the WMT dataset to 5000 files with 3 combined datasets (datasets explained more in Section 6.1.1.1 and Section 6.1.1.2).
2. The removal of the Softmax layer in the training loop (training loop in Section 6.2.3).
3. Transposition of all music to C major or A minor (explained more in Section 6.1.1.4).
4. Final hyperparameter tuning of model (explained more in Section 6.4).

7.2.3.1 Increase in Dataset size.

For the increase of dataset from 1010 files to 5000 files, this was done to combat the overfitting issue. One potential reason for overfitting could be a small dataset. As such, the usage of various datasets and more varied MIDI files was used for this purpose. More information on this can be found in Section 6.1.1.1 and Section 6.1.1.2.

7.2.3.2 Removal of Softmax Layer

The removal of the Softmax layer in the training loop improved performance greatly also. This was needed because the Cross Entropy Loss already uses an internal Softmax layer and hence an explicit Softmax layer was not required.

7.2.3.3 Transposition of all music

Lastly, the transposition of all music to C major was useful to standardize the key signatures for easier training of pitches. This was not done earlier as it was hoped that the model would learn and be able to generalize for a variety of key signatures. However, it was likely that the different key signatures affected the pitch levels and made it harder to draw patterns between sequences.

All these implementations helped to combat the problems mentioned in 7.2.2 regarding overfitting, overly random values. It also assisted the model to draw better relationships between the music.

7.2.4 Final hyperparameter tuning of chosen model

7.2.4.1 Pre Hyperparameter-Tuning

Before hyperparameter tuning and using the hyperparameters of the previously trained model, the results were very random and messy. An example of the output can be seen in Figure 7.10. This showed the impact that hyperparameter tuning has, that even if the model has been trained effectively, it can still create very random and messy music before hyperparameter-tuning. An audio of this file can be found at: https://tinyurl.com/FypPreHyperParam.

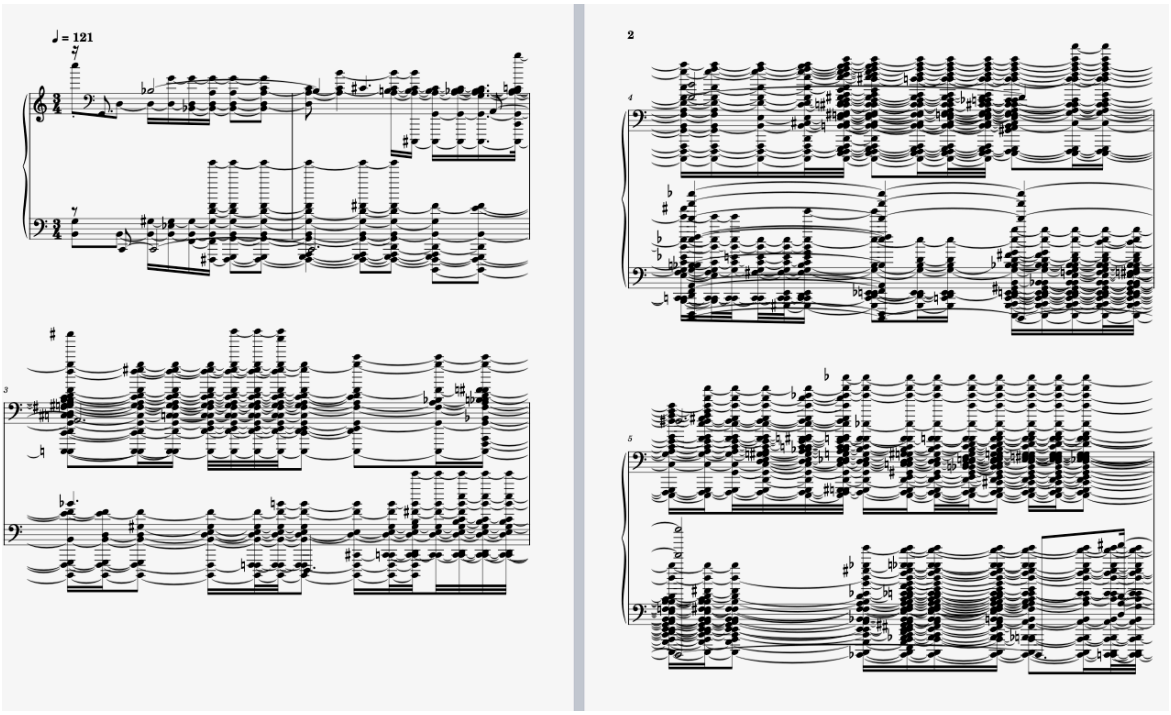

Figure 7.10 MIDI file before Hyperparameter tuning

7.2.4.2 Final Model Hyperparameter-Tuning

After all these changes were made, hyperparameter tuning of the values had to be done. Because an entire hyperparameter process had been done before, it was decided that manual hyperparameter testing would be more advantageous than automated hyperparameter tuning. This is so as the automated version only checked loss values, would take a very long time to check, and could only give general values. It was likely that finer adjustments needed to be made and as such manual hyperparameter testing allowed for that, to read the octuple outputs and check the MIDI files that were generated. 189 MIDI files were looked at and even more Octuples were generated in this process to find an ideal setting for the parameters as shown in Table 6.6.

In determining what hyperparameter values should be chosen, a value that allows the model to alternate between something more structured and something more creative was chosen. This allows more freedom of the model to express various prompts, without being overly rigid or overly random. For example, more flexibility in the position and duration parameters lead to more dynamic outputs, which sounds more human-like rather than rigid and fixed notes at each beat in a bar.

7.2.5 Cloud training attempt

In addition to training the model on 4 RTX2080 Ti GPUs, where each had 11GB of GDDR6 memory, the model was also trained on RTX A60000 which had 48GB of GDDR6 memory. This was done to test the differences and to see which machine trained a better model.

It should also be noted that each training process can yield different results and that makes it hard to directly compare the loss values and accuracy. The running time of the Cloud RTXA6000 model took 8 Epochs, and this was the best validation loss achieved after Batch 1961/2000 was run. The accuracy results can be seen in Table 7.3.

Cloud RTXA6000 Accuracy and Loss	
Parameter	**Accuracy**
Pitch	0.1743
Velocity	0.1372
Duration	0.1127
Position	0.3095
Bar	0.5415
Program	0.2325
Model Best Validation Loss	1.6443

Table 7.3: Cloud RTXA6000 Trained Model Results

Even though the accuracy results of this cloud RTXA6000 were quite comparable to the 4 RTX2080 Ti machine, the qualitative results of the cloud trained RTXA6000 was suboptimal. Regardless of the hyperparameters chosen for Bar and Program, the model was unable to achieve a non-random result for either of them. This can be seen in the octuple output in Figure 7.11, where the least random hyperparameters of t= 0.05 was used for both Bar and Program in evaluation. The Bar and Program are the second last and the last column respectively, and the Bar parameter especially has too random jumps between values. As seen in Section 6.4.1.2, this is an especially important heuristic that the Bar values are gradually increasing instead of them being random. Should they be random, this means that

the output is not considering the music sequentially which defeats the purpose of many pattern formulations in music as well. As such, this model was not used as the final model checkpoint for this project.

```
tensor([[[ 90,   2,  42,  17,  67,  73],
         [ 50,  30,   8,   6,  63, 103],
         [ 64,  25,  59,   0, 266,  16],
         [ 43,  11,  49,  17, 167,  31],
         [ 24,  22,  26,  31,  43,  15],
         [ 29,   9,  10,  34, 170,  31],
         [ 60,  15,   5,  30, 170,  15],
         [ 41,  31,  34,  33, 167,  15],
         [ 31,  27,  48,   4, 170,  31],
         [ 48,  28,  11,   6, 170,   2],
         [ 81,  13,   9,   1,   2,  96],
         [ 81,  32,   8,   6, 152,  15],
         [ 31,  21,   8,  18, 169,  15],
         [  6,  23,   8,  11, 185,  31],
         [ 56,  12,   1,   0, 101,  96],
         [  5,  31,   4,  21, 257,  95],
         [ 26,   7,  23,   8, 320,  15],
         [ 60,   4,  45,  15, 221,  29],
         [  5,   8,  35,  23, 311,  96],
         [ 75,   7,  42,   3, 104,  19],
         [ 67,  30,  19,  17,  56,  80],
         [ 52,  35,  62,   8, 131,  31],
         [ 15,   9,  59,  27,  37,  96],
         [ 19,   5,  28,  21, 104,  16],
         [  9,  33,  11,  30,   2,  80],
         [ 57,   9,  23,  23, 167,  96],
         [  1,   4,  43,  34, 221,  23],
         [ 54,   1,  11,   8,  41,  80],
         [  0,   7,  23,  27,   2,  73],
         [ 59,   3,   6,   1,  33,  96],
         [ 23,   2,  41,  20,  56,  68],
         [ 45,  10,  57,  17, 305,  45],
         [  1,   7,  40,  23,  57,  62],
         [ 56,  14,  12,  13, 152, 114],
         [ 36,  24,  36,   8,  93,  31],
         [ 36,   7,  50,  21,  60,  45],
         [ 51,  27,  45,  21, 169,  96],
```

Figure 7.11: Least Random RTXA6000 output

Nevertheless, it is possible that if more RTXA6000 machines were used with data parallelization, a better model could have been achieved with a larger batch size. It is also possible for the same code to have yielded a better trained model. However, it was too inefficient in terms of time and costs to train too many transformers, tune their hyperparameters and test their accuracy accordingly.

8) Results

This section will detail all the various results that were obtained, from examining the MIDI files that were generated to the survey results.

8.1 Quantitative Results of Trained Model

For the quantitative results of the chosen architecture using 4 RTX2080 Ti machines, the chosen parameters led to a trained model of accuracy and loss as detailed in Table 8.1. This validation model was achieved in the 10th Epoch, after the 689/1000 batch.

Final Model Accuracy and Loss	
Parameter	**Accuracy**
Pitch	0.1181
Velocity	0.1723
Duration	0.2466
Position	0.3202
Bar	0.4707
Program	0.1738
Model Best Validation Loss	1.6396

Table 8.1: Final Trained Model Accuracy and Loss

From the results, the Bar parameter performs the best among the other parameters in terms of accuracy. This should be evident as Bar in all the MIDI Octuples will always be strictly ascending 1 value at a time. Nevertheless, it may still be challenging to predict when exactly the Bar value should increase as that differs from piece to piece. Position and duration are the next easiest to predict as they likely have a smaller range of used values for the Octuples. Program is also tricky as the dataset contained files with multiple instruments as well and it would be challenging to understand which instrument should be used. Program also contains the greatest number of classes with 133 class

variables, but a large percentage of the pieces still use piano as the dominant instrument. Lastly Pitch performed the least favorably due to the complexity of the music and the contexts of the various pieces. Compared to other hyperparameters, it also has the second most number of classes with 92 different classes.

This model ultimately has a better validation loss compared to the Cloud RTXA6000 model, although other parameters have differing accuracies. Nevertheless, this trained model was chosen due to the way it was able to address the various challenges of other models.

8.2 Qualitative Analysis of Generated Octuple Outputs

In this section, Octuple outputs files will be evaluated qualitatively and judged mostly by the heuristics that was mentioned in Section 6.4.1.2 and what can be observed just on the Octuple outputs.

To illustrate how this model has more varied outputs, the evaluated octuple output results in Figure 8.1 and Figure 8.2 are outputs from the chosen model checkpoint under the same final hyperparameters in Table 6.6. In this output, each row represents the class value of a single Octuple or note in the MIDI file. On the other hand, each column represents each parameter in the order: Pitch, Velocity, Duration, Position, Bar, Program.

As can be seen from the Bar Parameter in the 5th column, both Octuple outputs have gradually increasing results, with multiple notes in a bar like a normal piece. This satisfies two of the major heuristic in Table 6.5 regarding non identical bar values and gradually increasing bar values.

As for pitch, The Pitch values in the first column are also not identical, with varied changes without too much clustering. This ensures that there is sufficient variation in the pitches that are played. In addition, the spread of the pitch values can be quite different for the two music files created, without

being overly extreme, such as having values near 0 or near 92. The lack of overly extreme values which have appeared in previous music models prevent the music from sounding too jarring.

For Program, the model can effectively alternate between having identical values in Figure 8.1 and varying values in Figure 8.2. This is especially advantageous as this means that the model can generate just single instrument values as well as orchestral pieces, without having to change the hyperparameters. This was impossible to achieve on previous iterations of the design and provides a major advantage in this model checkpoint.

```
tensor([[[ 38,   7,   4,  12,   4,   5],
         [ 38,  23,   7,  20,   4,   5],
         [ 42,  22,   7,  24,   4,   5],
         [ 42,  10,   7,  28,   4,   5],
         [ 43,  11,   7,   4,   4,   5],
         [ 45,  27,   7,   8,   4,   5],
         [ 40,  12,  11,  12,   4,   5],
         [ 43,  10,  11,  12,   4,   5],
         [ 47,  23,  11,  12,   4,   5],
         [ 43,   9,   7,  20,   4,   5],
         [ 38,  31,   7,  20,   4,   5],
         [ 43,  17,   7,  20,   4,   5],
         [ 43,  21,   7,  24,   4,   5],
         [ 45,  16,   7,  28,   4,   5],
         [ 47,  23,   7,  32,   4,   5],
         [ 42,  23,   7,   4,   5,   5],
         [ 42,  23,   7,   8,   5,   5],
         [ 42,  31,   7,  12,   5,   5],
         [ 42,  21,   7,  16,   5,   5],
         [ 45,   9,   7,  16,   5,   5],
         [ 38,  31,   7,  16,   5,   5],
         [ 42,  16,   7,  20,   5,   5],
         [ 45,  15,   7,  24,   5,   5],
         [ 43,  11,   7,  28,   5,   5],
         [ 43,  31,   7,  28,   5,   5],
         [ 40,   8,   7,  28,   5,   5],
         [ 43,  13,   7,  32,   5,   5],
         [ 42,  17,   7,   4,   6,   5],
         [ 42,  14,   7,   8,   6,   5],
         [ 45,  30,   7,  12,   6,   5],
         [ 38,  10,   7,  16,   6,   5],
         [ 42,   9,   7,  20,   6,   5],
         [ 42,  17,   7,  24,   6,   5],
         [ 43,  11,   7,  24,   6,   5],
         [ 43,  22,   7,  24,   6,   5],
         [ 47,  15,   7,  24,   6,   5],
         [ 45,  14,   7,  28,   6,   5],
         [ 43,  22,   7,  32,   6,   5],
         [ 42,  10,   7,  32,   7,   5],
         [ 42,  24,   7,  32,   8,   5],
         [ 43,  11,   7,  32,   8,   5],
         [ 42,  13,   7,   4,   8,   5],
         [ 45,   8,   7,   4,   8,   5],
```

Figure 8.1 Octuple output 1 of Music Transformer

```
tensor([[[ 19,  30,  19,  28,   4,  78],
         [ 21,  19,  35,  32,   4,  67],
         [ 19,   1,  35,  32,   6,  78],
         [ 19,  23,  51,  35,   6,  61],
         [ 19,  19,  11,   7,   7,  51],
         [ 23,  20,   9,   7,   7,  51],
         [ 47,  21,   7,   9,   7,  55],
         [ 21,  16,   9,  12,   7,  45],
         [ 43,  12,   5,  12,   7,  38],
         [ 47,   4,  11,  12,   7,  55],
         [ 47,  12,   5,  19,   7,   5],
         [ 16,   4,   8,  20,   7,  29],
         [ 19,   6,   8,  24,   7,  29],
         [ 21,  17,   9,  28,   7,   4],
         [ 19,  22,  10,  28,   7,  83],
         [ 19,  18,  11,  31,   7,   5],
         [ 26,  28,  11,  31,   8,  37],
         [ 18,  23,   5,   4,   8,  87],
         [ 19,  34,  35,   9,   8,  93],
         [ 14,  23,   8,  13,   8,  37],
         [ 47,  35,   9,  17,   8,  83],
         [ 19,  28,   9,  21,   8,   4],
         [ 21,   3,   9,  21,   8,  95],
         [ 19,  32,   4,  21,   8,   5],
         [ 21,   9,   8,  23,   8,  29],
         [ 21,  34,   5,  28,   8,  83],
         [ 24,  21,  11,  28,   8,   4],
         [ 26,  18,   7,   4,   8,  57],
         [ 24,  27,  11,   4,   8,  57],
         [ 21,  33,   5,   4,   8,  57],
         [ 21,  33,   5,   4,   8,  57],
         [ 24,  31,   4,   4,   8,   5],
         [ 21,  32,   9,   5,   8,   4],
         [ 26,  18,   9,   5,   8,  37],
         [ 19,  17,   7,   5,   8,  87],
         [ 19,  11,  11,   7,   8,  38],
         [ 19,  29,   8,   9,   8,  30],
         [ 23,  32,   5,   9,   8,   4],
         [ 23,  25,   9,   9,   8,   4],
         [ 21,  34,   8,  12,   8,  29],
         [ 21,  30,  11,  12,   8,  93],
         [ 19,   6,   9,  12,   8,  33],
         [ 21,  19,   9,  15,   8,  30],
```

Figure 8.2 Octuple output 2 of Music Transformer

From a wider perspective, the results in Figure 8.1 are more standardized while the results in Figure 8.2 are more varied. This variety in the responses allows for the model to express and depict more textual prompts and generate more diverse music. This attribute was something quite hard to achieve in previous models or by hyper parameter tuning for all the 6 parameters.

In conclusion, the qualitative analysis of the Octuple generation is that the model is more than able to solve the problems that other models faced. In fact, it can also show great variety in the outputs generated, which gives it remarkable potential to capture the varied textual prompts that can be used.

8.3 Qualitative Analysis of Generated MIDI files

There are many MIDI files that were generated by the model. To give an illustration of the model's abilities, 3 music outputs used for the survey were used, and will be evaluated qualitatively based on the prompts that were generated. This will be judged based on the musical quality of the generated music and will later be evaluated in the user study. In this Musical Analysis section, the overall quality of the music will be evaluated through its melody, harmony, musical direction, and theme [44]. The extent to which the music adheres to the prompt will also be ascertained.

8.3.1 Rationale behind selected scores

These 3 MIDI files were also used for the user study, and the table of the prompt and the converted audio file can be found in Table 8.2. The MIDI files were chosen as examples of the varied outputs the model can create on various levels.

Firstly, the excerpts are varied along their instrumentation. Excerpt 1 is a single-track output, Excerpt 3 is a more orchestral output, while Excerpt 4 has a main instrument with some interjections by other instruments.

Secondly, the excerpts are varied along their prompts. Excerpts 1 and 4 use some musical and genre descriptions to generate, while Excerpt 3 uses a more abstract and practical prompt to test if the model can generate a MIDI file that fits the music.

Lastly, the excerpts also have a varied length. Excerpts 1 and 3 are shorter in time, taking just about 10s, while Excerpt 4 is longer and runs on for 30s.

8.3.2 Introduction to Musical Analysis

In this subsection, a brief overview of musical analysis will be done to give some context for the musical evaluation of the music later.

8.3.2.1 Melodic Analysis

This would analyze the melody that is generated, based on the most prominent melody from the piece's velocity. This would involve tracking the melody, and understanding how the melody is phrased and constructed in the music.

8.3.2.2 Harmonic Analysis

A harmonic analysis would involve understanding the chord progression in the music, which is a series of combined notes that give the music structure at each point in the music. Specific sequences of chords result in different effects on the overall structure and mood generated by the piece. Such an analysis seeks to understand what chords are used and if this is coherent for the music. There are several popular chord progressions in music that make the music more tonal and are generally well liked in music. Nevertheless, a different or varied chord progression could still prove unique and generate new moods and emotions. The piece's cadences will also be examined, which involves the chord progression at the end of the music. Such cadences are important as they determine if the music sounds completed.

8.3.2.3 Dissonances in music

It should also be noted that there is subjectivity in music and in text as well, which makes this analysis very challenging. In addition, creating music that adheres to abstract prompts is extremely difficult and sometimes lead to dissonances and atonalities. This not only makes the musical analysis challenging but could also affect many people's views of the music. People are often averse to dissonances [45] and would dislike that music easily. While too much atonality is often not ideal, there are times where it might have a purpose, to create tension and resolution, to fit the uncertain nature of the prompt. As such, even if audiences do not enjoy the music, that does not necessarily make the music invalid or unmusical, and it may even serve the purposes of the prompt.

8.3.3 Excerpt 1: Dramatic Film Score

The prompt for the first music was "Dramatic film score with romantic melodies" and a screenshot of the generated MIDI file is shown below in Figure 8.3. The link to hear the music for is included in Table 8.2, and can be found at https://tinyurl.com/FypMusic1. It must first be mentioned that this score is a dramatic improvement compared to previous results. It is not repetitive with the same note appearing every time, nor does it have overly random notes and jumps.

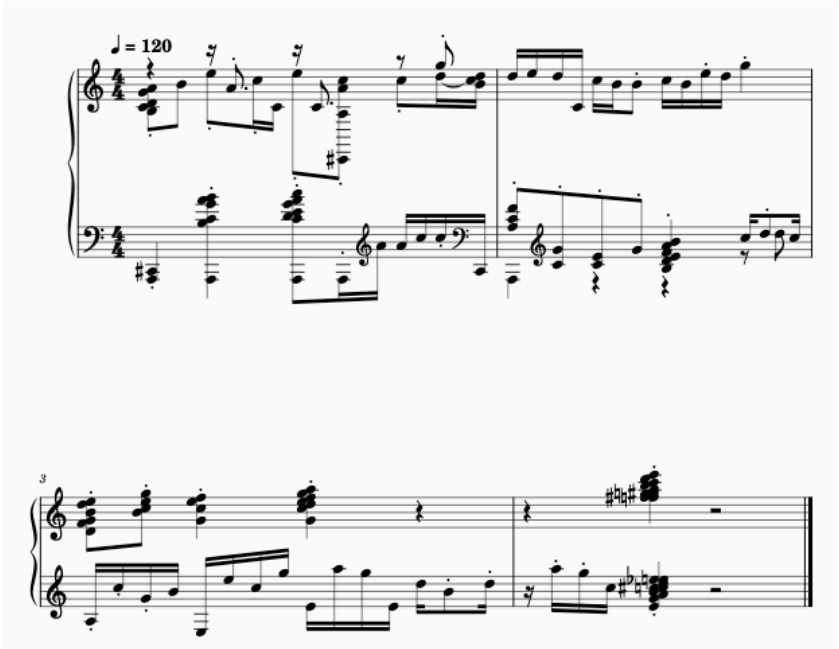

Figure 8.3 MIDI file of "Dramatic Film music with romantic melodies"

8.3.3.1 Excerpt 1: Melodic Analysis

One great strength of this generated music is the presence of a clear distinct melody. This melody alternates between the right hand and the left as highlighted by yellow in Figure 8.4. Many other previous models struggled to generate a clear melody, while the velocity of this output highlights the running melody line to a large extent.

8.3.3.2 Excerpt 1: Harmonic Analysis

This music can also be analyzed by its harmony, as represented by chords written in blue in Figure 8.4. It should be noted that the chords were based on the main notes that were more prominent and wouldn't include other less significant or passing notes. It is very remarkable that the piece does follow a variation of a 1 – 4 – 2 – 5 – 1 chord sequence. The use of a similar chord at the start and the end creates that idea of a home chord to start and end the music. The use of a 5-1 would be a common perfect cadence which the majority, if not almost all songs use as a nice ending. The difference is that more minor or diminished chords are used where it isn't a major 5 to a major 1 ending but a minor 5 to a major 1. That changes the harmony style and makes it something more novel for listeners. Therefore, even though there are some dissonances and interesting twists on the chords, they still largely follow chord sequences of popular songs, and it is impressive that the AI is able to understand and generate music that largely follow chord sequences and cadences.

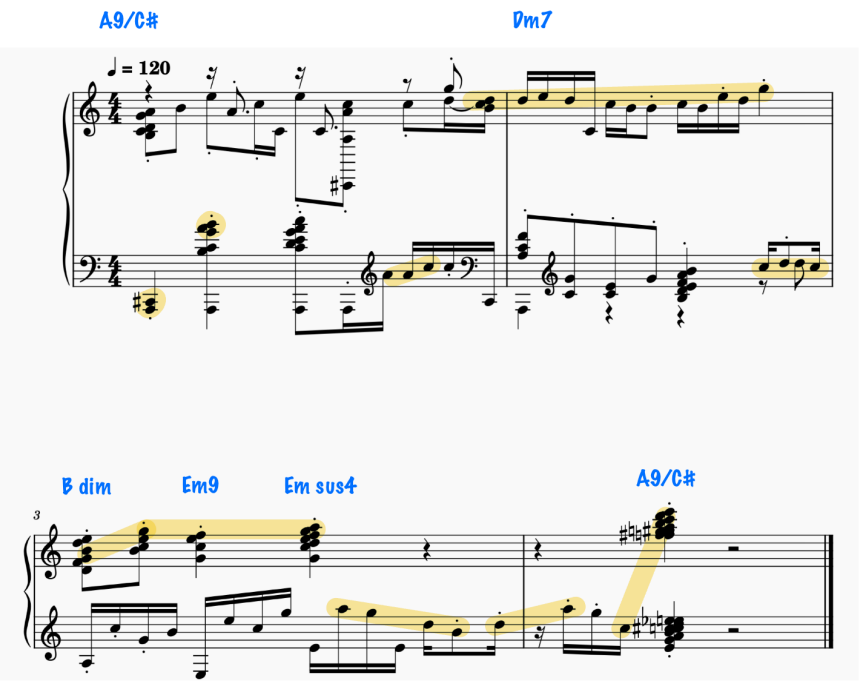

Figure 8.4 Annotated MIDI file of "Dramatic film music with romantic melodies"

8.3.3.3 Excerpt 1: Musical Direction and Theme

As for the musical direction and the theme, the music is rather playful and lighthearted, with the melody bouncing around frequently. The alternating melody between the right hand and the left further adds to that friskiness. It almost has a whimsical nature to the tune and the melody, as it doesn't settle on one spot for too long but bounces around in a very light manner. The slightly unusual chords also add to the listener's inability to settle on a chord, which adds to the whimsical mood of the piece.

8.3.3.4 Excerpt 1: Adherence to Prompt

With regards to the link with the prompt, it is true that this music might not fully encapsulate the essence of the prompt. However, it is also a very difficult prompt to describe. There are many kinds of film music for different purposes and scenes in the movie. A melody that is dramatic is also difficult to feel romantic to most people. Nevertheless, the music does capture the dramatic nature of the prompt through the sudden shifts in melody and tone, and has a distinct melody as requested by the

prompt. It is arguable that this could be a background music for a playful scene between a couple pranking each other. The lighthearted nature of the music combines with the dramatic whimsical nature could still be ultimately romantic in the larger context of the movie.

8.3.4 Excerpt 2: Noisy Alarm Bell in the Morning

Next, the prompt for the second music was “Noisy alarm bell in the morning” and a screenshot of the generated MIDI file is shown below in Figure 8.5. The link to hear the music for is included in Table 8.2, and can be found in this link: https://tinyurl.com/FypMusic3. This prompt was used as a test of whether the model was able to create a more applicable music, with a prompt that didn’t reference musical parameters but an everyday sound to see if the model could generate something like it.

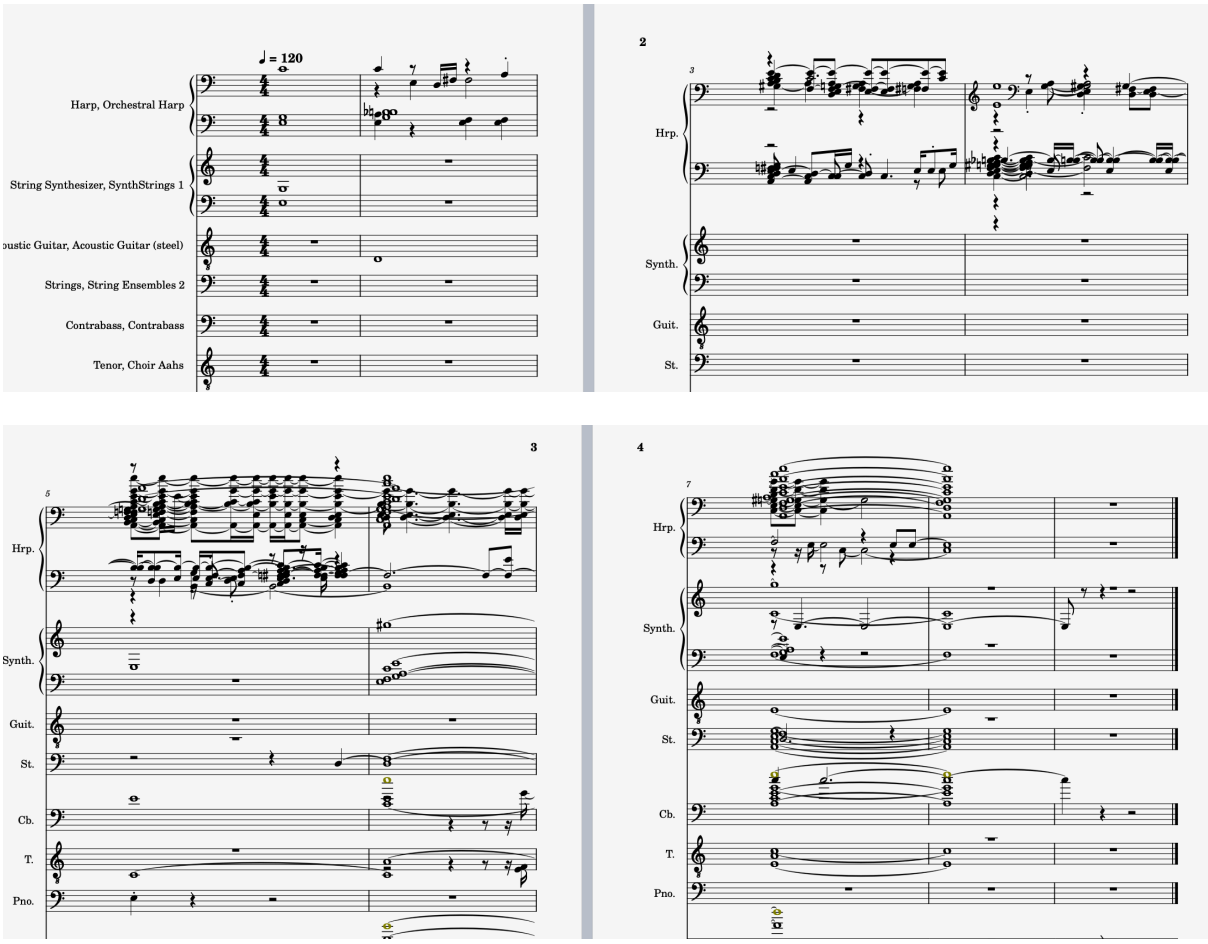

Figure 8.5 MIDI file of “Noisy alarm bell in the morning”

8.3.4.1 Excerpt 2: Instrumentation Analysis

Instrumentation wise, the model can choose a harp as the dominant instrument with several other instruments as backup supporting instruments. It illustrates that unlike the first excerpt, the model can alternate between a single instrument and a more orchestral piece. The harp's notes serve more as the melody or accompaniment in this piece, while the other instruments mainly the synthesizer, organ, strings, violas, contrabass and cellos provide that chordal crescendo.

8.3.4.2 Excerpt 2: Musical Structure Analysis

Next, the musical structure of this piece is quite influential in the music. The first half of the piece in the first 4 bars is dominated mostly by the harp's melody and other instrumentation is quiet. However, this flips in the second half where the strings and other instruments take over with a steady build in 3 main waves. More and more instruments enter the piece and the music crescendos before climaxing in the 7th bar. This is quite a captivating musical structure of 2 clear halves in this music, with different instrumentation choices by the model.

8.3.4.3 Excerpt 2: Melodic Analysis

As for the melody, it is highlighted in yellow in Figure 8.6, with it being the undertone melody running through the right-hand section. It is less volatile compared to Excerpt 1 and is marked by several accidentals that create tension at the start. Nevertheless, after the opening 2 bars, it gradually increases in pitch as it leads towards the orchestral segment especially in bars 3 and 4. This provides a strong melody push towards the first C chord in Bar 4 before the C in the orchestral bar 5.

8.3.4.4 Excerpt 2: Harmonic Analysis

Harmony wise, the chords are mostly in C with some interesting variations between them. There also is a slight foray into D7 and Dm7 in Bar 3. Looking at the start it is quite clear that the piece resides in C major with the opening C chord and the ending in C. It starts off with a strong C accentuated by the

String synthesizer in bar 1. The chord changes to largely C9 #11 might throw some listeners off with the clashing B♭, F# and F in Bar 2. This unsettledness carries on to bar 3 with G# being the main clashing note with D7 before it quickly resolves back to G in the passing notes. The chord progresses back to C before a G# is added to make it C aug 7. Even in the orchestral section, the theme of alternating between G and G# continues with a C major with the first chord, followed by C augmented for the next 2 chords. As such, the harmony analysis shows an intriguing dynamic to the music with tensions and a sense of unpredictability. It is remarkable that even if it sounds clashing or dissonant, there are clear patterns to the music, that it largely stays in a similar chord, but uses the one semitone incremental movements to create and resolve tension in the music.

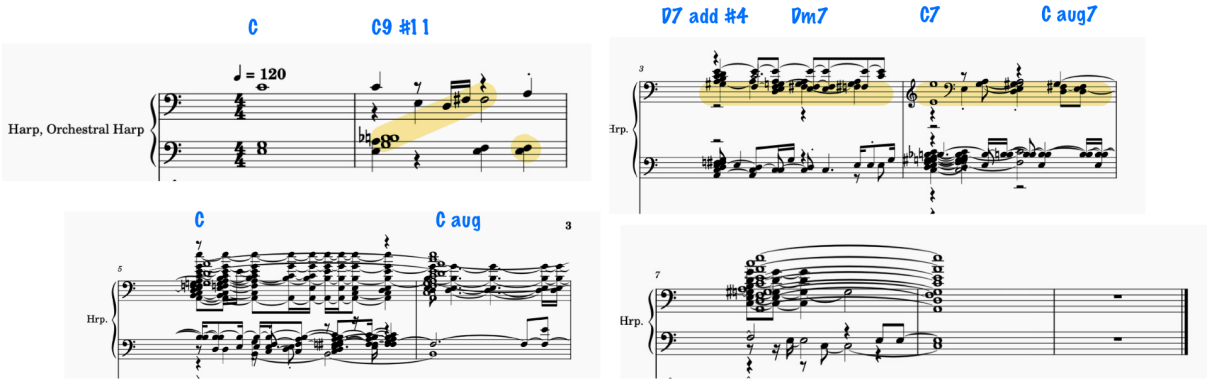

Figure 8.6 Annotated MIDI file of "Noisy alarm bell in the morning"

8.3.4.5 Excerpt 2: Adherence to Prompt

With regards to the adherence to the prompt, this music does capture the essence of a noisy alarm bell quite well. The more obvious reference would be the increase in dynamics especially in the last 4 bars which makes it more evident that the listener should wake up. The added instrumentation also adds to the loudness and urgency of the piece. This satisfies the noisy and alarm section of the prompt. On a deeper level, the piece starts off with a tonal C that hints to what is coming followed by a melody on a harp. This would ease the listener into the music, by playing a melody in the hopes that one would wake up. The second half of the piece is almost a response if the listener continues to sleep and ignore the alarm. The clashing notes are also especially helpful to create a sense of discomfort to encourage the listener to wake up. In this case it is quite interesting as even if the listener does not like this

melody that could still be very effective and useful in the context of an alarm in the morning. It is unfortunate that the model did not pick up on the bell idea and introduce some bell sounds, but this still largely follows the mood and essence of the prompt.

8.3.5 Excerpt 3: Frantic and Fast Paced Melody Lines

Next, the prompt for the second music was “Frantic and Fast Paced Melody Lines” and a screenshot of the generated MIDI file is shown below in Figure 8.7. The link to hear the music for is included in Table 8.2, at https://tinyurl.com/FypMusic4 . This music was chosen as an example of a longer excerpt music that was more melodious and involved non piano instruments.

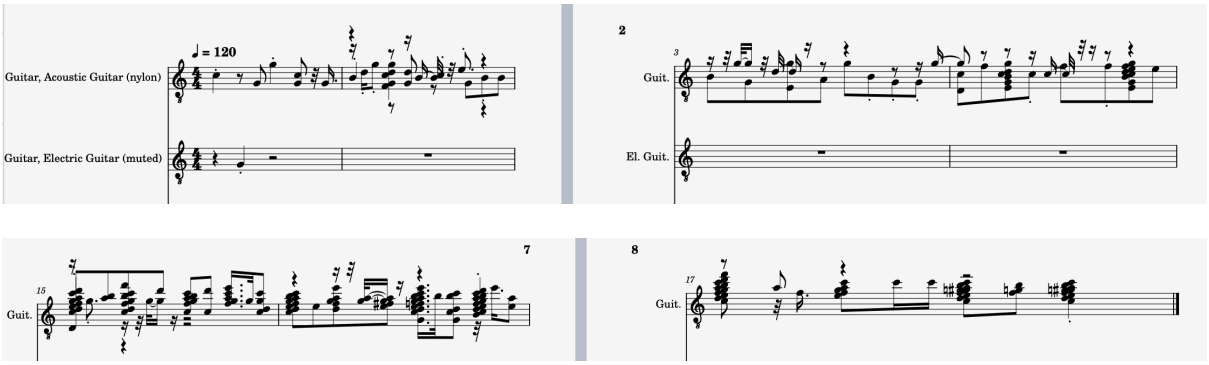

Figure 8.7 MIDI file of “Frantic and Fast Paced Melody Lines”

8.3.5.1 Excerpt 3: Musical Structure

This excerpt is harder to analyze due its length, and as such a full musical analysis will not be done. There is a clearer introduction in the first two bars before the musical idea develops further. The piece continues with the same consistent upbeat mood and driving structure and flow. It loses a bit of direction in the middle, but its consistency in theme is remarkable. Near the end however, there is a repeat of motif before the cadence is reached and it ends off. The ending is also heightened by the sense of the gong as a notice of finality.

8.3.5.2 Excerpt 3: Harmony

The piece is in C major and largely stays in this key, which adds to the coherency of the piece. It alternates between C chords and G chords as well as other related chords due to the accidentals in the center of the piece. Some chordal analysis of Excerpt 3's ending is shown in Figure 8.8. Though the transition is fast, the chord changes is unique and plays on the effect of similar chords. While the transition of a B diminished 7 to a C is more novel, the music model smoothly transitions from a B diminished 7 to a B diminished 7 flat 6 with just a semi tone drop from the A flat to the G. The unique choice of a B diminished flat 6 before the C is that it has the same notes as G7, which would be a perfect cadence. This shows a subtle play on the chords to generate novel chord progressions.

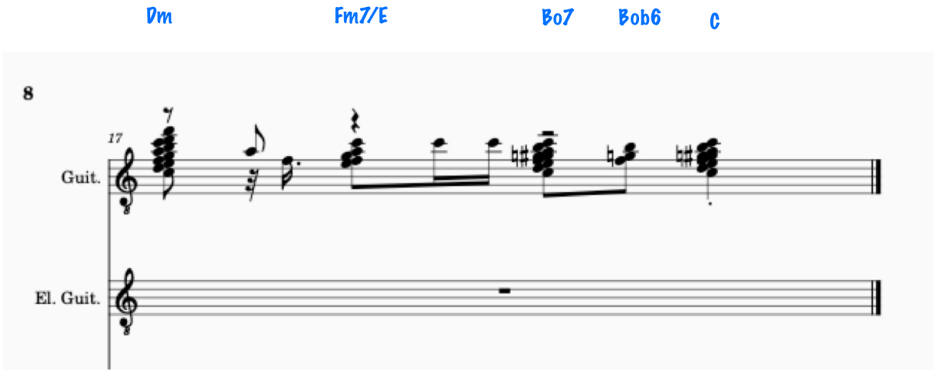

Figure 8.8 Harmony of ending in Excerpt 3

8.3.5.3 Excerpt 3: Tone

The choice of a guitar plucking, and the fast nature of the piece made sure that the piece felt upbeat and lively. This added to the fast-paced nature of the piece, even though frantic might not be fully described in this excerpt.

8.3.5.4 Excerpt 3: Rhythm

There is also an alternate usage of dotted notes and regular quavers to vary the rhythm. The dotted rhythms are found most throughout the piece, but most notably at the start and at the end. This alternation in rhythm keeps the piece fresh yet ensures that it sounds like a consistent theme.

8.4 User Study

A survey would be done, and participants would share their response to the music, to provide qualitative and quantitative feedback.

8.4.1 Overview of User Study

The objective of the survey would be to evaluate 3 of the AI generated music and their prompts, with a human generated excerpt used in the dataset as a benchmark. A key feature of this user study is that the survey respondent would not know if the music was generated by AI or not, nor would they know the prompt used to generate it. This lends the study greater credibility as it would be purely based on the user's thoughts on the music at that point without their opinion being affected by whether the music was created by AI or not.

To achieve this, many of the excerpt specific questions would be asked in this initial stage before the answer was revealed and additional questions were asked. Additional information was also taken before listening to the excerpts as well as after to draw more conclusions on the results. It was desired to obtain unbiased results and compare the 3 different AI generated music with the human generated music.

A table showing the excerpts used in the study with the links to the audio file is shown in Table 8.2.

User Study Music	
Excerpts	**Music Link**
Excerpt 1: AI music generated by prompt: "Dramatic film music with romantic melodies"	https://tinyurl.com/FypMusic1
Excerpt 2: Human composed music	https://tinyurl.com/FypMusic2
Excerpt 3: AI music generated by prompt: "Noisy alarm bell in the morning"	https://tinyurl.com/FypMusic3
Excerpt 4: AI music generated by prompt: "Frantic and fast paced melody lines"	https://tinyurl.com/FypMusic4

Table 8.2: Excerpts used in user study

8.4.2 User Study Questions

To achieve the User Study objective, some questions were asked both before and after the user listened to the excerpts, as seen in Table 8.3. This was used to understand the user's perspectives on music through the first three questions, and to obtain general views of the whole AI model and music generated in the last 3 questions.

General questions	
Before / after excerpts	**Questions**
Before	On a scale of 1 to 10, how would you rate your musical expertise?
Before	How frequently do you listen to music?
Before	On a scale of 1 to 10, what is your impression of AI generated music?
After	On a scale of 1 to 10, how varied was the music that was generated?
After	On a scale of 1 to 10, how useful do you think this AI model will be?
After	Do you think this AI model is better at making melodious music (like traditional pop songs) or background music / sound effects?

Table 8.3: General questions asked before or after listening to the 4 music pieces

For each of the excerpts, the following questions were asked in Table 8.4. There would be 2 sections for each AI-composed excerpt and only 1 section for the human-composed music excerpt. The first section would comprise of all the questions asked before revealing the prompt. Then, after answering those questions, the user will get to see the real prompt used, or in the case of the human composed music find out that it was not AI generated. Many questions were scaled questions to give the user more freedom to express their opinions on the question.

Extract-specific questions	
Before / after revealing prompt	**Questions**
Before	On a scale of 1 to 10, how much do you like this music?
Before	What do you like or dislike about it
Before	On a scale of 1 to 10, how melodious (pleasant or tuneful) do you find this music?
Before	On a scale of 1 to 10, how harmonious (coherent sound of multiple pitches) do you find this music?
Before	On a scale of 1 to 10, how likely do you think this was composed by a human?
Before	If this was created by an AI, what textual prompt do you think was used to create this music?
After	On a scale of 1 to 10, how much do you think the music captures the essence of the prompt?
After	How do you believe the music either conveyed or failed to convey the idea of the textual prompt?

Table 8.4: Excerpt specific questions before and after revealing the prompt

For the question asking users to guess which textual prompt was used to create the music, the real prompt was mixed with 3 other options chosen by the surveyor. It was difficult to decide what would be fair options to use, but at least 1 other option was ensured to be a prompt that sounded close to the music as well. It should be noted that while this question was asked, it is almost a reversal of the project from converting music to text, which is another very challenging task. Even if a piece of music is well generated from a text, it might be difficult to identify the right text among a few possible options.

8.5 User Study Results

8.5.1 Overview of Results

The User Study was done with 42 respondents, with the majority being students from Nanyang Technological University. 22 of them were classified as non-musicians and rated their musical

expertise 1 to 5. 20 of them were classified as musicians with a rating of their musical expertise of 6 to 10. This distribution and results can be seen in Figure 8.9.

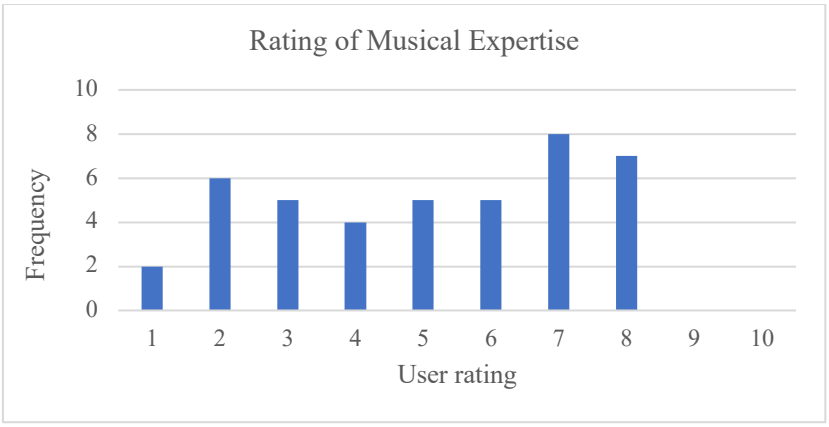

Figure 8.9 Results of Musical Expertise Rating

Many respondents also listened to music frequently with 83.33% listening to music more than once a day, as can be seen from Figure 8.10

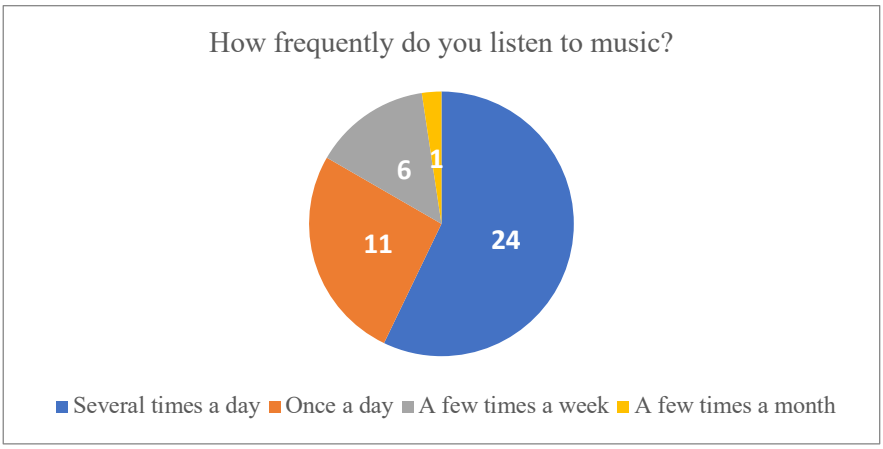

Figure 8.10 Results of Music Listening Frequency

However, the participants' impression of AI music was already not very positive. As can be seen from the distribution in Figure 8.11, the most chosen answer was just a 5/10, with the average being just 4.928. It is quite likely that this would affect their initial impression of the rest of the AI generated works, as there would already be a slight negative impression of the music. As a result, the ratings for subsequent music could also be affected by this impression.

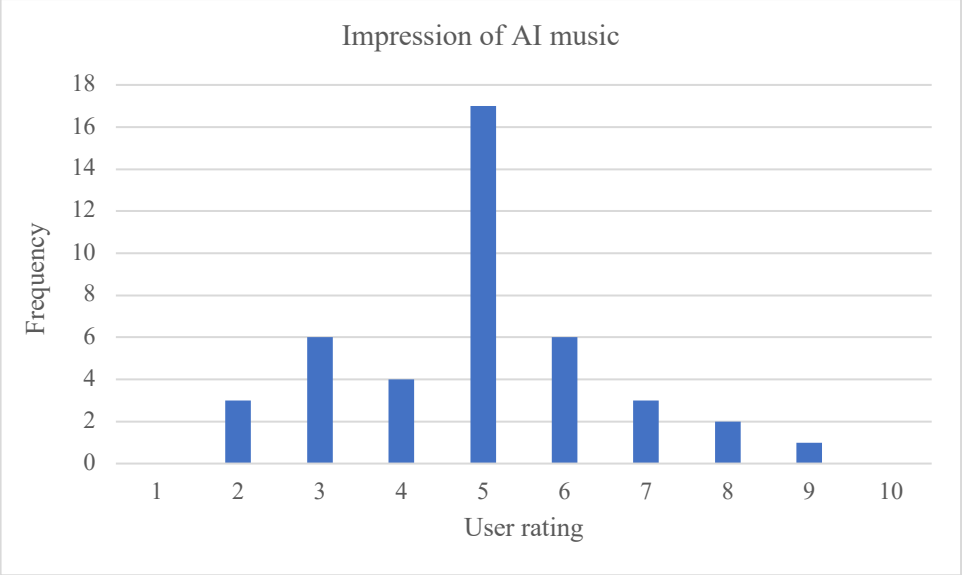

Figure 8.11 Participants' Impression of AI music

8.5.2 Excerpt 1 Results: "Dramatic Film Score"

The distribution of the results for the user's liking the music, finding it melodious and harmonious can be seen in Figure 8.12. It can be observed that the distribution of results is very wide, with 29 of the 30 scalar options being selected in these questions. This firstly is a testament to how polarizing this musical excerpt was, with many different views on the strengths and weaknesses of the music.

It can be observed that the mode for Like and Melody is 7, although there is a distribution of responses between 3 and 7 that pulls the general average down to 5.62 for Like and 5.48 for Melody. Harmony performed poorer at 4.93 on average.

It should be emphasized that the scores should be looked at in relation to the user's impression of AI music which were slightly negative to start off. The higher scores of Like and Melody compared to harmony shows that the users could likely identify and appreciate the melody. However, the presence of some dissonance or interesting harmonies likely resulted in a lower score for Harmony.

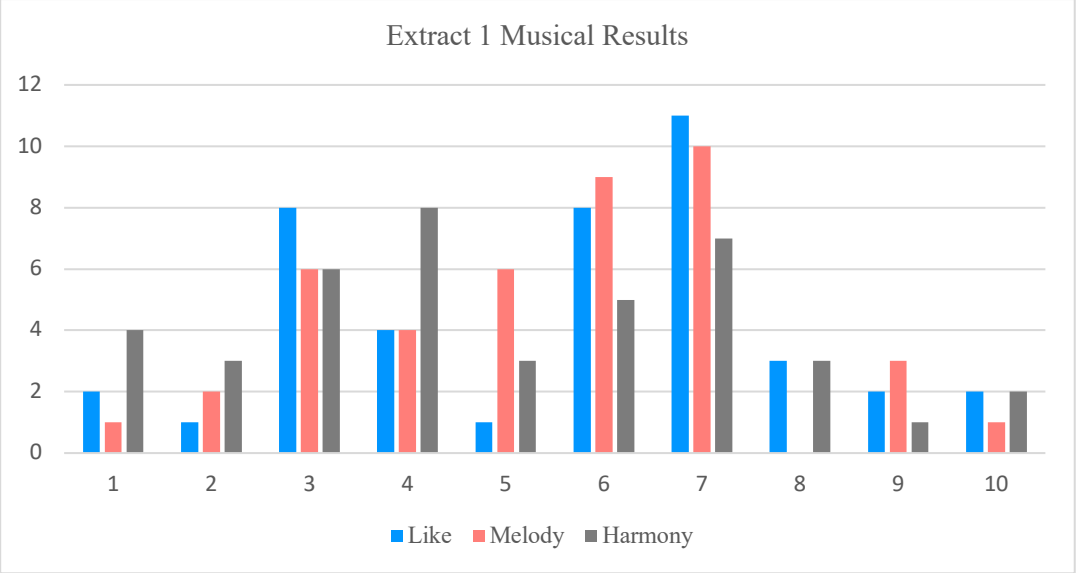

Figure 8.12 Excerpt 1 Musical Results Distribution

Some qualitative results for how users found the music can be found in Table 8.5 which shows both what the users like and dislike about the music. Some common sentiments were that the start was usually well liked by most people, while people found the ending abrupt. People enjoyed the uniqueness, playful and upbeat nature, and theme of the piece, while others found it erratic and were uncomfortable with the atonality or clashing notes.

Do you like the music?	
Like	**Dislike**
Simple, vibrant	A few notes felt out of place
it actually has some form of harmonic language	I dislike that it's not melodious and doesn't sound like it conforms to a particular music key.
I like the haunting effect, but it's a very niche tune	dislike how it ends quite abruptly
I like that it's quirky and doesn't sound like anything I've ever heard before.	It ends off a bit messy
it is chirpy and spirited	Too erratic
I like the playfulness at the start	The random-sounding notes and unpredictability
quirky and unique, sounds like it could be used in some form of game music	feels like a random mess of notes very messy
I like the speed and sound of it	It started out happy but ended a little strangely so I'm confused on what the vibe is.

It sounds very upbeat!	Too technical, not cohesive, no tonality, no phrasing etc.
It was pleasant	Abrupt, cold, clinical, not much melody
Enjoyed the 20th C music vibes, fresh dissonant harmonies (a bit like Paul Hindemith pieces)	I'm not a big fan of classical music so this music isn't to my liking.
the melody is abit unusual but I can still follow the melody	Doesnt sound very tonal.
I like the rhythm and swing, its got kind of a groove to it.	

Table 8.5: Excerpt 1 Qualitative feedback on music

Users were asked which textual prompt was most likely used to create the music and the results are in Figure 8.13. Very few users were able to identify the real prompt as this was a difficult question. Exciting electronic groove with uplifting energy was the more popular prompt by far. Between the most popular prompt and the real prompt, exciting and uplifting are words quite like the dramatic part of the prompt. However, even though electronic groove wouldn't likely be represented by piano, it was likely to be closer to the user's views compared to it being film score with romantic melodies.

Next, the real prompt was then revealed, and users were asked if the music was close to the prompt. The result distribution is shown in Figure 8.14, with most people disagreeing and an average result of 4.55.

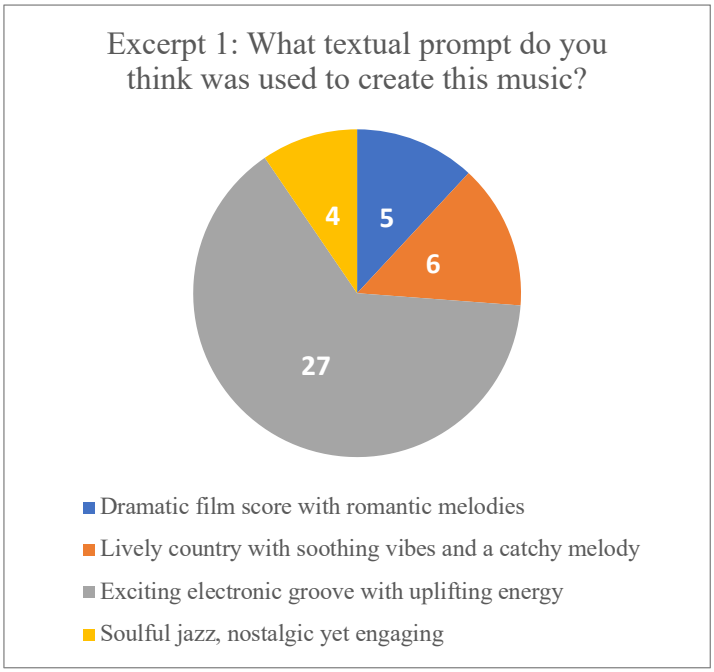

Figure 8.13 Excerpt 1 Textual Prompt results

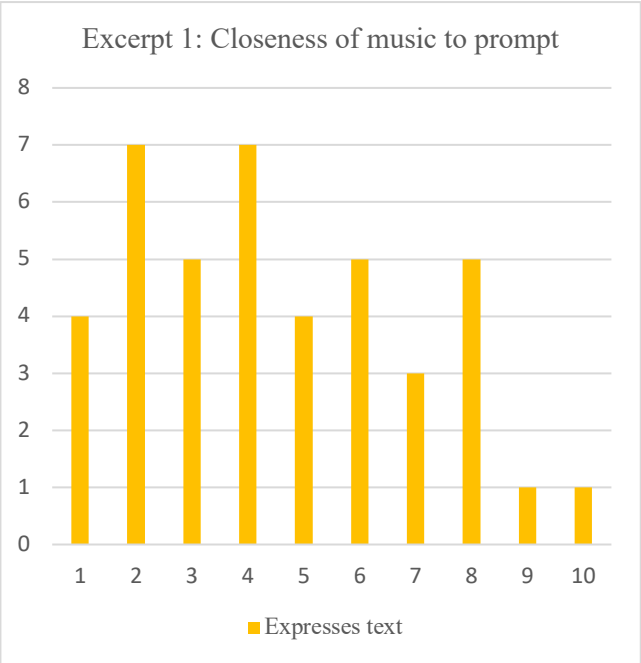

Figure 8.14 Excerpt 1 Textual Result Distribution

The qualitative answers are shown in Table 8.6, where most people agreed that it was dramatic. However, they did not think that the romantic and the film score portions of the prompt were well represented in the music. This ultimately led to a lower score for the textual section of this music.

How do you think the music conveyed the idea of the prompt?	
It did convey	**It couldn't convey**
The music managed to capture the dramatic aspect as heard from the first half of the tune	More upbeat than dramatic, not really romantic
It did sound dramatic especially at the end	can't really tell the romantic aspect of the music but it does sound dramatic
I think it did "Dramatic" well and "romantic" decently.	when i think dramatic film score i imagine hans zimmer, not a solo piano oops playing a fast paced melody
	I don't think it fully captured the romantic essence? It is indeed dramatic but for a prompt that is dramatic + romance for a film, I would imagine something more melodic and slower pace. Eg, My heart will go on, Titanic

Table 8.6: Excerpt 1 Qualitative feedback on Textual Prompt relation

8.5.3 Excerpt 3 results: “Noisy Alarm Bell in the Morning”

Moving on to the next excerpt, the musical results was shown in Figure 8.15, where the most common choice for ‘Like’ was 3 and 6. On the other hand, Melody and Harmony had their most common rating at 4 and 3 respectively. Therefore, the average results for all three metrics were 4.67 for like, 4.48 for melody and 4.38 for harmony.

Once again it should be noted that even though these musical metrics and likeability performed poorly in this user study, this does not necessarily mean this is a bad piece. It should be noted that in the context of a noisy alarm bell, these traits may even be something necessary for users to wake up, which would be the purpose and functionality of the music.

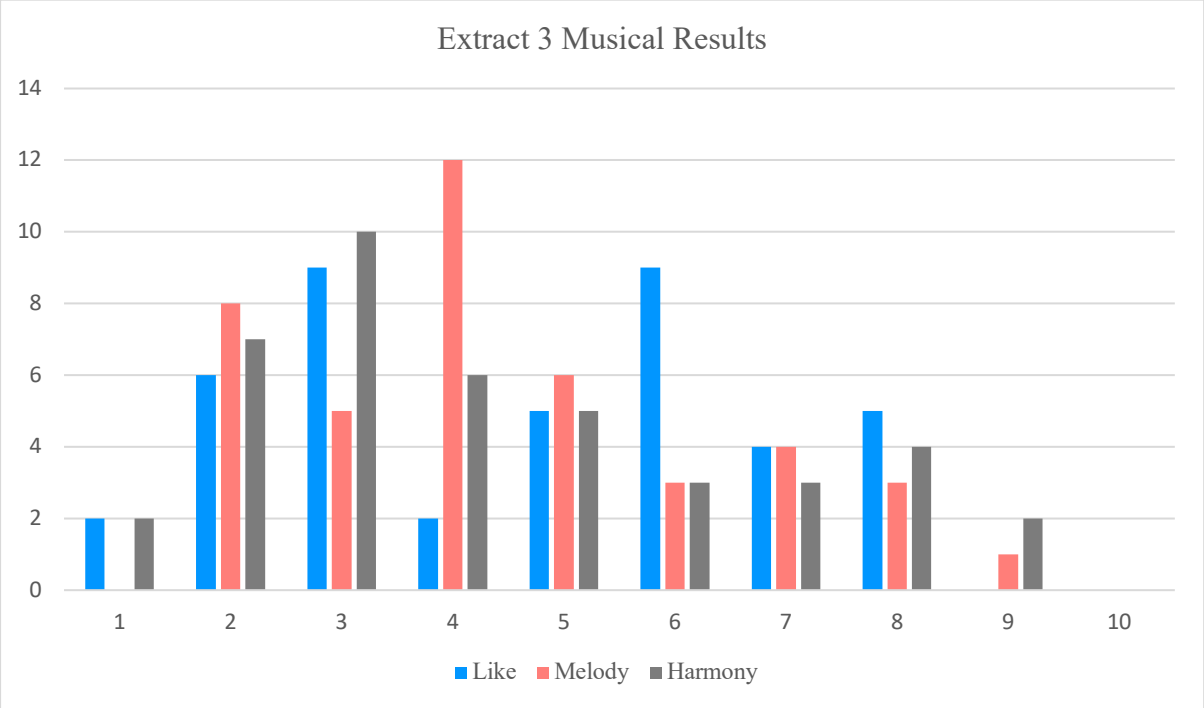

Figure 8.15 Excerpt 3 Musical Result Distribution

The qualitative results of the excerpt can be seen in Table 8.7. Users generally appreciated the complex build up and grandeur of the piece. However, others usually did not appreciate the dissonance and the unpleasant feeling the piece of music generated.

Do you like the music?	
Like	**Dislike**
I think it's fitting for a horror/ thriller kind of film. It's tense and there was a nice gradual build up	the notes were quite draggy and somewhat clashing
I like that it's very intense and if that's the emotion it was intended to invoke, then it was successful.	There's no flow to the song, very disjointed.
Like the grand-ish ending	The dissonance and unpredictability
very dramatic and chilling, love the build up of instruments on the E	Too much dissonance
liked the contrast in instruments	not pleasant to the ears
I liked the complexity of the song, the varying dynamics.	music failed to properly resolve it or reach a full climax

Table 8.7: Excerpt 3 Qualitative Musical Results

When asked which textual prompt was used, there was two clear favorites between "Noisy alarm bell in the morning and "Cacophony of voices in a crowded marketplace" as shown in Figure 8.16. Both prompts can be rather similar, with the noisy aspect both being expressed by the two prompts. The cacophony of voices can be also expressed as the music had multiple instruments join in eventually in the music. The qualitative feedback from Table 8.8 showed that many users were conflicted between these two options. This shouldn't take away from the real prompt but show that the same music can have multiple texts that describe it just as well.

Upon revealing the real prompt, the results for the relevance of the music were high with 69.0% of respondents giving it a score 6 or greater as shown in Figure 8.17. The mode of the results was 7 by a large extent, although there were some who disagreed with the alarm prompt. Once again, the distribution is still wide due to the subjectivities of textual contexts and general music perceptions.

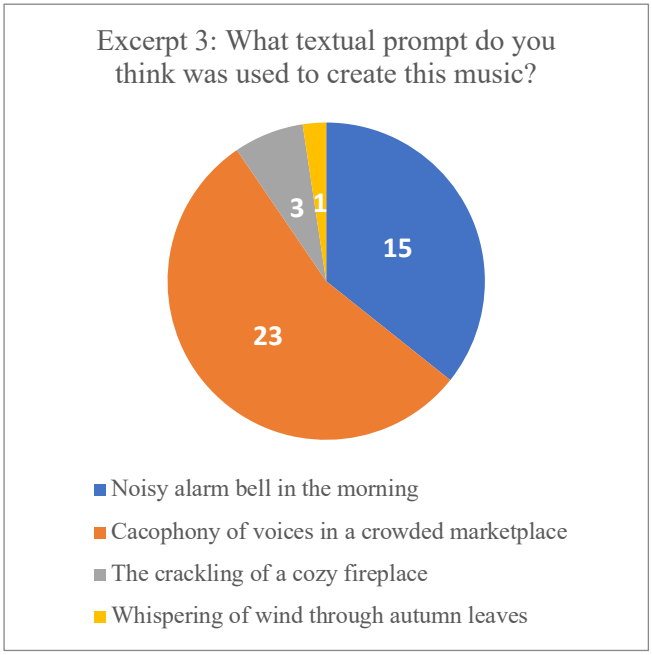

Figure 8.16 Excerpt 3 Textual Prompt Results

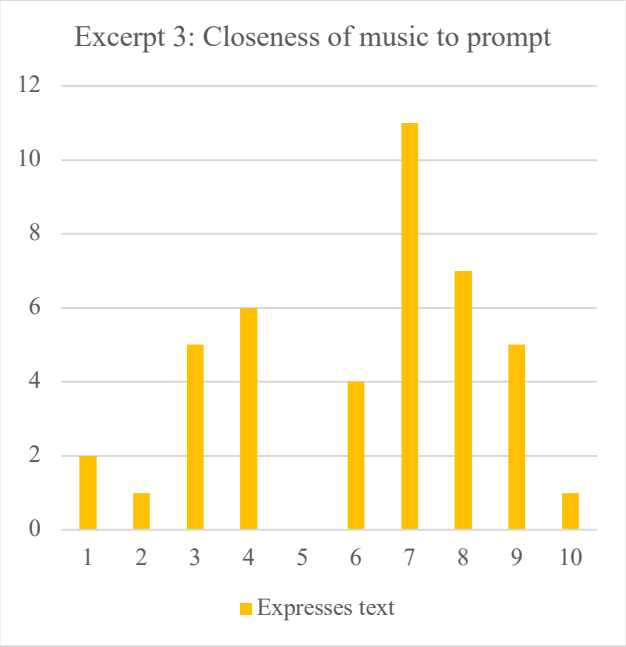

Figure 8.17 Excerpt 3 Textual Distribution

As shown in Table 8.8, there was a surprising number of conflicting views on what an alarm sound should be. Many people said that it does sound like their alarm sound, that it was loud, and they would wake up, while others said that it wasn't loud enough and wouldn't really wake one up. The stark disagreements between the two sides are difficult to evaluate. However, it sheds light on how different people with different contexts can have varied perceptions on both texts and music. The most plausible contextual reason is the phones that users use. Apple alarms use more sound effects and create annoyance in listeners, while default Samsung alarms are more pleasant and relaxing. [46] This leads to polarizing perceptions of what a good alarm sound should be, which is a task quite beyond the scope of this AI model's purposes if humans can't even agree on it. In addition, some users may be heavy sleepers while others are light, and users may have played this on their devices at different

volumes. All this considered, it is arguable that this piece did even better than expressed by the results in Figure 8.17 due to the challenging contexts of this prompt.

How do you think the music conveyed the idea of the prompt?	
It did convey	**It couldn't convey**
If this is an alarm, v good. Listen already confirm will wake up.	Doesn't really wake one up
I felt a similar annoyance that I feel when I listen to my alarm bell	somewhat busy, but not noisy enough to wake me up. Too pleasant in tone
indeed very noisy. I would wake up in the morning if I heard this (to shut it off)	can be an alarm sound, but to me sounds more of notes clashing together with a little bit of techno elements
intense build up like an alarm	I think maybe if the music was a bit more loud and glaring it would be more obvious
I think it did capture the essence somewhat. I was actually torn between this option and the option for marketplace.	Failed because I would imagine the alarm to be noisier and there was no essence of a bell in the sound.
The sudden tempo and crescendo? Reminded me of an alarm bell in the morning	I expect alarm bells to be continuous ringing and not periodic like in the piece
The sudden sounds do feel like an alarm	The music could have included more 'bell' sounds to convey the idea of a noisy alarm
the blaring sounds like the apple alarm	its different from how alarm sounds sounds. this sounds like an old video game soundtrack though.
it is loud and frantic so like an alarm	I think it's abit too minor/ dark for a morning alarm
I think it's a decent attempt. The spurt of volume at the end could sound like an alarm.	
I think it was successful in that it indeed sounds very noisy haha.	
It did have the same incoherency that an alarm bell has	

Table 8.8: Excerpt 3 Qualitative Textual Results

8.5.4 Excerpt 4 results: "Frantic and Fast Paced Melody Lines"

For the final excerpt, this piece of music by far performed the best musically. When compared to the human composed excerpt 2, the AI generated music performed better on all metrics, with even a

notable 0.9 advantage in rating for music likeability. These results are tabulated in Table 8.9 for comparison. This provides a tangible example that this music transformer model can create music that outperforms some human generated music.

Average results for Excerpt 2 and Excerpt 4		
Question	**Excerpt 2 (Human composed)**	**Excerpt 4 (AI composed)**
Likeability	5.31	6.21
Melody	5.81	6.64
Harmony	5.86	6.21
Human-like	4.55	5.12

Table 8.9: Musical Results compared between Human and AI

The result distribution is shown in Figure 8.18, with the peaks of the normally distributed curve being 7 for 'Like' and Melody and 8 for harmony. This highlights a drastic improvement in results for this piece of music and is largely agreed by most users. The distribution is also more normal compared to other extracts, making these conclusions more representative of the surveyed users.

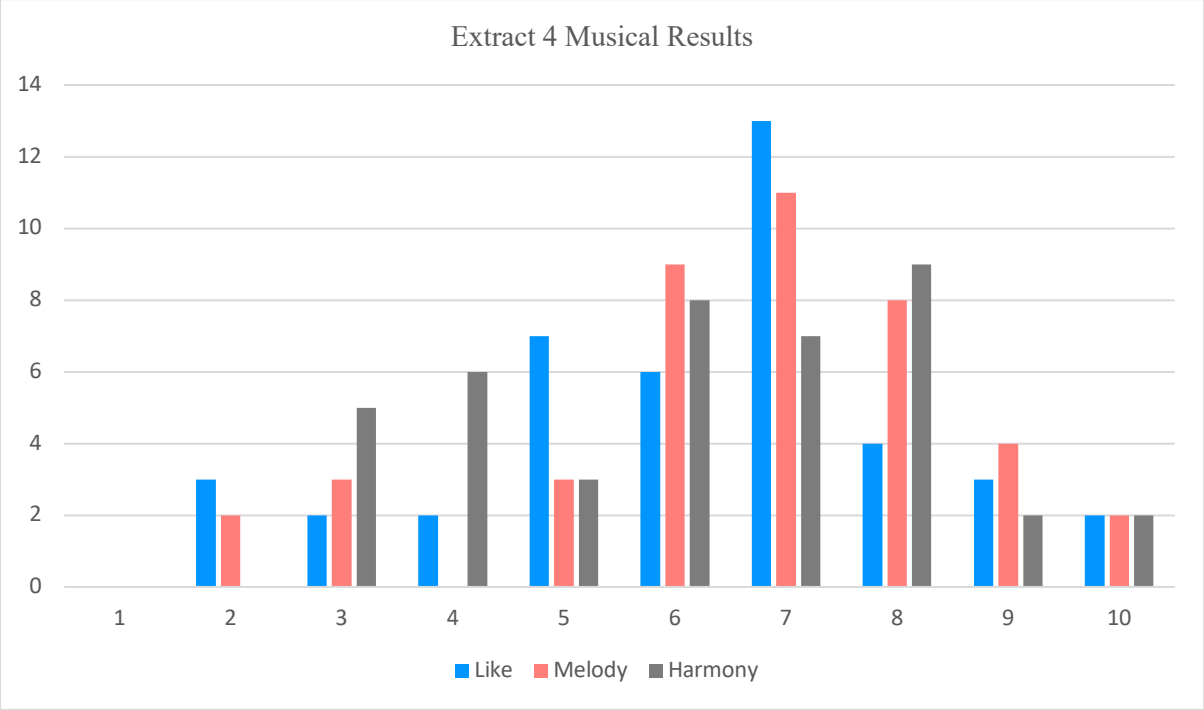

Figure 8.18 Excerpt 4 Musical Distribution Results

The qualitative musical opinions of some users are displayed in Table 8.10. In general, people liked the cheerful, peaceful, and cohesive music, but disliked the lack of direction and the additional sounds from other instruments.

Do you like the music?	
Like	**Dislike**
I like how every tune sounds like an adventure, and i could tell when it ended by the ‘bell’ as compared to previous music	disliked the two random sounds (buzz and dong) in the middle and end respectively
It has a consistent theme!	inconsistent rhythm
quite nostalgic with chill vibes. reminds of 8 bit type games or those farming kinda games where this music is played in the background and looped	it sounds like an attempt to mix everything together, but cant really tell the direction or the mood of the music
Rhythmically cohesive, tonal, interesting layering	The music went round and round with random dissonances and no direction
Calming & peaceful string instrument, but didn’t like the occasional interruption of a “bang”	Some of the sounds feel a little random
It gave a nice positive vibe	it didnt feel that coherent
I liked the tune, felt like it could belong in a game. The bell at the end helps add to the painting of the scenery.	There were several points were instruments were overlapping one another. There were odd sudden differences in pitch and intonation. The melody seems to be abit everywhere as well
I like that it sounds uplifting and cheerful. And it's more or less harmonious with some dissonant notes.	music started out nice but seemed to have lost some direction along the way, building into a series of country-vibe notes and ending with a ringing of the bell.
Upbeat and nice	Dislike that some notes are too repetitive
I can hear a key/mode. Distinct placement of instruments that is not random	Too random, no recognizable tune…
decent melody and harmony with quirky unexpected elements	
It sounds fun	
got a village vibe	
it's cute lively and cheerful	
Like that its peaceful	

Table 8.10: Excerpt 4 Qualitative Musical Results

Next, the results for the textual prompt are shown in Figure 8.20. In a similar fashion to the Extract 3, There were 2 favorites between “lively country with soothing vibes” and “frantic and fast paced melody lines”. However, apart from the lively nature of the two prompts, both prompts were fairly different to each other. The fact that these 2 prompts were the most popular is once again testament to the diversity in interpretations of a piece of music, where what is considered frantic for someone could be soothing to another.

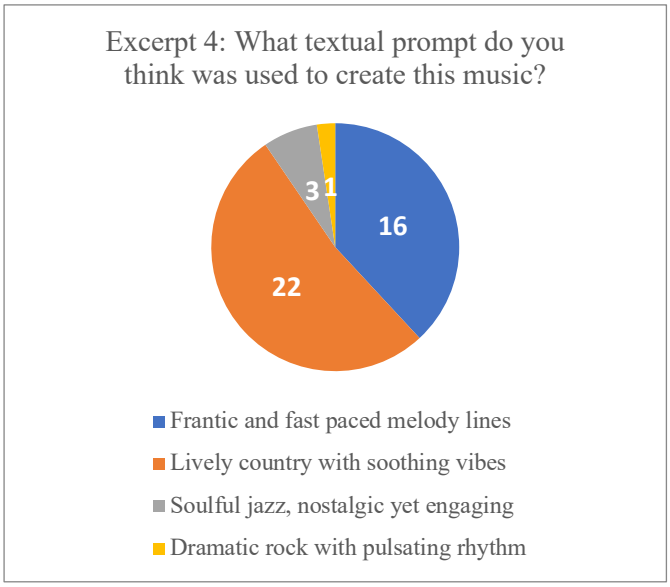

Figure 8.20 Excerpt 4 Textual Prompt Results

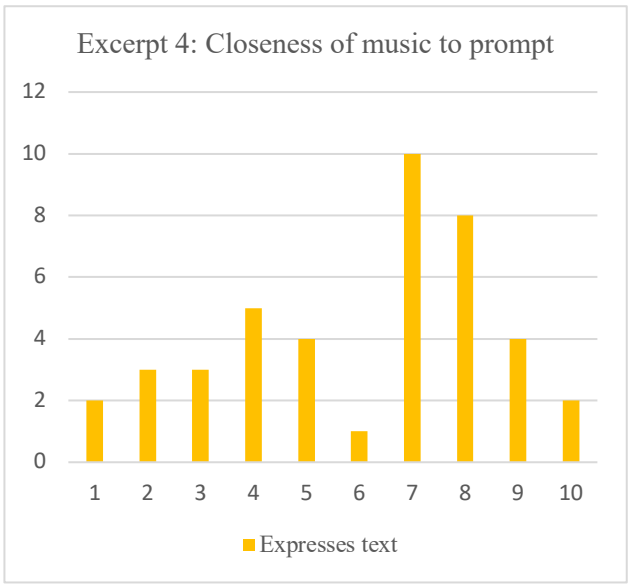

Figure 8.19 Excerpt 4 Textual Result Distribution

The distribution of the textual results can be seen in Figure 8.19, with its average score of 6.02 being very close to Extract 3’s average. Like other distributions, the results are not normally distributed, with 2 peaks, alluding to the difference in interpretation and opinions of the users.

Lastly, the qualitative feedback regarding the adherence of the text to the prompt can be found in Table 8.11. People largely agreed that it was fast paced but not to the extent of being frantic as it still felt calm and coherent. Interestingly, users who liked the piece in Table 8.7 for being peaceful and coherent likely didn’t agree with the adherence to the prompt as much. Nevertheless, even the users who did not feel the prompt was conveyed did not disagree to a large extent.

How do you think the music conveyed the idea of the prompt?	
It did convey	**It couldn’t convey**
Agree that the music was fast-paced, and effectively conveyed the idea of a frantic & fast paced melody	I felt it wasn't fast enough to be considered frantic or fast.
Fast paced was captured but an increasing pace would capture frantic much better imo	Doesn’t seems fast enough to be frantic. Also mostly is string plucking. If frantic should use strumming
the music sounds rushed	Not too bad, though I would not consider the tempo to be frantic
The roundabout aimless nature of the piece kind of reflects frantic, but not really fast paced.	it was fast, not really frantic
I don't think I felt frantic when I listened to this piece because the melody was pleasant, but I agree that it's fast paced.	Didn’t feel any sense of urgency or fast pacing, more of a sense of consistency & calmness.
it does capture the frantic and fast paced melodies because it does sound like a series of notes being mashed together. and each note sounds very short	Doesnt really convey 'frantic' When I think of frantic, i think of 'messy' and 'all over the place' but the music seems quite controlled and coherent.
I think it did feel frantic	It was upbeat but definitely not to the extent of being frantic, if I weren't paying attention this could've been background music somewhere
The many sounds coming together does seem rather frantic	

Table 8.11: Excerpt 4 Qualitative Textual Results

8.5.5 End of survey results

Finally, four final questions were asked regarding the music and model. The first question was regarding the variety of the music and the distribution is shown in Figure 8.21. Almost all users had the same impression that the music was very varied with an average result of 7.6. This affirms the progress of the model especially compared to previous iterations where so many results were too repetitive. This model shows potential in representing various texts because of its creative composition capabilities.

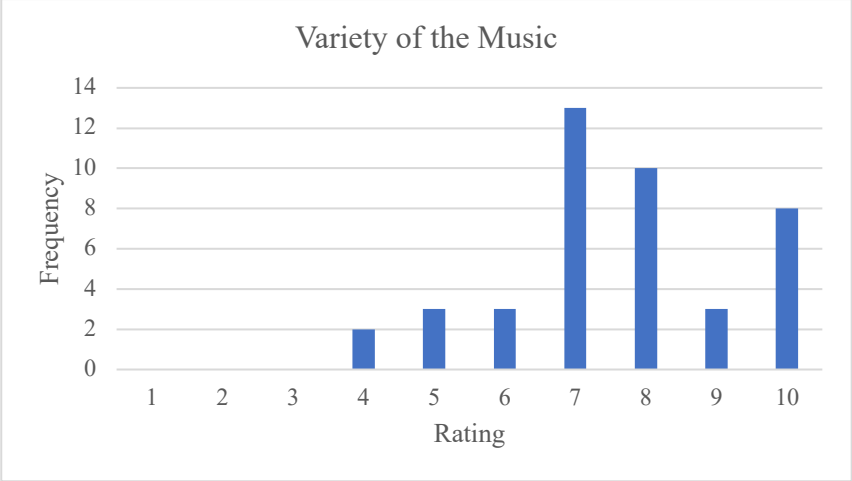

Figure 8.21 Distribution of Music Variety Results

Next, the usefulness of the model was asked, where there was quite a wide distribution of results. The average result was 6.17 which is relatively high, especially when it is considered that the initial impression of AI music was only 4.93. This can be an indirect comparison between the user's impression of AI music before and after the test, where there was a 1.24 increase in average ratings. As such, it is likely that users gained a better understanding of some of the applications and use cases of AI generated music through this survey, leading them to find this model more useful.

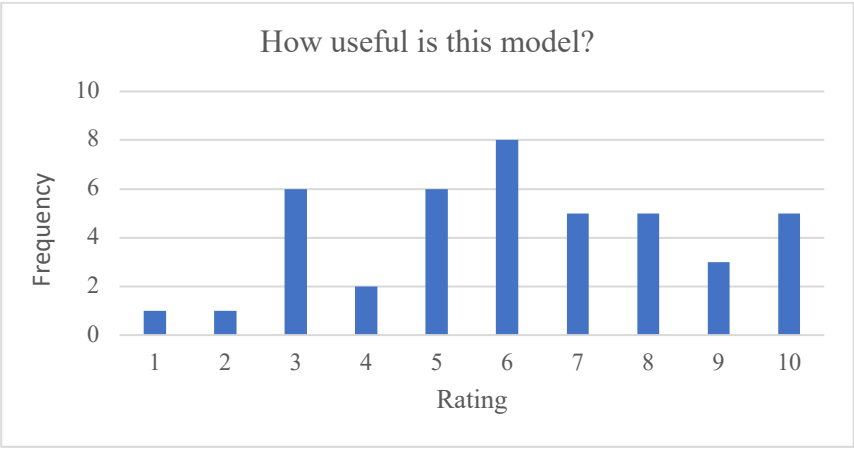

Figure 8.22 Distribution of Music Model Usefulness Results

As for which type of music this AI model would be better at composing, 83.33% of users felt that the model was more suited for background music, or sound effects compared to traditional songs as

shown in Figure 8.23. Along with the qualitative responses in previous questions, this suggests that the users believe that the AI model is more apt for tasks involving creating atmospheric accompaniments. This is due to its ability to provide nuanced variations in instrumentation, tone and mood to evoke various emotional responses that can go beyond just a normal melody.

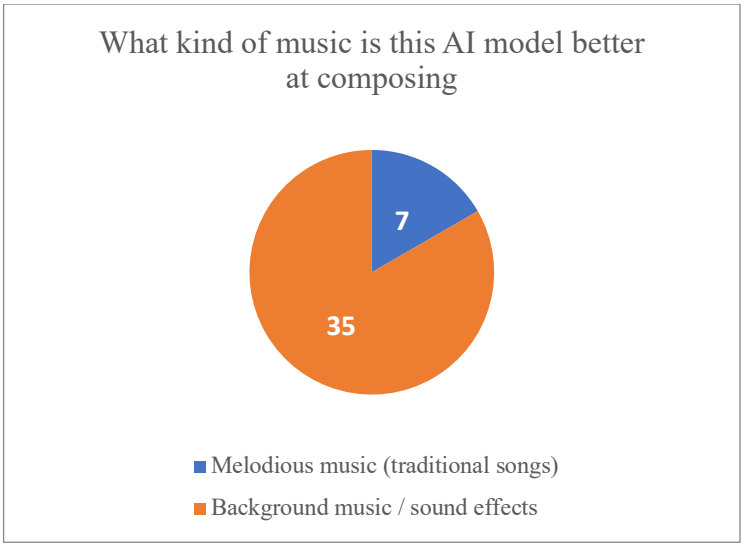

Figure 8.23 Model Generation Opinion

Lastly, the additional comments of users are depicted in Table 8.12. Users shared about the possible applications that the model could be used for, from background music, suspenseful movie tracks, game design. Other comments noted that to better generate other forms of music, it needs to be more melodious and have fewer dissonances. Users could largely identify the direction or melody, but the random notes eventually take over especially nearer the end of the piece. Additionally, this model could also be used to generate ideas for musicians.

Final comments	
Think this AI model is suitable to generate music for action, suspense, dramatic and horror movie. It needs to be more melodious and less dissonant for other genre. But good work nevertheless:)	I think the AI model generates a somewhat coherent piece but my biggest gripe is the random notes inserted here and there that is quite jarring for the listener. Would go up a few points with just the omission of those sounds
I think a recurring theme is that the model gets the vibe right at the start but seems to go a little	Very interesting as i've never heard of AI music generation before. I wonder how

random and haywire at the end. I think when we look at music pieces, we consider their entirety and possibly that plays a part in how we judge how we feel about the song!	music can continue to be shaped by AI in the future, and it also allows me to appreciate the intricacies that go behind AI music generation.
could an application be something like background music based on prompt/mood	Really cool project, the outputs are impressive!!
While it is interesting for the AI model to generate music, I think it is quite hard for the AI model to generate a piece of music that will convey feelings or emotions or messages the same way human composition works. To me all four tracks at some point sound very randomly put although there is some form of direction you can sort of figure out after a few listens. but the direction and intention of the music may not be clear to everyone. to be very honest, the AI can put together a short set of notes for a short tune, but i do not think that it can generate a song on the level of mozart of like taylor swift HAHA perhaps it needs to be trained further!	pretty good!! for most of the music, I think the generated music was pretty accurate. I think the thing that makes it easily identifiable as AI generated is the sudden changes in pitch, the
	great for generating short excerpts that could be used as ideas for longer music pieces, but unlikely to completely replace music generation entirely
	Probably needs a lot more training to produce actual music people listen to. But could be useful in certain situations if u need to come up with a specific sound
Cool stuff!	Could be good for game design.

Table 8.12: User Comments on Model Generation Project

8.5.6 Additional attempted analysis

Other analysis was also done to observe if musical training and experience would affect the results. However, no significant observations could be made. Non-musically trained users gave comparable results to musically trained users. Some slight differences were that non musically trained users preferred Excerpt 1 for all metrics while musically trained users preferred Excerpt 3 on all metrics. The mean results for musically trained users and non-musically trained users can be found in the appendix at Figure 11.1 and Figure 11.2 respectively.

8.5.7 Overview of Survey

In this section we'll be examining the consolidated results of the user study to draw some conclusions.

Firstly, the average results of the user study for the extracts are tabulated in Figure 8.24. From the diagram, a few observations can be made, firstly all extracts perform the worst on the human-like scale, even for the extract that was composed by humans. This is likely due to the audio files being generated by the machine, which further adds to the impression that it isn't human played. Secondly, the AI can outperform the human extract on all metrics in Extract 4 or be comparable such as in the case of Extract 1. Excerpt 1, Even for Extract 3 which does worse on all musical attributes, it scores extremely high for the expresses text metric.

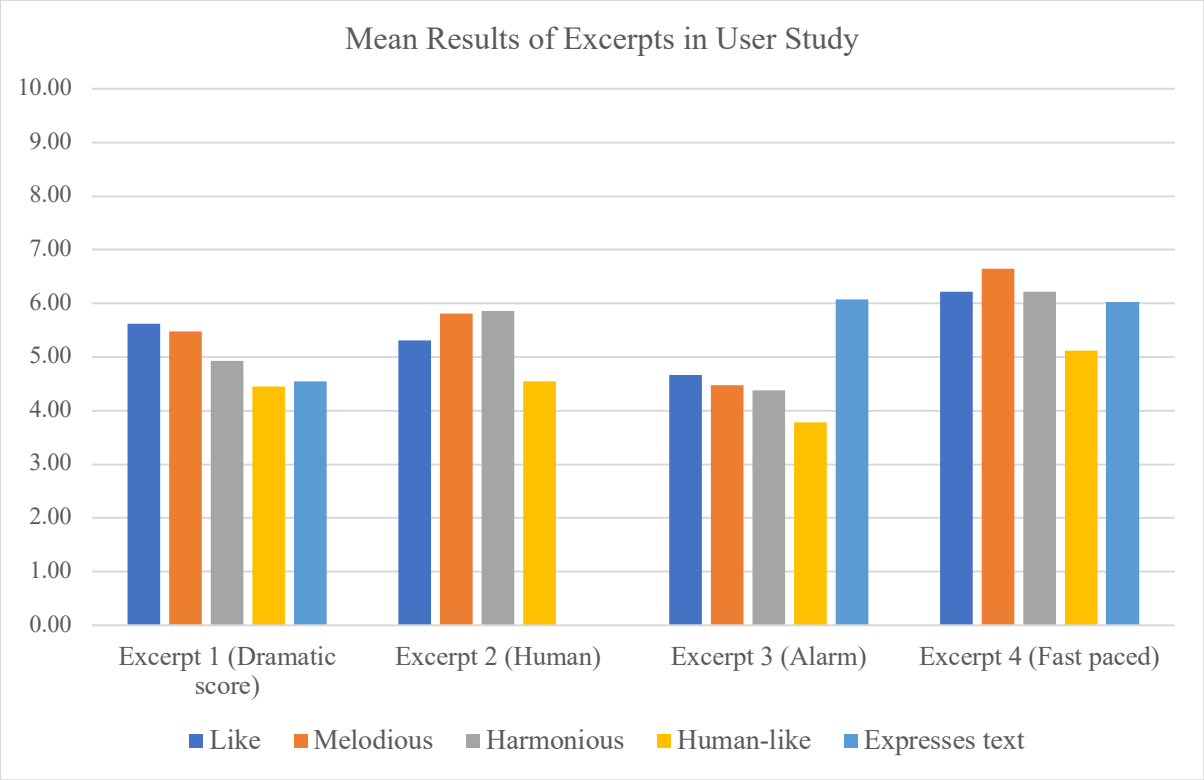

Figure 8.24 Mean user study results for all respondents

Most of the analysis in this user study has been univariate analysis with regards to each metric at a time. However, bivariate analysis exploring the relationship between users liking the music and thinking that the values fit the prompt would be a valuable analysis as well. This is especially relevant as prompt generated music would be useful if it is either likeable to the user or if it fits the prompt well. To explore this, Responses of 6-10 were considered a positive result that the user liked the music or agreed that the music adhered to the prompt. On the other hand, responses of 1-5 were considered a

negative result that the user didn't like the music or felt that the music did not adhere to the prompt respectively. The results are tabulated in Figure 8.25, and reveal helpful insights.

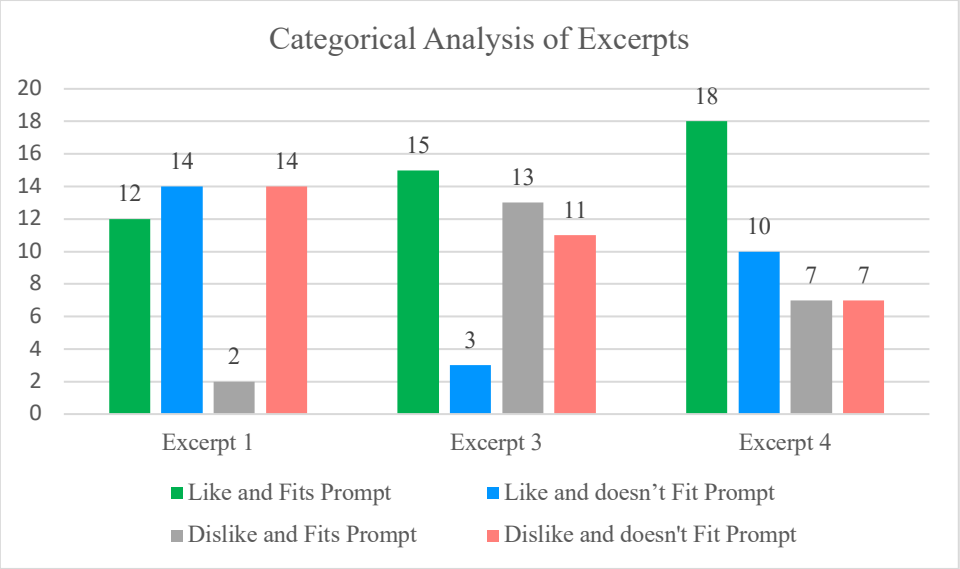

Figure 8.25 Bivariate analysis of Like and Prompt Adherence

Firstly, there are interesting observations regarding people who liked the music but felt it didn't fit the prompt and people who disliked the music but felt it fit the prompt. The number of users who fell into these two categories were relatively constant for all 3 excerpts at 16 or 17, however the distribution did vary. Excerpt one had a significant number who liked the music but felt it didn't fit the prompt. This was likely due to the prompt's elements of film and romantic melodies being difficult to resonate with even for people that enjoyed the music. Excerpt 2 was the opposite with a significant number who disliked the music but felt it fit the prompt. This is very apt and would fit the purpose of the music, as people who disliked how it sounded could still use it as a helpful alarm sound to wake them up. Lastly, Excerpt 4 was more even between the two. From the qualitative responses, some users liked the music because they found it peaceful and calming and would disagree with the prompt of frantic and fast paced melody lines. In contrast, some users who didn't like the music initially agreed that it was frantic and felt that it fit the prompt.

As can be seen for all 3 extracts, there can be a valid use case for the music if either people liked it or if it fit the prompt. Music that was liked even if it didn't fit the prompt could still be adapted by

musicians or used as musical ideas. On the other hand, music that only fit the prompt meant that it was more likely to be a useful and practical sound effect, as seen by Excerpt 3's alarm.

A summary of these results can be seen in Table 8.13, where a large majority of users did like the music or agree that the music captured the idea of the textual prompt.

User Study	
Excerpts	**Users who liked the music or agreed that it captured the idea of the textual prompt**
Excerpt 1: AI music generated by prompt: "Dramatic film music with romantic melodies"	66.67%
Excerpt 3: AI music generated by prompt: "Noisy alarm bell in the morning"	73.81%
Excerpt 4: AI music generated by prompt: "Frantic and fast paced melody lines"	83.33%

Table 8.13: Summary of Extract Results

8.5.8 Discussion of User Study Results

As a whole, the survey results largely support the hypothesis that the music model was effective in capturing the prompt or creating melodious music.

Even though the raw user study results might not seem the most impressive, there are many mitigating reasons and factors that resulted in these results. Firstly, the initial impression on AI music was already not very positive. This resulted in a negative outlook on the AI excerpts that would be presented in this case. In fact, quite a few results scored greater than the 4.93 benchmark impression which is a testament to the effectiveness of the music in improving user's expectations.

Secondly, the AI generated music performed comparably or even exceeded the human-composed music. This showed that the users were not able to discern well between the human composed and the

AI composed music and that some model generated music could outperform some human composed music.

Thirdly, the subjectivity of both text and music resulted in a wider range distribution of results, with users having contrary opinions to each other. Contextual subjectivities such as what an alarm should sound like or if Extract 4 sound frantic or peaceful divided users.

Fourthly, the survey was designed in a way to limit biasness by providing limited information on whether each extract was a human composed music or machine composed. This resulted in more honest feedback regarding the music for what it really is to the user.

Lastly, some music is also created to be less likeable because of the prompt. Just like Extract 3's alarm bell, the prompt resulted in music that was not liked by most users, but still fulfilled its purpose by fitting to the prompt.

With all these differences in opinion, and varied purposes in the music, evaluating the music on whether it is liked or captured the idea of the textual prompt is preferred. As such, this model can be seen to have fulfilled these outcomes from Table 8.13.

8.6 Overall Discussion

8.6.1 Discussion of results

8.6.1.1 Validation of Hypothesis

Overall, the complete and rigorous analysis of the musical outputs through a musical analysis and a user study shows that the model was largely able to generate melodious music that fits with the prompt. This validates the initial hypothesis of the music transformer's ability to process textual inputs and generate MIDI files that achieves this purpose.

8.6.1.2 Extensive variety of results

Going beyond just the hypothesis, the results were also incredibly diverse and varied. The model was able to process not just musical descriptions but everyday sentences. It could generate single track and multi-track MIDI files and composed music of varying styles, and moods.

8.6.1.3 Melody and Harmony

The harmonies and melodies created were complex and rich, creating unique sounds and textures to illustrate the theme of the prompt. Some generated music, being unlike the music that users usually listen to may be slightly more challenging to appreciate. However, on closer inspection, they are still largely coherent in their effort to illustrate the textual prompt.

8.6.1.4 Useful applications

As suggested from the User study, the music generated can have many applications. From creating background music to sound effects, to creating musical ideas for themes and melodies, this music model creates opportunities for these use cases to be explored. The usage of a MIDI file as an output rather than just an audio file makes it even easier to adapt and expand on the AI's music ideas. An example of this application is illustrated and further explored in Section 8.7.

8.6.2 Limitations

Nevertheless, this model still has some limitations, and 2 of them will be addressed in this sub section.

8.6.2.1 Dissonances in music

One of the notable limitations is that this model still produces dissonances in the music, and it might be difficult to produce music that is completely tonal yet non repetitive.

To address this limitation, more training would need to be conducted, where the complexity of the model can be improved to better capture the rich patterns that are in music. More exploration would need to be done to improve the pitch accuracy to overcome this limitation. In addition, more extensive hyperparameter tuning can be done to achieve better and more precise results.

However, another approach to solve this limitation is by editing the results on the MIDI file to remove the atonal sections of music. This allows for this music model to still be used in this process.

8.6.2.2 Loss of musical direction

As suggested from one of the comments of the music model, one limitation of the current music model is that even though it largely starts well but it starts to lose focus and become more random near the end.

This is a subjective interpretation of the music and difficult to ascertain the reason. More careful research could be done to examine the accuracy of the predictions throughout the musical sequence.

8.6.2.3 Random Instruments

The other limitation is that this model could generate other instruments in the process, and while this could be a strength for varied instrumentation, the user study showed that some do not enjoy the inclusion of other instruments.

This can be very readily addressed because the MIDI file generated can easily delete other instruments quickly, to remove undesired instruments in this process.

8.7 Application of model

There can be various real-life applications of this trained model, and one of which is musicians adapting the musical ideas generated by the model.

8.7.1 Generated Music Example

To illustrate this, another generated music that the model produced was based on the prompt: “tranquility of a misty morning in a woodland symphony”. The MIDI file is shown in Figure 8.26 and the link to the machine performed music is shown in Table 8.14.

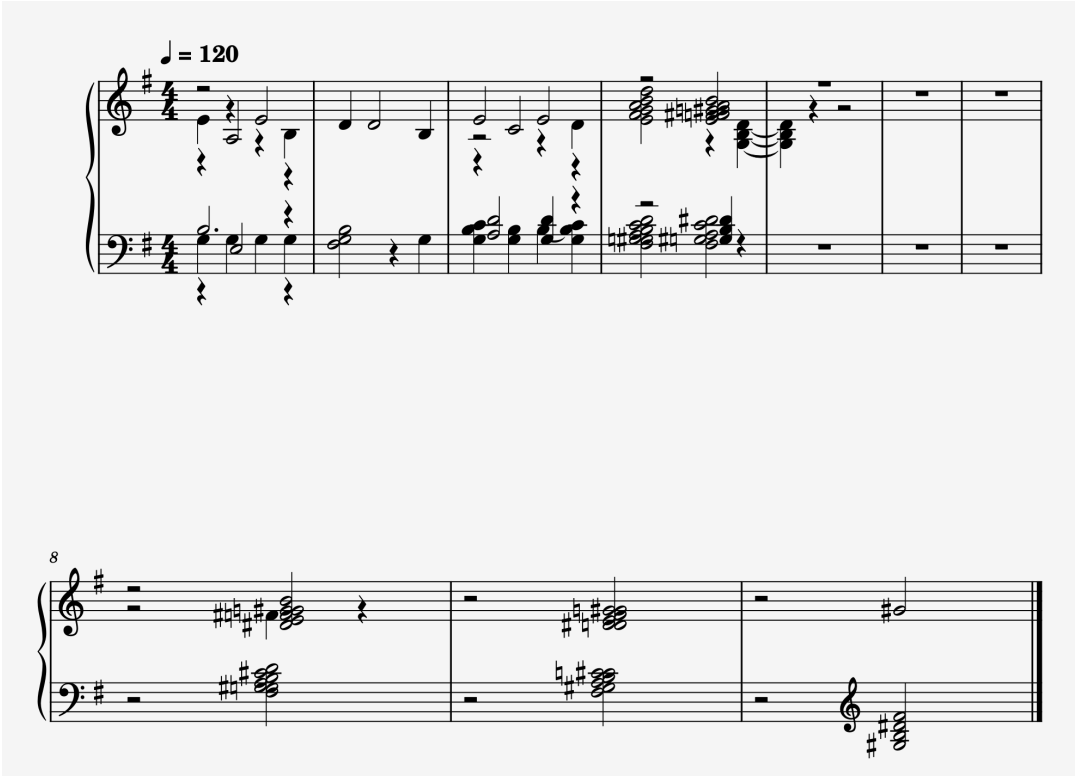

Figure 8.26: “Tranquility of a misty morning” generated music

A brief analysis of the music is that the model does well to create that tranquility atmosphere with the limited notes and slow pace of music, with deliberate pauses to highlight the mood even more. The harmony choices are also unique and interesting, which makes it slightly darker but with a smooth and distinctive cadence to end off the piece. There could still be improvements, with a more controlled start to the piece, and by polishing the chords used to make it more tonal and likeable.

8.7.2 Human Adaptation of Generated Music

To illustrate how this generated music can be adapted by a human, the author decided to adapt the ideas in this generated music and perform it accordingly. To capture the same tone of the music, the melody was largely unchanged, and the harmony was only slightly edited. Some flourishes were made to improve the quality of the music, while controlled playing of the notes would deliver the effect of the prompt to a larger extent.

The comparison between the two audio files can be seen here in Table 8.14.

Audio files of AI Generation Application	
Audio files	Links
AI music generated by prompt: “Tranquility of a misty morning in a woodland symphony”	https://tinyurl.com/FypTranquilityAI
Human Interpretation of the AI generated MIDI score	https://tinyurl.com/FypTranquilityHuman

Table 8.14: “Tranquility of a misty morning” generated Music audio and Human recording

This illustrates the potential for both humans and this AI model to use the strengths of both in delivering creative music. In this situation especially, the Music Model was able to generate useful and innovative ideas through chord progressions that were new to the author. Those ideas were adapted and reimplemented, using human experience and musical knowledge to improve the music. This is an example of how the hypothesis can be fulfilled in Section 4 with a collaborative approach between AI and the human composer.

9) Conclusion

9.1 Recommendations

This text-to-music AI model already provides many advancements in AI-generated music composition, with this novel way of transforming textual input into MIDI files for composition. While this model has impressive capabilities currently, there are some areas that could be further developed to enhance the effectiveness of this model.

9.1.1 Exploring complex models

Firstly, while this model showed that reasonable results can be achieved on 4 RTX2080 Ti GPUs, the model could be run on more powerful GPUs with more memory to facilitate the use of more complex neural network layers, heads and an increased batch size. This could lead to more effective training results and allow the model to capture more complex relationships.

9.1.2 GAN integration

Using a Generative Adversarial Network (GAN) into this music model could help refine the outputs to generate higher quality music. As covered in Section 3.3.3, training a GAN on top of this architecture would allow for the discriminator to identify what outputs sound AI generated, and improve the generated output accordingly. This creates a feedback loop to improve both the GAN as well as the neural network model.

9.1.3 Dataset generation

The CLaMP approach already helps significantly as a method to generate text descriptors for any MIDI file. Nevertheless, more expansion can be done in this area to use more specific text descriptors and non-music related text to describe the music. This could help the model understand more complex relationships and make better associations between text and music.

9.2 Conclusion

In the process of making this music model, many systems were developed. An entire transformer text to music architecture and a novel system of obtaining the text dataset from MIDI files has been fully developed. In addition, a complete evaluation, hyperparameter training process and MIDI file processing has been created. While unsuccessful, other text-to-music transformer and evaluation architecture has been developed, with the results showing that these methods are not effective in generating music.

In conclusion, the goals of the project have been achieved as this music model can effectively convert text to music that is reasonably melodious and describes the prompt.

10) References

[1] J. Post, “Don’t Lose Your Keys: Exploring the Transition from Harpsichord to Piano,” *Capstone Proj. Masters Theses*, May 2022, [Online]. Available: https://digitalcommons.csumb.edu/caps_thes_all/1306

[2] B. Bank and J. Chabassier, “Model-Based Digital Pianos: From Physics to Sound Synthesis,” *IEEE Signal Process. Mag.*, vol. 36, no. 1, pp. 103–114, Jan. 2019, doi: 10.1109/MSP.2018.2872349.

[3] J. Garon, “A Practical Introduction to Generative AI, Synthetic Media, and the Messages Found in the Latest Medium.” Rochester, NY, Mar. 14, 2023. doi: 10.2139/ssrn.4388437.

[4] M. Civit, J. Civit-Masot, F. Cuadrado, and M. J. Escalona, “A systematic review of artificial intelligence-based music generation: Scope, applications, and future trends,” *Expert Syst. Appl.*, vol. 209, p. 118190, Dec. 2022, doi: 10.1016/j.eswa.2022.118190.

[5] J.-P. Briot, G. Hadjeres, and F.-D. Pachet, “Deep Learning Techniques for Music Generation -- A Survey.” arXiv, Aug. 07, 2019. doi: 10.48550/arXiv.1709.01620.

[6] C. Fremerey, M. Müller, and M. Clausen, “Towards Bridging the Gap between Sheet Music and Audio.,” Jan. 2009.

[7] N. Fradet, J.-P. Briot, F. Chhel, A. Seghrouchni, and N. Gutowski, *MidiTok: A Python package for MIDI file tokenization*. 2021.

[8] I. Goodfellow *et al.*, “Generative Adversarial Nets,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2014. Accessed: Sep. 25, 2023. [Online]. Available: https://papers.nips.cc/paper_files/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html

[9] A. van den Oord *et al.*, “WaveNet: A Generative Model for Raw Audio.” arXiv, Sep. 19, 2016. doi: 10.48550/arXiv.1609.03499.

[10] L.-C. Yang, S.-Y. Chou, and Y.-H. Yang, “MidiNet: A Convolutional Generative Adversarial Network for Symbolic-domain Music Generation.” arXiv, Jul. 18, 2017. doi: 10.48550/arXiv.1703.10847.

[11] J. Wu, C. Hu, Y. Wang, X. Hu, and J. Zhu, “A Hierarchical Recurrent Neural Network for Symbolic Melody Generation,” *IEEE Trans. Cybern.*, vol. PP, Dec. 2017, doi: 10.1109/TCYB.2019.2953194.

[12] A. Vaswani *et al.*, “Attention Is All You Need.” arXiv, Aug. 01, 2023. doi: 10.48550/arXiv.1706.03762.

[13] B. Yu *et al.*, “Museformer: Transformer with Fine- and Coarse-Grained Attention for Music Generation.” arXiv, Oct. 30, 2022. doi: 10.48550/arXiv.2210.10349.

[14] W.-Y. Hsiao, J.-Y. Liu, Y.-C. Yeh, and Y.-H. Yang, “Compound Word Transformer: Learning to Compose Full-Song Music over Dynamic Directed Hypergraphs.” arXiv, Jan. 07, 2021. Accessed: Jan. 28, 2024. [Online]. Available: http://arxiv.org/abs/2101.02402

[15] P. Lu *et al.*, “MuseCoco: Generating Symbolic Music from Text.” arXiv, May 31, 2023. doi: 10.48550/arXiv.2306.00110.

[16] Q. Huang, A. Jansen, J. Lee, R. Ganti, J. Y. Li, and D. P. W. Ellis, “MuLan: A Joint Embedding of Music Audio and Natural Language.” arXiv, Aug. 25, 2022. doi: 10.48550/arXiv.2208.12415.

[17] S. Wu, D. Yu, X. Tan, and M. Sun, “CLaMP: Contrastive Language-Music Pre-training for Cross-Modal Symbolic Music Information Retrieval.” arXiv, Jun. 24, 2023. Accessed: Sep. 27, 2023. [Online]. Available: http://arxiv.org/abs/2304.11029

[18] M. Zeng, X. Tan, R. Wang, Z. Ju, T. Qin, and T.-Y. Liu, “MusicBERT: Symbolic Music Understanding with Large-Scale Pre-Training.” arXiv, Jun. 10, 2021. Accessed: Jan. 17, 2024. [Online]. Available: http://arxiv.org/abs/2106.05630

[19] Y. Liu *et al.*, “RoBERTa: A Robustly Optimized BERT Pretraining Approach.” arXiv, Jul. 26, 2019. Accessed: Jan. 20, 2024. [Online]. Available: http://arxiv.org/abs/1907.11692

[20] M. Saeed, “A Gentle Introduction to Positional Encoding in Transformer Models, Part 1,” MachineLearningMastery.com. Accessed: Jan. 28, 2024. [Online]. Available: https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/

[21] “Masking - Fast Transformers for PyTorch.” Accessed: Mar. 11, 2024. [Online]. Available: https://fast-transformers.github.io/masking/

[22] S. R. PhD, “Understanding and Coding Self-Attention, Multi-Head Attention, Cross-Attention, and Causal-Attention in LLMs.” Accessed: Mar. 11, 2024. [Online]. Available: https://magazine.sebastianraschka.com/p/understanding-and-coding-self-attention

[23] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, “Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention.” arXiv, Aug. 31, 2020. Accessed: Jan. 28, 2024. [Online]. Available: http://arxiv.org/abs/2006.16236

[24] M. Chandel, S. Silakari, R. Pandey, and S. Sharma, “A Study on Machine Learning and Python’s Framework,” *Int. J. Comput. Sci. Eng.*, 2022.

[25] P. Sodhi, N. Awasthi, and V. Sharma, “Introduction to Machine Learning and Its Basic Application in Python.” Rochester, NY, Jan. 06, 2019. doi: 10.2139/ssrn.3323796.

[26] A. Paszke *et al.*, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2019. Accessed: Feb. 03, 2024. [Online]. Available:

https://proceedings.neurips.cc/paper_files/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

[27] "Build the Neural Network — PyTorch Tutorials 2.2.0+cu121 documentation." Accessed: Feb. 03, 2024. [Online]. Available: https://pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html

[28] "PyTorch CUDA - The Definitive Guide | cnvrg.io." Accessed: Feb. 03, 2024. [Online]. Available: https://cnvrg.io/pytorch-cuda/

[29] "A detailed example of data loaders with PyTorch." Accessed: Feb. 03, 2024. [Online]. Available: https://stanford.edu/~shervine/blog/pytorch-how-to-generate-data-parallel

[30] A. Vyas, A. Katharopoulos, and F. Fleuret, "Fast Transformers with Clustered Attention." arXiv, Sep. 29, 2020. Accessed: Jan. 28, 2024. [Online]. Available: http://arxiv.org/abs/2007.04825

[31] "The Lakh MIDI Dataset v0.1." Accessed: Feb. 17, 2024. [Online]. Available: https://colinraffel.com/projects/lmd/

[32] C. Raffel, "Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching," 2016.

[33] J. Ens and P. Pasquier, "MetaMIDI Dataset." Zenodo, Jul. 28, 2021. doi: 10.5281/zenodo.5142664.

[34] L. Maršík, "Cover Song Identification using Music Harmony Features, Model and Complexity Analysis," 2019. doi: 10.13140/RG.2.2.20024.29449.

[35] A. Agostinelli *et al.*, "MusicLM: Generating Music From Text." arXiv, Jan. 26, 2023. Accessed: Sep. 25, 2023. [Online]. Available: http://arxiv.org/abs/2301.11325

[36] M. Production, “10 Words to describe Music | Musical Parameters & Categories,” Music Production HQ. Accessed: Feb. 03, 2024. [Online]. Available: http://musicproductionhq.com/words-to-describe-music-musical-parameters/

[37] M. S. Cuthbert and C. Ariza, “music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data,” 2010.

[38] K. Doshi, “Transformers Explained Visually (Part 3): Multi-head Attention, deep dive,” Medium. Accessed: Mar. 07, 2024. [Online]. Available: https://towardsdatascience.com/transformers-explained-visually-part-3-multi-head-attention-deep-dive-1c1ff1024853

[39] J. Brownlee, “Gentle Introduction to the Adam Optimization Algorithm for Deep Learning,” MachineLearningMastery.com. Accessed: Mar. 01, 2024. [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[40] J. Brownlee, “How to Configure the Learning Rate When Training Deep Learning Neural Networks,” MachineLearningMastery.com. Accessed: Mar. 01, 2024. [Online]. Available: https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/

[41] H. Sharma, “Softmax Temperature,” Medium. Accessed: Mar. 08, 2024. [Online]. Available: https://medium.com/@harshit158/softmax-temperature-5492e4007f71

[42] Rania _Hossam, “Sampling Methods in Text Generation Unlocking Diversity and Creativity,” Medium. Accessed: Mar. 08, 2024. [Online]. Available: https://medium.com/@raniahossam/sampling-methods-in-text-generation-unlocking-diversity-and-creativity-ab706b6250c9

[43] N. V. Otten, “Teacher Forcing In Recurrent Neural Networks (RNNs): An Advanced Concept Made Simple,” Spot Intelligence. Accessed: Mar. 12, 2024. [Online]. Available: https://spotintelligence.com/2023/10/12/teacher-forcing-in-recurrent-neural-networks-rnns-an-advanced-concept-made-simple/

[44] D. Dempster and M. Brown, “Evaluating Musical Analyses and Theories: Five Perspectives,” *J. Music Theory*, vol. 34, no. 2, pp. 247–279, 1990, doi: 10.2307/843838.

[45] S. Magazine and C. Schultz, “Why Do People Hate Dissonant Music? (And What Does It Say About Those Who Don’t?),” Smithsonian Magazine. Accessed: Mar. 15, 2024. [Online]. Available: https://www.smithsonianmag.com/smart-news/why-do-people-hate-dissonant-music-and-what-does-it-say-about-those-who-dont-120781501/

[46] M. Armstrong-López, “The Reason the Default iPhone Alarm Is So, So Terrible,” *Slate*, Oct. 15, 2022. Accessed: Mar. 16, 2024. [Online]. Available: https://slate.com/technology/2022/10/radar-iphone-alarm-apple.html

11) Appendix

11.1 Running Edited CLaMP model

1. Ensure the variables in the edited CLaMP file have been updated with the right file paths to the mxl folder.
2. Open Terminal
3. Navigate file directory to folder with edited CLaMP file
4. Run command: python clamp.py -clamp_model_name sander-wood/clamp-small-512 -query_modal music -key_modal text -top_n 30

11.2 Additional User Study Results

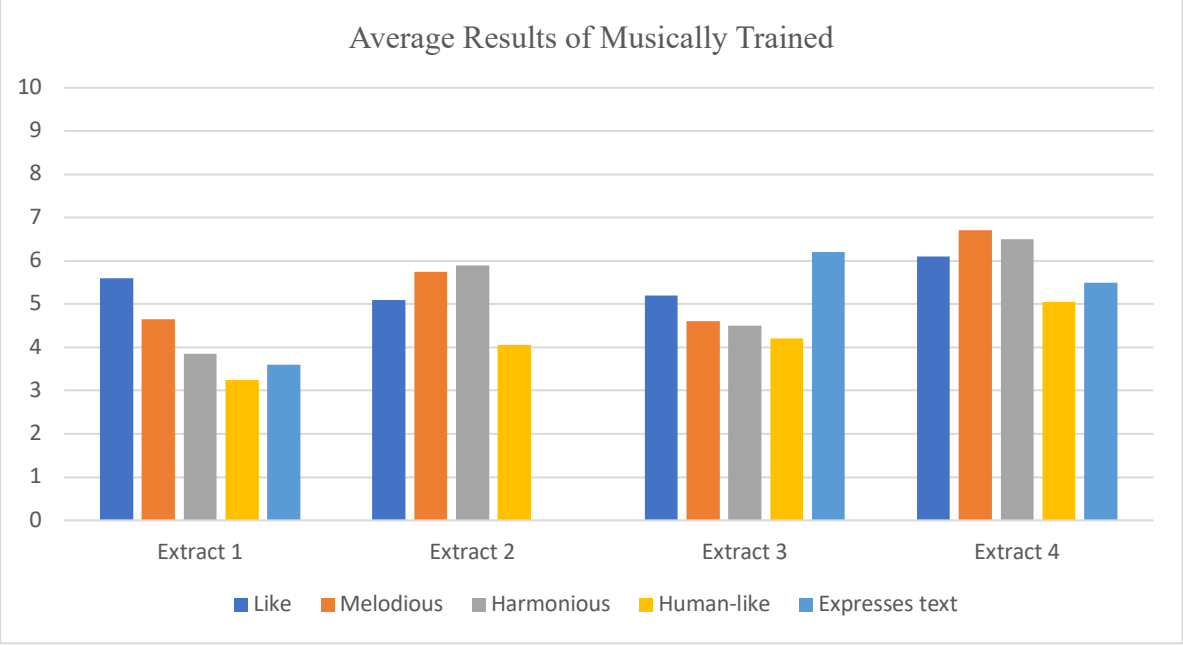

Figure 11.1 Average Results of User Study for 20 Musically Trained Users

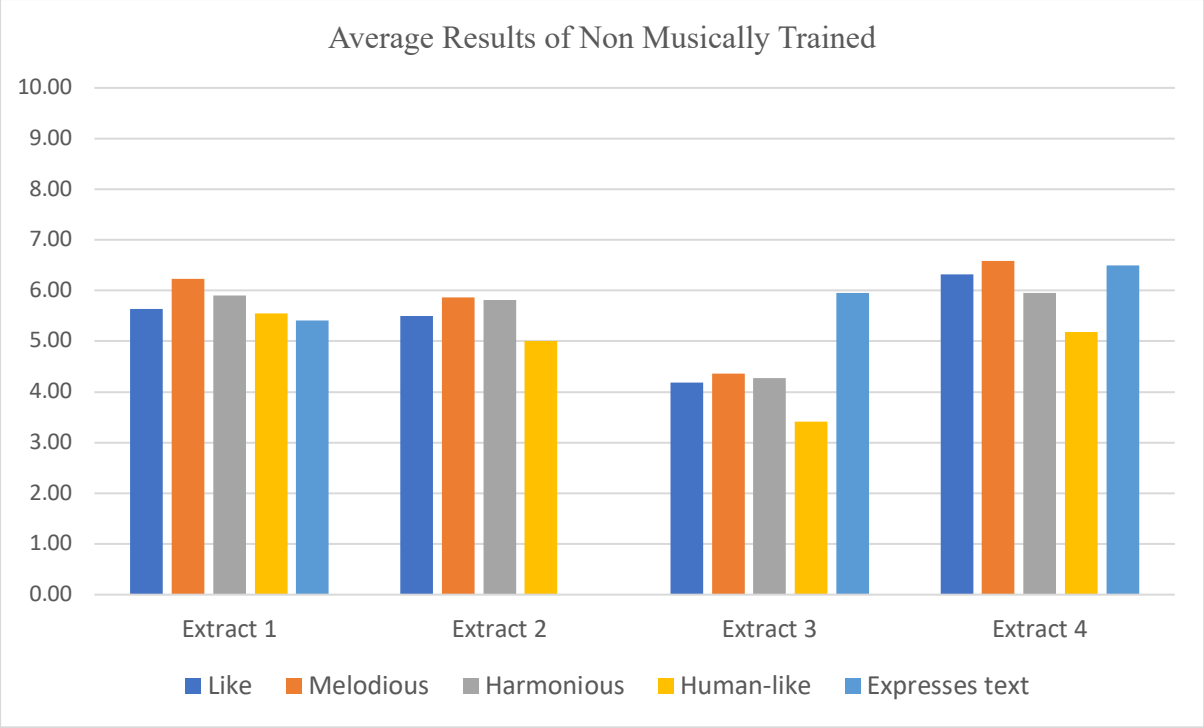

Figure 11.2 Average Results of User Study for 22 Musically Trained Users