Heriot-Watt University

F20PA Dissertation

Emotion Conditioned Music Generation

Author:
Megha Sharma

Supervisor:
Dr. Hani Ragab

A thesis submitted in fulfilment of the requirements for the degree of BSc (Hons) Artificial Intelligence

in the

School of Mathematical and Computer Sciences

April 2022

Declaration of Authorship

I, Megha Sharma, declare that this thesis titled, 'Emotion Conditioned Music Generation' and the work presented in it is my own. I confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: *Megha* Sharma

Date: 21st April, 2022

“ How is it that music can, without words, evoke our laughter, our fears, our highest aspirations?”

Jane Swan

Abstract

Music is a universal experience. Similar to how we speak our languages, we play our instruments and sing our songs to convey our emotions. The music production industry is dedicated to create music using instruments and voices, and with the recent wave of digitisation, they are adapting machines for better tools.

One such tool is automated music generation, which uses computer algorithms to generate music. In recent years, deep learning has become a popular approach in automating music generation. However, the generated music still lacks the quality we find in human-composed music. Some of the concerns include global coherence, long-term generation, and emotional impact

We plan to address this gap by conditioning the music generation with emotions. Our approach is to review existing models and propose a conditioned Transformer-GAN, that generates music input with an emotion. We investigated the effects of different music representation, architecture, hyperparameters and loss functions on the performance of the model. The model was evaluated against music metrics, negative log likelihood loss and a subjective tests with human participants. Our results find that our model is comparable to existing implementations, with a NLL of 1.2, lower than the unconditioned Transformer-GAN. Subjective tests indicate that the model has potential to generate music, however needs longer training sequences for effective musicality in the results.

We hope that in our pursuit, we can understand how we can make machine-composed music to sound more human-like, for a future where machines and humans will collaborate in producing music together.

Keywords: Music, Emotions, Artificial Intelligence, Deep Learning, Music Generation, Music Emotion Recognition

Acknowledgements

This dissertation is for everyone who works hard, despite their challenges. I dedicate my hard work to all of you, in no particular order. My family, whose support and optimism keeps me here. To my supervisor Dr. Hani Ragab, for his insights and motivation that help me become a better a writer and researcher. To my second reader, Dr. Alister Gray, for his feedback and support from the first deliverable. To my friends, for making it a bit more fun. To the IT staff, who tried their best despite the setbacks. To BTS, who continue to inspire and comfort me with their music. I'd also like to thank the public transport in Dubai, for its punctuality and its never ending traffic, which held me up when I did not have the strength to stand anymore. Lastly, I'd like to thank Burjuman, for allowing me to rest. Thank you all.

Contents

List of Figures

List of Tables

Abbreviations

AI	**A**rtificial **I**ntelligence
ML	**M**achine **L**earning
MIDI	**M**usical **I**nstrument **D**igital **I**nterface
CP Word	**C**om**P**ound Word
NLP	**N**atural **L**anguage **P**rocessing
MER	**M**usic **E**motion **R**ecognition
ANN	**A**rtificial **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
RNN	**R**ecurrent **N**eural **N**etwork
MER	**M**usic **E**motion **R**ecognition
LSTM	**L**ong-**S**hort **T**erm **M**emory
HVHA	**H**igh **V**alence **H**igh **A**rousal
LVLA	**L**ow **V**alence **L**ow **A**rousal
LVHA	**L**ow **V**alence **H**igh **A**rousal
HVLA	**H**igh **V**alence **L**ow **A**rousal
NLL	**N**egative **L**og **L**ikelihood

Dedicated to my family, always and forever

Chapter 1

Introduction

Music creates both personal and cultural identity for a listener [17]. At the same time, humans create emotional value around themselves and their society. Hence, it is reasonable to believe that the societal, cultural, and individual impact of music is also emotional. The music production industry is aware of this emotional impact of music, and producers often use music theory to compose sentimental pieces. With recent trends of digitisation and the growing reach of pop music, the industry looks forward to automate music production. The goal of this project is to automate music production using emotion as conditions during training and generating.

1.1 Motivation

Although it is difficult to reach a consensus on the origin and meaning behind emotions arising from music, it is widely agreed that music is emotional [18] [19] [20]. However, current state-of-the-art approaches such as the Music Transformer [12] and Transformer-GAN [10] that generate music using deep learning techniques do not employ sentiments in their music theory. In an effort to create more natural and impactful music with Artificial Intelligence (AI), we aim to use deep learning to condition emotions in music generation models. Hung et. al proved early success of conditioning with emotions using their own compiled dataset with a Transformer model [7]. Our approach is infusing the techniques of conditioning emotions in the current state-of-the-art Transformer-GAN architecture. In this architecture, the generator acts as a music generation model while the discriminator detects how realistic, or how close the music is to real samples. Muhamed et al. used advanced transformers such as Transformer-XL and BERT as the generator and discriminator respectively. However, given the resource block in the second half of the semester, as well as our GPU limit (explained further in section 5.3), the model

could not be scaled with the same hyper parameters and complex architecture as found in the literature. This gave us an opportunity to build a simpler Transformer-GAN model from scratch that meets the same objectives with the given computation resources we had. Our model adds another layer of complexity where the GAN is conditioned on the input emotion. We also looked into the effects of different representations, datasets and compared the scale of the performance with the existing approached [7] [10]

1.2 Aim

Conditioned music generation is a subset of music generation models in AI that allows an end user of the system to input a condition for the generated music. Our aim is to **develop a deep learning model that can generate pop music conditioned on emotions input by the end-user**. This model will be independently analysed, and compared to existing methodologies, using both objective metrics and subjective human reviews.

The overall goal of this project is develop an effective model that can assist music producers and composers in the music production process.

1.3 Objectives

The objectives identified for the project are as follows:

- Identify suitable deep learning approaches in music generation and music emotion recognition to design a GAN model where the generator is a music generation model and the discriminator is a music emotion recognition model
- Develop the GAN model to generates music conditioned on the input emotion
- Investigate the influence of different music representations and data sources on the quality of generated music
- Compare and analyse the musicality of the proposed model with existing approaches, more precisely those proposed by [10] [7] [12]
- (Optional) Develop a methodology to inpaint (join 2 pieces of) music based on emotion and analyse the results

1.4 Manuscript Organisation

The organisation of the manuscript is designed to flow from the chapter Background and Related Work covering the relevant knowledge and the literature review. Then, we go through the implementation of the project in Project Implementation. We discuss our results from our implementation in Results and Evaluation. Lastly, we cover the Conclusion which concludes the project, reiterating the achievements, limitations and future work.

Chapter 2

Background and Related Work

This chapter focuses on the definition and explanation of the main concepts that will be referred to in the remainder of the paper. Topics covered include music theory, emotion and music representation, deep learning fundamentals, and the literature review. For further reading on deep learning and music, a more detailed overview on the same concepts in the appendix Background Details

2.1 Music Theory

Music theory not only helps us quantify the information on music, but is often the basis of an alternate representation such as those discussed later in the section 2.2. This section builds from the fundamentals to the structures that define music, starting from the notes to the notation of a musical piece [21].

2.1.1 Fundamentals in Music Theory

- ***Note***: Sound is measured as pitch in Hertz (Hz). A note is the symbol assigned to specific pitches, labelled from A to G. It is the token representation of sound, storing accent, dynamics, frequency, duration of sound and more.
- ***Accent***: The emphasis (intensity) of a note
- ***Dynamics***: The volume (loudness) of a note.
- ***Chord***: Groups of three or more notes played simultaneously. A chord progression is a sequence of chords.

- ***Beat***: An unheard and steady pulse behind the music that acts as ruler for the placements of the notes.
- ***Rest***: A pause in the music between notes for a specific duration
- ***Tempo***: The speed of music, measured as beats per minute (BPM)
- ***Bar***: A segment of time in music. Each bar has a set number of beats
- ***Time Signature***: A fraction where the top is the number of beats in a bar and the bottom is the duration of each beat. For example, for the time signature of 4/4, we say the piece of music has 4 beats in each bar, and each beat has a duration of 1/4 or a quarter.

2.2 Music Representation

Music representation is a field of representing musical features in audio or MIDI files as input for Machine Learning (ML) models. MIDI (Musical Instrument Digital Interface) files store music as a set of instructions that can be played on a machine, which reduces the storage need compared to audio files [22]. The following music representation can be divided into 2 types: raw and symbolic. Intuitively, the types have a trade-off between human and machine comprehension. Raw representation is intuitive for humans however needs processing for machine learning, whereas symbolic representation may not be intuitive however it is simpler to feed into ML models. In this project, we only deal with Symbolic representations as these are derived from MIDI files, our source data.

2.2.1 Symbolic Representation

This is a notation of music stored as a sequence or matrix of events. Symbolic representations are extracted from MIDI files, some key representations are given below:

- **MIDI-Like Encoding**: Inspired from MIDI, a MIDI-Like encoding stores music as a sequence of events [23]. The tokens defined in this encoding represent Tempo, Note On, Note Off, Velocity and Time Shift. While it is computationally cost effective, it lacks an explicit relationship between simultaneous notes [24].
- **REMI Encoding**: Huang and Yang introduced the REvamped MIDI-derived events (REMI) encoding to improve on the MIDI-Like encoding [1]. Unlike MIDI, REMI encodes chords explicitly, and divides the music with tokens representing a musical bar like a musical score. The tokens used in REMI are shown in fig. 2.1.

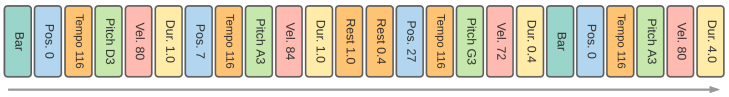


FIGURE 2.1: REMI Encoding [1]

- **CP Word Encoding**: CP Word (Compound Word) is a more complex event representation [2]. Here, each event actually contains 6-8 dimensions, where each dimension represents a feature of the event. As seen in fig. 2.2, each event contains separate tokens for family (Note or a metric), pitch, velocity etc. This means there's less number of events as tokens are processed together, however it expands the corpus to k-dimensions where k tokens are used to describe an event [25]. The first token identifies the type, for eg. Note events only use Pitch, duration and velocity, while Metric events use chord, rest, and tempo information.

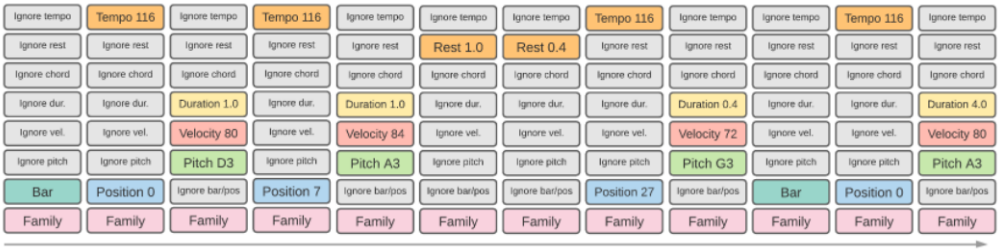


FIGURE 2.2: CP Word Encoding [2]

- **Lead Sheets**: Makris et al. introduced a format based on Lead sheets, which are special musical scores where a melody is annotated with the chord symbols and lyrics [26]. This representation encodes each bar, storing information like time signature, valence and density. Makris et al. used this as an input sequence, and decoded it into melody, chord and duration information for the next bar.

- **Piano Roll**: Piano roll is a matrix-like representation that can support polyphonic music [27] [28]. A cell in the matrix of time and notes stores tempo, which creates tabs of notes over time as seen in fig.2.3. Without any information on rest, an additional pause of one-time step is added between two consecutive notes in the same pitch to play as two separate notes instead of one. Sometimes additional matrices are used for separate information, eg. dynamics.

2.3 Emotion Representation

Representing human emotions objectively is a continuous work of improvement. The lack of ground truth is an obvious difficulty, however we can be inspired from the research done in psychology in developing a model for emotions [7].

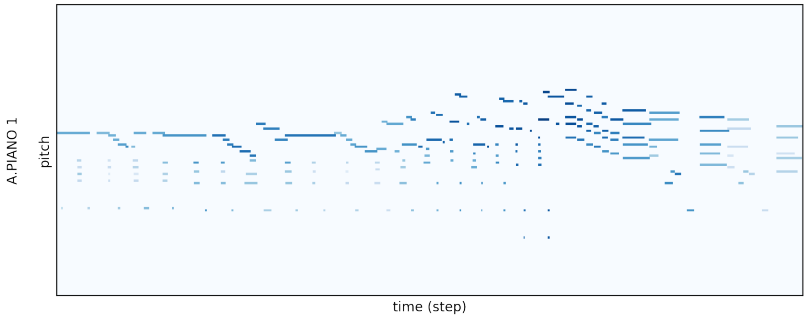


FIGURE 2.3: Piano Roll Representation [3]

Some psychologists approach emotion as distinct categories. In 1990s, psychologist Paul Ekman listed six basic human emotions, eventually expanding to over 15 [29] [30]. Although this model still persists, it faces the risk of subjectivity.

Some scientists prefer mapping emotions on a multi-dimensional plane. One very popular approach is referred to as the Russell's 4 Quadrants (Russell's 4Q) [31]. Russell described emotions with two dimensions: valence (whether the emotion is positive or negative) and arousal (whether the emotion is intense or not). In fig 2.4, we can see how emotions are placed in different quadrants. Russell's 4Q can also be categorised as discrete quadrants. The four categories are: HVHA (High Valence High Arousal), LVHA (Low Valence High Arousal), LVLA (Low Valence Low Arousal), and HVLA (High Valence Low Arousal). Other dimensional models include Plutchik's 3 dimensional wheel of emotions, with increasingly complex emotions in larger circles as shown in fig. 2.5 [32] and a 3-dimensional model based on pleasure, arousal, and dominance by Mehrabian and Russell, which adds another independent dimension to Russell's 4Q [33].

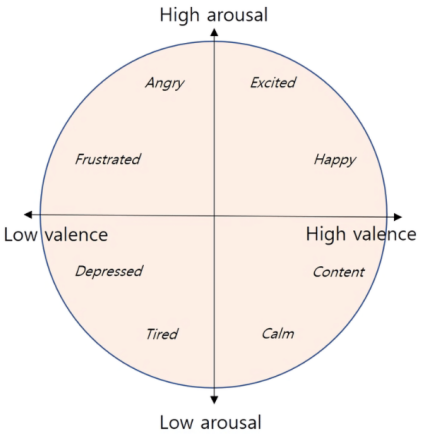


FIGURE 2.4: Russell's 4 Quadrants. Figure from [4]

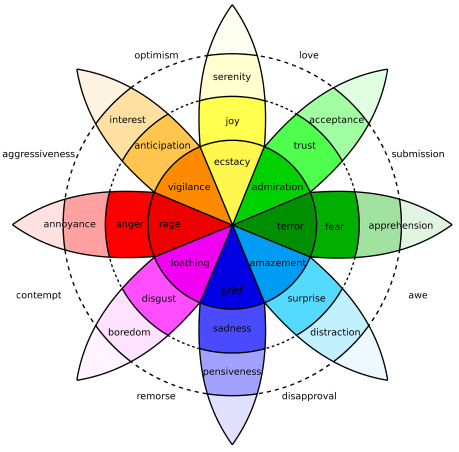


FIGURE 2.5: Plutchik's model, a wheel of emotions

2.4 Deep Learning Fundamentals

Traditionally, computer algorithms are rule based programmes that define output by a set of rules. Machine learning (ML) uses computer algorithms to study data to define output by a "learnt" pattern [34]. Deep Learning is a subset of ML that uses Artificial Neural Networks (ANN) to imitate human-like learning. Learning can be of the following types:

- **Supervised Learning**: The model learns to match the output to features of the input [35]. Types of supervised learning techniques include classification algorithms, regression algorithms etc.

  In classification algorithms, the output is a set of discrete values. The algorithm can learn to classify an input to one or more such values [34]. For eg. we can use a classification algorithm to classify music input as a category of emotion(s). For our project, we will apply classification algorithms to recognise the emotion of a piece of music, as well as how critic the music as real or fake. The performance of these models depends highly on the input it is taught which we will explore in the Datatset Analysis

- **Unsupervised Learning**: The model learns patterns from the data only [34]. In the real world, most data is unlabelled, hence, these algorithms are powerful in learning to create labels by grouping data by patterns into discrete clusters.

In this section, we will detail few Artificial Neural Network (ANN) techniques that are relevant to music generation and music emotion recognition, including Recurrent Neural Networks, Convolutional Neural Networks, Transformers and Generative Adversarial Network.

2.4.1 Recurrent Neural Networks (RNN)

Recurrent Neural Networks use cyclic (recurrent) paths in the hidden layers to learn the context withi[36]. Their core advantage is the concept of "memory" which arises by sharing the weights in the hidden layers across time steps. Hence, they work well with sequential data like music and text, which are understood by their context. However, RNN often face the issue of vanishing gradients (very small gradients) and exploding gradients (very large gradients) with longer sequences [37]. To combat the issue of memory, Hochreiter and Schmidhuber introduced the Long Short Term Memory (LSTM) architecture [38]. An LSTM unit is composed of a cell with three gates. The cell

controls the flow of data with the gates, with selective memory to only remember relevant information.

2.4.2 Convolutional Neural Networks (CNN)

Convolutional Neural Networks are a different type of ANN that learn from matrix or image-like data [39]. They are generally used in the field of computer vision, however certain music representations like spectrograms and piano rolls can also be used with CNN [40].

2.4.3 Transformers

Transformers are a unique type of deep learning model, borrowing concepts from RNN and CNN for language modeling tasks [5]. Introduced in 2017, Transformers have been successfully used with sequential data.

The authors introduced the attention mechanism to develop long term memory [5]. Traditionally, encoder creates a vector representation of all the tokens in a sequence. With attention, we also create a vector representation of every individual tokens. An attention module then uses this to calculate the following attention score:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Where Q is the query (vector representation of current token), K is the matrix of keys (vector representation of all tokens), and V is the value (the weighted average vector representation of current token). The transformer architecture includes an encoder and a decoder with attention, visualised in fig. 2.6

2.4.3.1 Encoder

The encoder works as follows: At *Input Embedding*, each token is embedded as a vector. Since the data is not sequentially processed, the positions of the tokens are explicitly encoded at *Positional Encoding* and added to the token embedding [5]. This is fed into a *Multi-Head Attention* module, which computes multiple self attention heads in parallel. Each attention head calculates the attention score defined above, and updates three parameter weights: *Query*, *Key* and *Value*. The output of this module, the attention head, scores how much the current token should pay attention to the previous ones. The

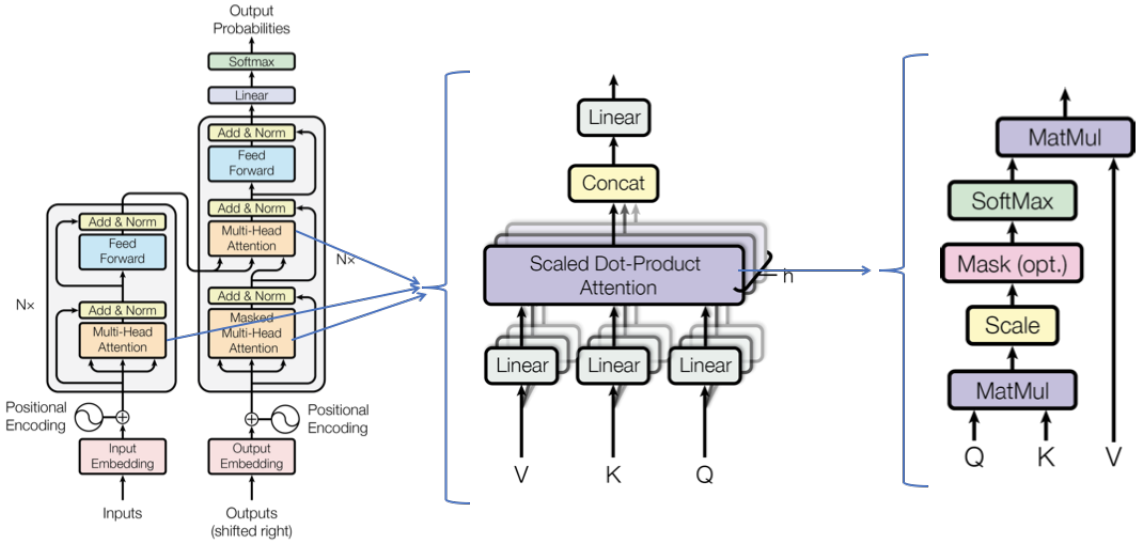


FIGURE 2.6: The transformer architecture, with focus on attention [5]

outputs are merged and fed into the *Feed Forward* layer. The output from this layer is used in the decoder.

2.4.3.2 Decoder

The decoder works as follows: The decoder retains the previous output, and creates an embedding and encoding similar to the encoder. This is fed into the first *Multi-Head Attention* layer. However, this module is modified to only pay attention to the previous tokens of the input sequence, as out the output can only depend on the previous tokens of the output sequence. This is done by masking succeeding tokens with a *Mask* matrix product on the vector representations. In the second *Multi-Head Attention* layer, the *Keys* and *Values* are the output of the encoder, while the *Queries* are from the previous *Multi-Head Attention* layer. Similar to the encoder, the outputs from the multiple attention heads are aggregated and fed into a *Feed Forward* layer. The final output is the next predicted output token by the model. This continues until all input tokens are processed.

2.4.4 Generative Adversarial Networks (GAN)

Generative Adversarial Networks or GAN are generative deep learning models introduced in 2014 [41]. They consist of two deep neural networks in an adversary (competition) with each other. These are dubbed as the Generator and the Discriminator. The generator generates new data from random noise, and the discriminator learns the real data to determine whether the input is a real sample or not. The aim of such networks is

to create realistic data that cannot be distinguishable from the training samples. Both the generator and the discriminator can be any type of neural network, including RNN, CNN and transformers. The model is visualised in fig. 2.7

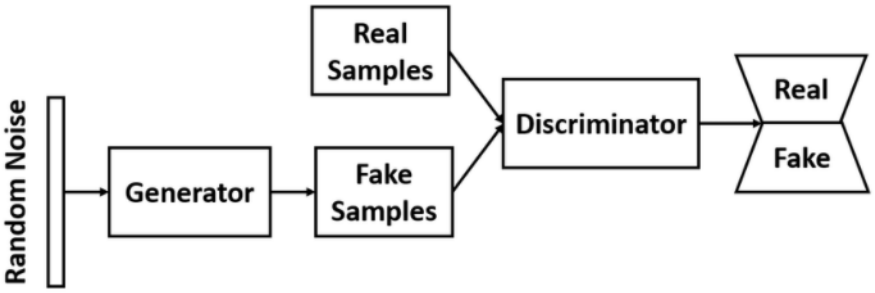


FIGURE 2.7: The GAN architecture [6]

2.5 Datatset Analysis

Considering the aim of the project is to generate emotion-conditioned music, our focus on datasets was to find music clips annotated with their emotion. The following datasets were reviewed:

- **MooDetector Dataset (2013)**: The dataset consists of 193 songs from multiple genres stored in three modes: audio, MIDI and lyrics [42]. Each song is classified as one of five categories, adapted from MIREX competition categories [43]. Music professionals annotated the music clips.
- **Jamendo Dataset (2019)**: This dataset is made for music auto-tagging to predict moods, genres and instruments from raw audio [44]. With over 55,000 tracks of audio, 59 unique tags are identified. The annotations are taken from Jamendo, which is a website that hosts free music in all genres.
- **VGMIDI Dataset (2021)**: This dataset is used for generating sentiment-conditioned music, and features 966 clips of MIDI files used in video games [8]. Volunteers annotated the valence in the music clip, and these annotations are smoothed and clustered to remove noise annotations. The publicly available dataset hosts 200 clips.
- **EMOPIA Dataset (2021)**: This dataset features over 1000 clips of piano covers of Japanese, Korean and western pop music [7]. A few authors annotated the dataset into 4 quadrants of Russell's 4Q [31], while others cross validated. The clips were pre-processed and encoded as different music representations such as MIDI-like and REMI to test for music emotion recognition and music generation.

2.5.1 Critical Analysis

Panda et al. annotated the MooDetector dataset in multiple modes [42]. They noted that lyrical data needs extensive feature extraction to create any valuable effect in music emotion recognition, preferring audio and MIDI instead. We primarily searched for datasets with MIDI files for two reasons: (1) In order to stay consistent with music producers who usually use MIDI files and (2) MIDI files are generally under the CC-BY license which allows us to edit and distribute them, unlike audio which is generally copyrighted. We also searched for scientific emotion classes for consistency. The MooDetector and the Jamendo dataset are annotated with descriptors set by professionals or content creators. The quality of these categories cannot be verified or formalised. On the other hand, VGMIDI and EMOPIA annotated the data with Russell's 4Q [31] and these annotations were analysed and pre-processed. VGMIDI only contains 200 examples, though annotating both valence and arousal. We choose EMOPIA as our primary dataset for the following reasons: (1) Russell's 4Q categories is scientific and easier to define for participants in the subjective tests, (2) Pop piano music standardises the genre and instrument for our model (3) Clip-wise annotations account for emotion variation in songs, and (4) it contains 1070+ clips alone. However, we are still interested in examining the results from training other discussed datasets, which we will discuss in Results and Evaluation.

2.6 Music Emotion Recognition (MER)

Music emotion recognition is a difficult task to model, even for human beings [45]. Researchers find it difficult to summarise and identify state-of-the-art ML approaches without knowing how and which features affect emotions in music. This issue was recognised in 2012 [45], and the field has grown even more complex and difficult to organise ever since. Hence, to simplify the organisation, we approach the review based on two approaches: Regression-based and Classification-based approaches

2.6.1 Regression-based Approach

Malik et al. stacked a CNN and RNN to predict the emotions of a 30 second piece of music as numerical values of valence and arousal [46]. The model is trained on two type of inputs: baseline (vector of 65 features such as dynamics) and raw (vector of 60 spectrograms of every 500 ms). The CNN returns a dense vector representation of the input, which is fed as input for the RNN. Two RNN are used for valence and arousal separately, and the authors found better results with 1200 times less parameters

compared to previous approaches, reporting the root mean squared error on valence and arousal as 0.268 and 0.202 respectively.

2.6.2 Classification-based Approach

Su et al. aim to develop MER for music retrieval systems [47]. They divided the CAL500 dataset [48] into seven emotions: Sad, Romantic, Happy, Exciting, Light, Calm and Angry. The model is a CNN stacked on a Support Vector Machine (SVM). Raw audio is segmented, and the spectorgram of each segment is input to a CNN (VGG-16). The CNN outputs a new data representation which is then fed into the SVM model. The SVM model outputs a probability for each of the seven emotions, which is fitting to the subjectivity of emotion classification. The model resulted in better F1 score compared to other approaches including K-Nearest Neighbours, Random Forest, and vanilla SVM. However, the model only gained marginal improvements over the vanilla SVM.

Hizlisoy et al. also used CNN to learn high level features from raw audio [49]. They collected the "most salient part" of 124 Turkish songs and volunteers annotated the clips with Russell's 4Q [31]. The authors extracted features from both the raw audio (with open-source tools) and the spectrogram of the audio (with a CNN model to return a vector representation). Since there is no true consensus on how and which features affect our emotions, usually all features are used. This can be computationally expensive, especially with spectrograms. Hence, the authors applied correlation-based feature selection on the combined feature set (the raw audio features and the vector representation). The selected features then go through an LSTM and two fully connected layers to predict the emotion. The dataset only had samples from HVHA, LVHA, LVLA. The authors reported an accuracy of over 99% with this approach. The high accuracy can be attributed to the data, which is both limited in size and classes. An SVM achieved an accuracy of 98.6% on the same dataset, and there are no samples from the class HVLA.

Zhao et al. classified emotions in MIDI files using a simple RNN [50]. 15 students categorised MIDI files on YouTube in five classes: Aggressive, Bittersweet, Happy, Humorous, Passionate. The authors extracted the melody trend as a tuple of the note type (quarter, one-eighth etc) and pitch. After the hyper-parameter tuning, the model reached a final accuracy of 75.4% on the dataset.

Hung et al. studied emotion recognition on their EMOPIA dataset for both symbolic and raw audio [7]. For raw audio, they transformed three seconds of music to a vector representation of the spectrogram. This was input to two models: Logistic Regression Classifier and a CNN-based model. For symbolic audio, the MIDI files were encoded as

MIDI-Like and REMI representations for a LSTM and a feature vector for Logistic Regression Classifier. The results indicate that deep learning with symbolic audio performs the best as the LSTM with attention and MIDI-Like encoding gave the highest classification accuracy on the four quadrants (68.4%), although valence ubiquitously performed worse for all models. This is a recurring issue in MER [51] [52]. However, MIDI-Like classified valence better than REMI encodings. Chou et al. also worked on the same dataset, training a BERT classification with REMI and CP word encoding for emotion classification [16]. The paper reported a test accuracy of 67.89%, which outperformed their BiLSTM with Attention model (54.13%). However, even the BERT model faced confusion in identifying valence

2.6.3 Critical Analysis

A summary of the reviewed models in table 2.1 highlights the music and emotion type and other relevant results. We could only find regression approach on MER with raw audio. Hence, we cannot perform any valuable comparison with symbolic music emotion recognition. Based on Malik et al. approach, we can integrate simple models to create better results. However, we face two main issues in the approach. The first is the number of features required for raw audio. Raw audio is already computationally heavy to store, and the proposed model input 65 features and 60 spectrograms for 30 second clips. Secondly, the model output is not intuitive for human listeners. Humans generally use adjectives to describe music while listening and producing. We want to follow this intuition of mapping emotions with discrete categories. This can allow us to support music producers with more human-friendly descriptors. Hence, we decided to further review other classification-based approaches instead.

In classification-based approaches, raw audio is the preferred mode for music with CNN-based models. Raw audio representations are usually matrix or image-like, which is a suitable input for a CNN. In our review, CNN was only used to extract features from the audio, and did not perform any classification. On the other hand, even the simple RNN by Zhao et al. with symbolic audio input effectively classified emotions. They only used the pitch and note information from the MIDI files to reach an accuracy of above 75%. However, their approach classified emotions into broad categories which cannot be systematically defined. The EMOPIA paper gave a more systematic evaluation of the results on their dataset, using both symbolic and audio music files. Symbolic audio performed better on the dataset, which is encouraging as we plan to use this dataset primarily. We plan to use models like the LSTM model and the BERT Classifier by Hung et al. [7] and Chou et al. [16] respectively as our guide for the MER model and experiment with other RNN-based architectures for the project. The accuracy of valence

across all models is lesser than arousal, hence it is important to observe how different encoding can affect valence. In our project, we plan to analyse the performance of the emotion quadrants with different music representations.

Model	**Emotion Type**	**Music Type**	**MIDI / Audio**	**Metric**
Malik et al. [46]	Numerical Valence and Arousal	Multiple Genre	Audio	RMSE: 0.202 for arousal and 0.268 for valence
Su et al. [47]	Seven emotion types	Western Pop	Audio	∼0.66 F-measure on top 3 emotions
Hizlisoy et al. [49]	Russell's 4Q	Turkish	Audio	0.992 F-measure and 99.2% Accuracy
Zhao et al. [50]	Five emotion types	Multiple Genre	MIDI	75.4% Accuracy
Hung et al. [7]	Russell's 4Q	Pop	Both	68.4% Accuracy on MIDI

TABLE 2.1: Summary on Music Emotion Recognition Models

2.7 Music Generation

The interest in generating music with AI is even more apparent with the increasing consumption of music [53]. More people are listening to music, as a result, demand of new music has also increased. The music production industry is going through a wave of digitization and automation itself, and with advancements in machine learning, they look towards the field of AI to optimise the production process [53]. This section divides symbolic music generation in two types: unconditioned and conditioned music generation.

2.7.1 Unconditioned Generation

Unconditioned generation is essentially generating music "from scratch". The models are trained on a given dataset, and the generated music learns a similar style with the training dataset.

Transformers are a popular generative model. The Music Transformer is an adaptation of the transformer model that aims to generates long-term coherent music [12]. The model specifically uses relative attention to measure the distance between two positions in a sequence. It is trained on two datasets: Bach's chorales and piano-e-competition [23]. The former was fed as a matrix of note and time while the latter was encoded as MIDI-Like encoding. The Bach output generated more global coherence using local relative attention compared to the baseline transformer and gave better loss than other CNN-based models with validation negative log likelihood (NLL) loss at 0.335. The Piano-e-competition output returned state-of-the-art results on the dataset compared to other RNN models with NLL at 1.84.

Inspired from the Music Transformer, Huang and Yang built a competitive model dubbed Pop Music Transformer that generates pop piano music with better results compared to the Music Transformer [1]. The first improvement is the music representation: The paper introduces the REMI encoding, which encodes more information on the MIDI music files including tempo, chords, and most effectively, bars and positions. The bars explicitly add temporal information. The second improvement is the additional techniques used with transformer variant Transformer-XL [54]: Recurrence Mechanism and Relative Positional Encoding. The model adapts recurrence from RNN to learn from the previous segments in a sequence. Here, each MIDI sequence is divided into fixed-size segments. Since segments are no longer independent of each other, relative positional encoding is used to encode the positions of different segments in one layer. 775 piano covers of pop music were taken from YouTube in the genres of Japanese anime, Korean and western pop songs to train the model. The results were compared to a Music Transformer adapted with Transformer-XL. The authors proved that position and bar tokens can successfully create coherent rhythm using objective and subjective tests. The objective metrics included standard deviation on the beat, which was 0.0386 for their model. This was closer to the training data (0.0607) than Music Transformer (0.0968)

Other generative models like GAN are also popular for music generation. Dong et al. introduced MuseGAN to generate multi-track (polyphonic) piano rolls of music [27]. Here, each track is an instrument. The authors convert MIDI files of pop and rock music into piano rolls as input for a hybrid model with a generator for each track but only one discriminator to decide if the combined multi-track music is real or not. The model incorporates temporal information by generating bar by bar. Although the generated music still falls behind human scope, the single discriminator performs well enough to generate some rhythmic cohesion. Non-professionals in music gave an overall rating of 3.16 out of 5 in the subjective human study. Dong et al. attribute the poor musicality of the music to the excessive fragmentation of notes in the generated binary piano roll. The paper shows that simple note features such as duration and pitch encoded in binary piano rolls are not enough to imitate human-composed music.

A hybrid approach with transformers and GAN is the Transformer-GAN [10]. Muhamed et al. use Transformer-XL as generator and a pre-trained variant of a transformer called spanBERT [55] as the discriminator. The authors argue that BERT-based predictions are correlated to human predictions, which is desirable in a discriminator. A dataset of classical music is encoded in MIDI-Like format and used to pre-train the discriminator and train the model. The final results were comparable to the Music Transformer. In subjective tests, 57% of the people preferred the music generated with the Transformer-GAN compared to the Music Transformer. The music generated with the spanBERT discriminator is the most similar in music metrics like pitch range to the training samples.

2.7.2 Conditioned Generation

These models allow user interaction by defining condition on style, genre and emotions.

- **Conditioned on Rhythm**: DeepBach is an interactive approach for generating Bach's chorales by allowing users to condition input on a rhythm [56]. Users can input a part of the chorale for the model to regenerate. Such interactivity is not possible on vanilla left-to-right RNN, hence, Hadjeres et al. use Bi-LSTM that can learn in both directions. A Bach's chorale consists of four voices, which the authors encode as metadata (key, time signature etc) and sample parts of the chorale as input. The model predicts the current note for one voice based on the current notes of other voices and previous notes of all voices. Over 50% of 1272 listeners believe that the chorales are composed by Bach himself. While this system is novel in its approach and results, the architecture fails to capture changing temporal and dynamic information, since it only encodes the rhythm as with hold tokens.

- **Conditioned on Style**: DeepJ aims to improve on DeepBach using a Bi-Axial LSTM to generate polyphonic music from different classical composers [57]. Bi-Axial LSTM stacks a time-recurrent and a note-recurrent LSTM to predict the next note based on all generated notes in the current time-step (note-axis) and all generated notes in previous time-steps (time-axis). The authors augment a piano roll of the music with a many-hot encoding of style(s). This representation does not account for similarities between styles, yet effectively produces stylistically similar music. The authors also augment the input with a matrix for dynamics and hold to improve musicality. In the subjective tests, around 70% of 300 participants prefer DeepJ's compositions over Bi-Axial LSTM. The authors report that dynamics aids the quality of generation, however, the music still lacks long-term coherence.

- **Conditioned on Emotions**: Zhao et al. address the poor long-term coherence and generate emotion-conditioned symbolic music by improving the BiAxial LSTM (BALSTM) in DeepJ [58]. The model architecture is updated with a LookBack layer and silence temperature. LookBack learns from the previous time step and silence temperature monitors the duration of silence. The input is a note-time matrix like a piano roll, where each cell contains a tuple like (note is playing, note is articulating, dynamics of note). For training, a one-hot encoding of emotions is augmented to the input, where the emotion categories are Russell's 4Q [31]. For testing, only the emotion vector is input. The proposed model shows better negative log likelihood loss (4.22) than the baseline BALSTM (4.9) using the piano-midi dataset. Subjective tests reveal that valence was difficult to distinguish.

Hung et al. also evaluate symbolic emotion-conditioned music generation on the EMOPIA dataset [7]. An LSTM with attention mechanism and a pre-trained transformer are trained. The latter performs better on objective and subjective metrics, creating consistent rhythm. The authors report an accuracy of 41.8% for the transformer and 23.8% for the LSTM. Neither model can reproduce low valence music well.

Makris et al. introduce a novel method of calculating valence from chords [26]. Chords can be of different modes: major and minor. Generally, major chords sound "happier", while minor chords sound "sadder". While these distinctions are largely cultural in western music [59] [60], Makris et al. successfully map the modes of the chords to emotions to improve the valence of generated music. They generate emotional chord progressions in symbolic music with two models: LSTM-based encoder and decoder and a transformer. The models are trained on a lead sheet representation of the VGMIDI dataset [8]. The transformer has more coherence according to the results in both objective and subjective tests, however even the best generations failed to capture the naturalness of music composed by humans by not focusing on rhythm.

2.7.3 Crticial Analysis

As seen in our summary in table 2.2 with the relevant results of the models explored here, there are no standard evaluation metrics for music generation (and music emotion recognition) models. Hence, we developed our review by focusing on the improvements made in comparison to prior models. Generative models are the preferred approach for generating unconditioned music. Transformer models give us flexibility on input type over CNN and RNN models. Huang and Yang successfully use matrix-like and sequential music representations with transformers to learn rhythmic structure and temporal information. Given the many type of representations in music, we view transformers as advantageous for our project. Our view is strengthened in the survey, where the Pop Music transformer outperforms other models, making it suitable for our objectives. Moreover, it is trained and tested on pop music from the same background as the chosen EMOPIA dataset [7]. However, our project advances from unconditioned generation and can benefit from GAN-based architectures to teach the model to discriminate between different emotions. While the MuseGAN fails to generate complex harmonies, even a single discriminator can effectively discriminate between real and generated rhythms from multiple generators. Hence, we look forward to harnessing the power of discriminators in GAN to generate emotionally cohesive music. The Transformer-GAN architecture

integrates these models, and can be the baseline for our project. The model is also noted as future work on the EMOPIA dataset.

Conditioned generation techniques help us learn how we can add conditioning in the input. The DeepBach input is too simple and the lack of features leave a noticeable gap between the actual and produced music in other genres. DeepJ tries to overcome these disadvantages, and their many-hot representation of style is similar to the one-hot encoding of emotions used in later models. Both models continue a recurring theme in long term generation: RNN and its variants often fail to capture the long term dependencies in musical events. Long-term generation is important with music because unlike text, a single token cannot describe anything, even a single note is encoded as multiple tokens. Zhao et al. and Hung et al. use additional techniques with LSTM such as LookBack and attention for long-term generation. However, we find that results by Hung et al. and Makris et al. using transformers outperform LSTM for coherent emotion-conditioned generation. All models unanimously face difficulty in generating valence-rich music, similar to the difficulty of modeling valence in MER. Makris et al. make a valuable contribution of detecting valence in music by focusing on chords. However, despite the improvement in valence, the model fails to capture rhythm and musicality compared to human composers. We observe a research gap here, as the performance of transformers with chord progression to improve valence and variant architectures to improve long-term rhythmic cohesion is not reported. Hence, we plan to integrate existing work with transformers with the new Pop Music Transformer to generate emotion-conditioned symbolic music. Similar to existing work, the emotions can be augmented as a single one-hot representation encoding.

Model	**Architecture**	**Music Type**	**Condition**	**Metrics**
Music Transformer[12]	Transformer	MIDI	N/A	Validation Negative Log Likelihood: 0.335 for chorales and 1.84 on piano
Pop Music Transformer[1]	Transformer	MIDI	N/A	Standard Deviation on beat: 0.0386 and 0.0968 on Music Transformer
MuseGAN[27]	CNN-GAN	MIDI	N/A	Best Overall Rating by humans: 3.16 / 5 for Hybrid model
Transformer-GAN[10]	Transformer-GAN	MIDI	N/A	Classification Accuracy: 81.8% by discriminator
DeepBach[56]	LSTM	MIDI	Rhythm	Subjective Tests: ~50% believe it is composed by Bach
DeepJ[57]	Bi-Axial LSTM	MIDI	Style	Subjective Tests: Over 70% prefer over Bi-LSTM
Zhao et al.[58]	BALSTM+LookBack	MIDI	Emotions	Negative Log Likelihood: 4.22 and 4.9 with DeepJ
Hung et al.[7]	LSTM and Transformer	MIDI	Emotions	Accuracy: 41.8% (Transformer) and 23.8% (LSTM)
Makris et al.[26]	LSTM and Transformer	MIDI	Emotions	Better pattern metrics (compression ratio, long/short patterns) in transformer compared to LSTM and MuseGAN

TABLE 2.2: Summary on Music Generation Models

Chapter 3

Project Implementation

Our project aims to generate music by conditioning emotion labels in the input during training and inference, similar to the idea proposed by hung et al. [7] using the Transformer-GAN architecture proposed by Muhamed et al. [10]. The goal of this project is to assess the musical quality of the music generated, and whether it is impacted by emotion conditions. Hence, our model consists of two parts:

1. **Generator**: A condition music generation model, and
2. **Discriminator**: A classifier that can critic fake music from real music

An overview of the model is described in fig. 3.1, which shows the most important steps of building the model.

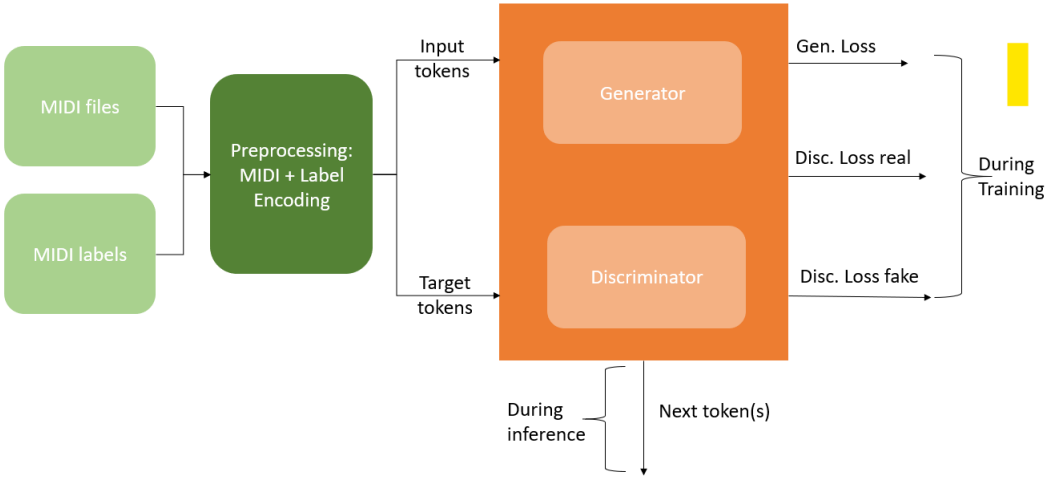


FIGURE 3.1: Overview of the model implementation

The project was implemented using the iterative Agile methodology as discussed in Appendix E Methodology. The project also completed the "Must Have" requirements as discussed in Appendix F Requirements Analysis within the computation scope. Given the expensive computation required for ensemble models such as GANs, the project depended on good resources. However, the project suffered a set back due to the key server lockdown between March 15 - April 15, which shifted our focus to creating sustainable models that can perform within the given resources available. In this chapter we will reiterate our initial implementation methodology and then look into the practical implementation, with justifications and challenges faced. The one drive link to the repository can be found here and Github repository can be found here

3.1 Initial Implementation Methodology

The implementation will follow the steps of a general machine learning pipeline.

1. **Data Collection**: According to the literature review, most authors either compile their own data or use existing datasets with annotated emotions. We chose the latter with the EMOPIA dataset as our primary dataset [7]. We chose EMOPIA since it stores symbolic or MIDI file which can be edited easily and is the preferred format in music production. EMOPIA also reflects current top genres, by using pop music from around the world. Moreover, the data is annotated with balanced classes of Russell's 4Q [31]. The MIDI file and labels will be taken from the dataset.

   We will evaluate the datasets using objective metrics like unique pitches used and more defined in Evaluation Strategy.

2. **Data Pre-processing and Music Encoding**: The MIDI files will be extracted, using libraries such as MidiTok [25] in Python to tokenize events in MIDI files and convert to different music representations. The tokens can then be segmented per sequence if necessary

   We plan to test between different representations, to see which improves the musicality and maintains long-term coherence, as well as test how well the representations identify valence and arousal using accuracy, precision and recall

3. **Building the model**: We hypothesize that GAN-based approaches could create more musicality by using discriminators, like those proposed in [27] and [10]. In our project, the discriminator can be an MER model, while the generator can be a music generation model. In our literature review, LSTM-based models can quickly learn to differentiate between different emotion categories, which is ideal

for our discriminator [7] [50]. For generators, transformer architectures perform well with both unconditioned and emotion-conditioned music generation [1] [7], specifically outperforming LSTM with our chosen EMOPIA dataset. Although these models fail to model valence accurately, Makris et al. proposed using chords to improve valence in transformers [26]. A model integrating Transformer and GAN is the Transformer-GAN which uses a spanBERT classifier as the discriminator [10]. Although we could not find existing work on MER with transformer, we can experiment with LSTM and transformer discriminators to create a model that (1) generates music conditioned on emotions, (2) validates the generated music with a MER model (3) generates long term coherence in the rhythm, and (4) effectively models emotion categories

Our development is a trial-and-error process of combining different architectures and music representation for the model. This can be compared to existing work, and evaluated with by comparing the accuracy of the model and the subjective tests. Objective metrics like information on beats and notes can also be tested

3.2 Software and Hardware requirements

The project was implemented using Pytorch, a Python library for building deep learning models [61]. We chose this library as it offers customisation and lower-level build compared to other libraries. From our initial testing, we found that using the MidiTok library allowed us to efficiently load MIDI datasets [25]. Our computation requirements largely came from building the model, given that we are building a GAN with a transformer as generator and discriminator. There was little to no information available on the GPU resources required for training the EMOPIA Transformer and the Transformer-GAN from the literature review. Hence, we looked into the architectures these models were inspired from, namely CP Word Transformer [2]. In this paper, Hsiao et al. reported on the GPU usage by different types of models based on the data representation and model type. Their results are displayed in fig. 3.2. We could infer that we would need an 11GB GPU RAM to run this model. The closest RAM available to us during the project development was the 8GB RAM Nvidia GeForce RTX 2080 GPU from the lab computers and the 12GB RAM offered by Google Colab. In practice, we could only implement the model on the lab computers, hence we follow the same specifications. More details on our challenges and opportunities with computation limits is described in section 5.3.

Task	Representation + model@loss	Training time	GPU memory
Conditional	Training data	—	—
	Training data (randomized)	—	—
	REMI + XL@0.44	3 days	4 GB
	REMI + XL@0.27	7 days	4 GB
	REMI + linear@0.50	3 days	17 GB
	CP + linear@0.27	0.6 days	10 GB
Unconditional	REMI + XL@0.50	3 days	4 GB
	CP + linear@0.25	1.3 days	9.5 GB

FIGURE 3.2: The GPU usage as reported by Hsiao et al. [2]

3.3 Data Collection and Representation

As per the dataset discussion in sec. 3.1, our primary dataset is the EMOPIA dataset. The dataset is reasonably balanced, as seen in fig. 3.3, in our experiments reducing the dataset to a balanced dataset did not affect the learning significantly, and could worsen the performance given the limited size of the dataset. We also used MusPy, a library for processing MIDI files in Python [62], to calculate the the tempo range and pitch range of the dataset. Our findings report that the MIDI files range in pitch from 22 Hz to 105 Hz, with a tempo of 120 BPM (Beats Per Minute) across all samples.

FIGURE 3.3: Class balance for EMOPIA [7]

Another dataset we used is the VGMIDI dataset [8], which contains 200 clips annotated with arousal and valence. This data set is 5 times smaller than the EMOPIA dataset, moreover, it is not as balanced as seen in fig. 3.4. We used this data to explore the performance of limited data on the Transformer-GAN architecture.

From our initial discussions, we wanted to experiment with sequence music encoding such as Midi-Like and REMI representations as described in sec. 2.2. During the project implementation, we also looked into an additional representation, called as the

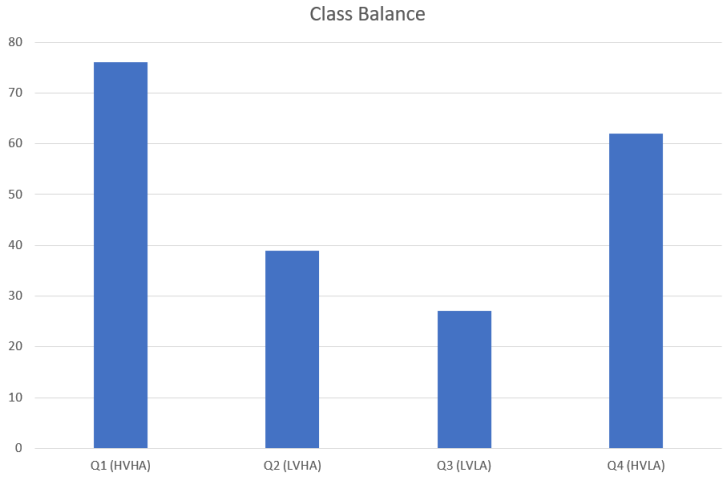


FIGURE 3.4: Class balance for VGMIDI data [8]

Compound Word or CP Word representation. The representation is also described in detail in sec. 2.2. Literature suggests that CP Word performs well in generation tasks, which was also supported by our results later in section 4.4 [2] [7].

The representations generally encode information regarding the musical notes, metrics, and rhythm. In our critical analysis, we noted that chords can improve valence in the generator [26]. Hence, we also used additional tokens to explicitly encode harmonic information such as chords and tempo using the MidiTok library [25]. We also limited the pitch range and tempo in the encoding vocabulary to the pitch range of the original dataset and a tempo range of 100-140 BPM. We reduced our vocabulary so we could limit the embedding sizes in the model, which saves memory required. The vocabulary sizes of each representation shown in tab. 3.1.

Representation	**Vocabulary Size**
Midi-Like	317 words
REMI	245 words
CP Word	276 words

TABLE 3.1: Vocabulary size for each representation

3.4 MER Model

In this section we will go over the steps of implementation for a music emotion recognition model. This model will help us classify models into given emotions, and guides us during the implementation of the GAN.

3.4.1 Data Representation

To train the MER model, the EMOPIA dataset was first encoded using different representations, MIDI-Like and REMI. The encoding uses additional harmony tokens described in sec. 3.3, we hypothesize this may help improve the confusion seen in predicting valence [7] [16]. The steps of building the corpus are described below:

1. The MIDI files are tokenised into notes of arbitrary lengths
2. The tokenised notes are then divided into fixed-size sequences. The original MIDI-Bert trained on sequences of length 512 trained for over 6 days using four GeForce GTX 1080-Ti GPUs [16]. Given our computation limits, we could only train on sequences of length 50 tokens using one GPU. The EMOPIA dataset is labelled with clips sections of 40s of music, this was to maintain the clip-wise emotional impact of the music, compared to an entire piece which can have range of emotions. On average, each clip is 957 tokens with REMI encoding and 983 with MIDI-Like encoding, by dividing each sequence into fixed sizes of 50 tokens, we essentially split each clip into 19-20 sequences each. Through this model, we can observe how strongly can discriminating information be learnt using much shorter sequences, which will be discussed in section 4.3
3. The target is prepared as a label between 0 - 3 where each label i represents the (i+1)th Quadrant in Russell's 4Q.
4. The data and target are split with a 70-30 split on training-validation. Given the slight imbalance, the splits are stratified on the labels. Hence, there are roughly the same percentage of representation in both
5. The split data is loaded into the training and validation data loader

3.4.2 Model Building and Training

We implemented a BERT classifier inspired from the Midi-BERT model [16]. BERT (Bidirectional Encoder Representations from Transformers) is a transformer architecture adopted from the original encoder-decoder Transformer [63]. BERT was developed for effective language modeling, hence, it only uses an encoder to learn how to model different languages, which can be down-streamed to specific tasks such as sequence classification, next token prediction, question-answering, etc [63]. Since it uses the transformer encoder, it encodes the input and learns the context, which is particularly helpful in context-classification tasks such as emotion classification in our case. The architecture of the model is defined below in fig.3.5.

```
# creates an embedding of the tokens
self.embedding = nn.Embedding(self.n_token, self.emb_size)

# passing through a linear layer, to reduce the features to the dimension of the model
self.in_linear = nn.Linear(self.emb_size, self.d_model)

# pass through a BERT model
self.bert = BertModel(config)

# output to 4 classes
self.out_linear = nn.Linear(self.d_model, 4)
```

FIGURE 3.5: The MER model architecture

The input tokens first pass through an embedding layer. We used a custom embedding layer as Bert Embeddings are trained on natural language words. The vocabulary size of the embeddings depend on the music representation. The embeddings further pass through a linear layer, in order to reduce to a fixed model size. Then, the linear output is passed through a BERT model by the HuggingFace library [64]. This library allows us to build complex models easily. The original MIDI-Bert model was build with default HuggingFace values, as described in the tab.3.2. The paper first pretrained for over 6 days with 500 epochs of batch size 12 to learn embedding weights. Then the model was finetuned in 10 epochs [16]. We trained our model for 200 epochs with batch size of 32. The final model's architecture is provided in Appendix G Architecture

Parameters	**MIDI-BERT**	**Our MER Model**
Vocab Size	30522	Vocab Size of Representation
Hidden layers	12	6
Size of hidden layers	768	240
Attention heads	12	6
Sequence Length	512	50
Learning Rate	0.00002	0.0001
Num of Parameters	111M+	7M+

TABLE 3.2: Comparison of parameters in the reviewed MIDI-Bert and our model [16]

3.5 Music Generation Transformer

The next step to conditioned-transformer GAN was building a music generation model. This helps us guide the implementation process, as many of the concepts discovered here are used further in our Transformer-GAN, moreover, this serves us as a baseline for a fair comparison given the difference in scope of our models with the current literature. The implementation is discussed as follows:

3.5.1 Data Representation

Similar to MER models, the data has to be first encoded into different representations, however, unlike MER models, the target is actually the sequence of next tokens to predict. Given length x of a sequence, the input is the sequence of size *(0, x-1)* (x-*1* inclusive), and the target is the sequence of size *(1, x)* (x inclusive). This idea can be demonstrated through a visual example in fig.3.6.

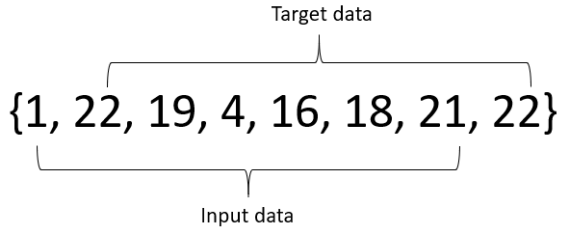


FIGURE 3.6: Splitting a sequence into input and target data

The length of the sequences were determined by the computation resources available, with the initial attempt of 512 tokens in a sequence to mimic BERT transformers [16]. However, in practice, we were only able to train sequences of length 20 for each input and target. In hindsight, this could effect the performance of the model, as it learns from very small sequences. The trade-off between the model size and sequence length is discussed further in the next sub-section.

In addition to the token notes, we also need to encode the emotions for a conditioned input. This can be done in two ways, either by (1) one-hot encoding or (2) by concatenating each note token with the emotion. The former can be done by simply creating one hot encodings of the target, while the latter can be done by stacking the emotion input to every note. For eg. if the note sequence is [1,2,3,..], and the sequence is labeled with emotion 0 (Quadrant 1, HVHA), then the conditioned input is [[1,0], [2,0], [3,0] ....]. In practice, we found that the latter was more powerful in generating emotion embeddings, as it is the same size as the tokens, more over, when embeddings are concatenated, each embedding is of the size (*batch size* * *sequence length* * *embedding size*). With one-hot encoding of four emotion labels, the embeddings of the notes and labels do not match in the first dimension (*sequence length*), as the notes are of length 20 while the emotions have length 4. This can be solved by padding the one-hot encoding to the sequence length of the notes, however, this can result in weaker embeddings. This can be explained as follows, in case (1) the loss function would compare [1,0,0,0] to [0,0,0,0], which would return a smaller loss compared case (2) which compares [1,1,1,1] to [0,0,0,0]. This was also proven during implementation where we saw smaller losses yet the music

had long, incomprehensible empty beats. Hence, the embeddings are weaker, and we continued with the second approach

3.5.2 Model Building and Training

The initial plan was to implement a Transformer encoder-decoder model for music generation, which we attempted to implement using Pytorch's TransformerEncoder and TransformerDecoder modules. Given the computation resources, the model could not converge using only one layer and one attention head. Hence, we looked into alternate architectures such as BERT and GPT-2 which only user encoder or decoder part of the architecture. We noted how BERT only uses encoders in the transformer to model context in a language in section 3.4.2. GPT-2 is another state of the art transformer architecture, which only uses the transformer decoder for auto-regressive generative tasks [65]. Auto-regressive prediction can be described as predictions dependent on previous input values, for eg. predicting the next token based on all previous tokens [66]. This is very similar to the music generation task, where each run predicts the succeeding token based on the current played notes. The Transformer-XL model in the literature review is another auto-regressive model [54]. These models only require a decoder, which also reduces the model size by roughly half. Due to computation limits, we developed models inspired by the GPT-2 and Transformer-XL [65] [54].

Embeddings

The first step is to create embeddings of the input tokens. The notes and emotions are embedded separately, each of size 128. The embeddings are then concatenated and fed into an absolute positional encoder. This encodes the positional information into each token. Once the embeddings are calculated, the output is concatenated and passed through a linear layer. This allows the input to be a concatenated representation of size (*batch size* * *sequence length* * *model dimension*). The formula and code for positional encoding are provided in fig. 3.7 and fig. 3.8 respectively.

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/dmodel})$$
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/dmodel})$$

FIGURE 3.7: Position encoding formula [5]

```python
class PositionalEncoding(nn.Module):

    def __init__(self, d_model, dropout=0.1, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)

        # PE is the Positional Encoding matrix
        # THIS STORES THE POSITIONS OF THE SEQUENCE
        pe = torch.zeros(max_len, d_model)

        # Arange - RETURNS A RANGE BETWEEN VALUES, HERE IT IS 0 - max_len
        # unsqueeze - adds a dimension, 1 means that each element in the first list is now in a list
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        # division term, here it is (10000 ** ((2 * i)/d_model))
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-math.log(10000.0) / d_model))

        # calculating the position encoding for the even and odd terms
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        # Unsqueeze 0 will put PE in one list
        pe = pe.unsqueeze(0)
        self.register_buffer('pe', pe)

    def forward(self, x):
        # make embeddings relatively larger
        # This is so we do not lose the importance of the embedding
        # we add the embedding to the PE
        x = x + self.pe[:, :x.size(1), :]
        return self.dropout(x)
```

FIGURE 3.8: Position encoding code [7]

Transformer Encoder Layer

Although we dubbed GPT-2 and Transformer-XL, the guide for our model as 'decoder-only', they are not true encoders defined by Vaswani et al [65][66]. Encoders learn the entire context of the sentence, while decoders only rely on previous tokens in the sequence. However, decoders need a 'memory' input, which is the memory representation created by an encoder. Transformer-XL solves this by using recurrence to create memory, while GPT-2 solves this by using an encoder with masked attention, an upper triangular mask [65]. We follow the latter approach, as adding recurrence to the model is much more computationally expensive and cannot effectively produce reasonable results within our scope.

Pytorch has its own Transformer Encoder layer, which computes self attention and feed forward, with a source mask. We can provide an upper triangle mask, which masks all succeeding tokens for each token in the sequence. We could implement this model with a model dimension 512, and 8 layers of 4 multi-attention heads, each of hidden size 512. We also added a dropout of 0.2 to encourage generalising. The input is sent as (*sequence length* * *batch size* * *embedding size*). Hence the output from the positional encoding has to be reshaped to be sequence-length first.

Output

The final output is project the inputs as notes and emotions, this is done simply by projecting the output from the encoder with two linear layers, each of vocabulary size of the encoder tokens and emotions respectively. This approach is also adopted by EMOPIA Transformer [7], as it allows parallel computation of notes and tokens in one run. Hence, the output of the model is the predicted weights of the next token.

Training

An Adam optimizer was used while training, with a scheduler to decay the learning rate at each step by 0.05. CrossEntropyLoss by Pytorch was used to calculate the loss, which performs a log-softmax and then NLL loss for multi-class classifications [67]. The model has 25M+ parameters alone, which could be only trained for 10 epochs. After trial and error, this was the maximum parameter size that could run effectively on sequences of length 50 at a time. REMI also had 63459 less parameters as it uses fewer tokens compared to MIDI-Like. As seen in our literature review, REMI is more efficient in implementation compared to MIDI-Like representations, and hence will continue as our preferred representation.

The music produces by these transformers suffered long rests and difficulty in developing patterns, the music subjectively lacked coherence, which could be explained by the shorter sequence length. However, with growing model complexity, the sequence length suffers more, as seen in section 3.7 Our model was approximately half the size of the EMOPIA transformer, as seen in tab.3.3, and was only learning sequences of length 50 at a time compared to 1024 with their transformer [7]. The final model's architecture is provided in Appendix G Architecture

Parameters	**EMOPIA Transformer**	**Our Transformer**
Hidden layers	12	8
Size of hidden layers	512	512
Attention heads	8	4
Sequence Length	1024	50
Learning Rate	0.00001	0.0001
Num of Parameters	39M	20M+

TABLE 3.3: Comparison of parameters in the reviewed EMOPIA Transformer and our model [7]

Generation

Generation with REMI/MIDI-Like transformers is straightforward. Given the number of generations required, say, *ngen*, the model is run *ngen* times, starting with a vector of size 1 and appending the new tokens to the input after every run. The next tokens are predicted by sampling from the multi-nomial probabilities output by the transformer. This is implemented as shown in fig. 3.9.

```
n_generate = 4000
temperature = 1
sequence = []
log_interval = 4000 # interval between logs
input = torch.randint(len(corpus), (1, 2), dtype=torch.long).to(device)
emotion = torch.full((1, 2), emo).to(device)
# emotion[:,0] = emo

# generate new mask
src_mask = generate_square_subsequent_mask(len(input)).to(device)
with torch.no_grad():  # no tracking history
    for i in range(n_generate):
        output, _ = model(input, emotion)
        output = F.log_softmax(output, dim=-1)

        # get the weights of the last (next word) token
        # divide by temperature for diversity
        word_weights = output[-1].squeeze().div(temperature).exp().cpu()
        # return one sample from the distribution
        word = torch.multinomial(word_weights, 1)[0].tolist()
        word_tensor = torch.Tensor([[word]]).long().to(device)

        # concatenate new word and same emotion to the token
        input = torch.cat([input, word_tensor], 1)
        emotion = torch.cat([emotion, torch.Tensor([[emo]]).to(device)], -1)
```

FIGURE 3.9: Generating music with transformers

3.6 Loss functions

Negative Log Likelihood, or NLL loss is a common classification loss function that compares the distribution of the predicted weights to the actual target values [68]. In our literature review, NLL loss is a common evaluation metric for evaluating generative models, as it can be used to compared the predicted tokens to the target tokens. In multi-class classifications, it is referred to as 'cross entropy loss', which uses a Softmax activation + NLL Loss. Pytorch implements this as the CrossEntropyLoss function,

Unlike other generative models, GANs are trained using Adversarial losses, which are loss functions that play against each other, in-tune with the generator and discriminator playing against one another. The most common loss function for GANs is a log loss function for predicting the real or fake samples [41]. With conditioned inputs, the loss functions have to be adapted to accept conditioned inputs. We used the approach proposed by Mirza et al., which adds a conditioning to the vanilla gan loss [69]. As seen

in eq. 3.1, the discriminator aims to minimise the loss of predicting x sampled from p_{data} (real batches) as real and predicting z sampled from p_z (noise function) as fake. Both the samples are conditioned when input.

$$L_D = E_{x\sim p_{data}(x)}[logD(x|y)] + E_{z\sim p_z(z)}[log(1 - D(G(z|y)))] \tag{3.1}$$

The generator tries to minimise the loss of predicting the fake sample as real, as shown in eq. 3.2

$$L_G = -E_{z\sim p_z(z)}[log(1 - D(G(z|y)))] \tag{3.2}$$

This can be implemented using BCElossWithLogitsLoss in Pytorch, which returns the binary cross entropy (log softmax + negative likelihood) of the target and prediction.

3.7 Transformer-GAN

As we can see, the transformers alone do not capture any reasonably comparable results, moreover, the music is suffering from long pauses in its implementation. We recovered these by implementing our Transformer-GAN with the following changes. The data representation of the model was similar to our generative Transformer.

3.7.1 Generator

The generator is the conditioned-music generation model. We adapted our conditioned-music generation Transformer as our generator. Our transformer architecture was similar to EMOPIA transformer [7]. However, we developed the model without the recurrence mechanism and memory used in the EMOPIA Transformer to implement within the computation scopre.

3.7.2 Discriminator

The discriminator is the critic that decides how realistic the generated output is [41]. At first, it was intuitive to implement the MER model as our discriminator, so the model could learn to produce emotional music. However, that is not the case in practice. What we failed to account for was that although the discriminator and generator are trained every epoch, they are trained individually, and the loss from the discriminator does not get propagated to the generator. This leads to erratic models which only tries to fool the discriminator by using selective pitches, chords and other tokens found in each quadrant,

but without any musicality and often with very long rest periods. Hence, we revisited the implementation, by first developing a simple binary class discriminator that decides if the music is real or fake, and then building upon this architecture. We used the MER model for classifying the data instead as another evaluation metric

The principle of our discriminator is similar to the architecture of official GAN models, which encodes the input and predicts between real or fake samples. However, it is important that the architecture is as comparable in size to the generator, to avoid an imbalance in the training dynamics, since GANs (and effectively Transformer-GANs) can be notably difficult and unstable to train [41] [70]. LSTM discriminators struggled to catch up to the transformer generator, even if trained multiple times in each run. Hence, in this project, we chose to implement a transformer-classifier, which predicts a binary value of real or fake input. To learn the complete context of the input, we do not apply triangular masking. We also input the conditioned emotions to the discriminator, and create a combined encoded representation of the music tokens and the emotions that can be classified as real or fake. Although the model is still smaller in capacity compared to the Generator, it is effective enough to develop overconfidence as described in section 3.7.3.

3.7.3 GAN

Now that we have our individual generator and discriminator, we combine them in our GAN architecture. The model is simple in theory, following the approaches used in vanilla GANs and conditioned-GANS (CGANs) [41][69]. Both the generator and discriminator take conditioned inputs, i.e. the tokens with accompanying emotions, as described in section 3.5.1. The real data is prepared by sequentially sampling from the train data loader, in batches of eight at a time. Each input sequence is of shape *(x, k+1)*, where k dimensions represent the music and one dimension represents the emotion. With REMI encoding, there are 1 dimension (sequence) of MIDI events. The target sequence is also of length x, with the last token as the predicted token. Fake samples can be generated with random sampling from the vocabulary.

The following steps are following with each batch. The first step is training the discriminator as shown in fig. 3.10. To learn how to predict real data, the discriminator first inputs a real sample with it's emotion, and predicts the probability of the sample being real or fake. Then, the generator (frozen to avoid propagating discriminator gradients to the generator), is given a fake sample with the same emotions, and the discriminator predicts the probability of the generated sample as real or fake. Both the loss (fake) and loss (real), can be combined as shown in eq. 3.1.

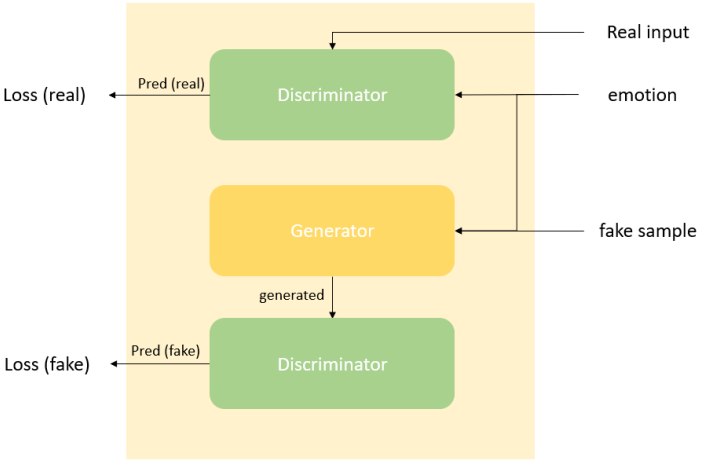


FIGURE 3.10: Training the discriminator

For the generator, we sample a fake noise input again, but with the same emotions as the discriminator was trained on for every input. Here, the model generates a sample, which is classified by the discriminator as real/fake. Hence, the generator minimises the loss in eq. 3.2. This is visualised in fig.3.11. For a fair comparison with existing models, we also compute the NLL loss, which compares the generated tokens with the target token. The model is trained with Adam optimiser, as it works well with GANs [71].

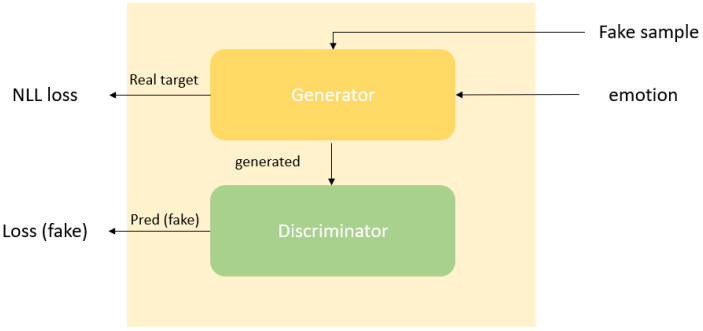


FIGURE 3.11: Training the generator

The following hyperparameters were used for the REMI Transformer-GAN, as shown in tab. 3.4. In total, the model had over 11M parameters, with the same hyperparameters set up for the generator and discriminator. Although there was no exact number of parameters given for Muhamed et al.'s Transformer-GAN, an estimate of BERT and the Transformer-XL's number of parameters were calculated to be over 123M parameters. Our model is effectively at least ten times smaller than the original model, with sequences six times shorter and without recurrent memory.

Parameters	**Transformer-GAN**	**Our conditioned Transformer-GAN**
Hidden layers	6+12	4+4
Size of hidden layers	500+768	512+512
Attention heads	10+12	6+6
Sequence Length	128 + 2048(memory)	20
Learning Rate	0.004	0.001
Num of Parameters	123M+	11M+

TABLE 3.4: Comparison of parameters in the reviewed Transformer-GAN and our model [10]

The final model's architecture is provided in Appendix G Architecture. During implementation, we addressed several tricks and tips for modeling with GANs as follows.

Generation length

Our generator is an auto-regressive model, which means it generates tokens one at a time. Traditionally generators generate entire images or sequences in one run [41]. Hence, traditional GAN architectures only require one run of the generator to be fed into the discriminator. When we followed this approach, we saw that the model was not able to converge, with the discriminator overpowering the generator. This can be explained as the model is effectively classifying noisy data with the one predicted token. That is, given a noisy data of length x, the model passes the input through the generator and returns the target sequence with the next $x+1$th token. If we simply input this to the discriminator, the discriminator is classifying x-1 noisy tokens and one generated token, hence the generator and discriminator are imbalanced, as they are not learning equal amount of data, and the model would not converge. Hence, we generated x tokens for every run of the generator, which could be fed to the discriminator as one complete fake sample. This helped the model to begin to converge.

WGAN / WGAN-GP

The convergence, however, was within five epochs, suggesting that the model was stuck in a local minima, as the discriminator and generator did not improve after a few epochs. With random weight initialisation, the model often suffered exploding or vanishing gradients, even after clipping the range from -0.1 to 0.1. Wasserstein GANs (WGANs) are an adaptation of vanilla GANs to combat vanishing gradients [9]. The loss function with conditioned input can be described as eq. 3.3. It is important to note that WGAN losses can become negative

$$L_D = Sigmoid(D(G(z|y))) - Sigmoid(D(x|y)) \quad L_G = -Sigmoid(D(G(z|y))) \quad (3.3)$$

To implement the WGAN loss, the discriminator should return one value, over which the sigmoid activation function is used. Sigmoid activation function helps clip the range of the losses, which can be further smoothened using a gradient penalty.

Gradient penalty is an additional regularisation function that penalises the discriminator to avoid vanishing and exploding gradients [9]. In gradient penalty, a sample is interpolated between the real sequence and generated sequence. Since the interpolated value is a floating tensor (gradients can only be calculated as floating points), we manually calculate the embedding vector with the current weights. The discriminator then skips embedding, and returns a classification on the input. Gradients are calculated on the input interpolated sequence and output classification, and it is returned after regularising with a lambda value (0.02 in our case). The formula is given in eq. 3.12 and the implementation in fig.3.13. Muhamed et al. also found best results using the WGAN-GP loss in their Transformer-GAN [10].

$$\mathcal{L} = \mathop{\mathbb{E}}_{\tilde{\mathbf{x}} \sim \mathbb{P}_g} [f(\tilde{\mathbf{x}})] - \mathop{\mathbb{E}}_{\mathbf{x} \sim \mathbb{P}_r} [f(\mathbf{x})] + \lambda \mathop{\mathbb{E}}_{\hat{\mathbf{x}} \sim \mathbb{P}_{\hat{x}}} [(||\nabla_{\hat{\mathbf{x}}} f(\hat{\mathbf{x}})||_2 - 1)^2]$$

FIGURE 3.12: The discriminator loss with gradient penalty [9]

```python
# adapted from
# https://github.com/amazon-research/transformer-gan
def calc_gradient_penalty(self, real, fake, LAMBDA=0.02):
    temp_notes = torch.rand([real.shape[0], 1]).to(device)
    # expand into the shape
    temp_notes = temp_notes.expand(real[:,:,0].size())
    # interpolation
    mid = temp_notes * real[:,:,0] + ((1 - temp_notes) * fake[:,:,0])
    mid = mid.type(torch.LongTensor)
    mid = mid.type(torch.FloatTensor).to(device)
    mid = torch.autograd.Variable(mid, requires_grad=True)
    # print(mid.shape)
    mid = torch.einsum(
        "ve,bn -> bne",
        self.discriminator.embedding_notes.weight,
        mid,
    )
    classification = self.discriminator(emb_note = mid, x_emo = real[:,:,1].to(device))
    gradients = torch.autograd.grad(outputs=classification, inputs=mid,
                                    grad_outputs=torch.ones(classification.size(), device=device),
                                    create_graph=True, retain_graph=True, allow_unused = True)[0]
    gradients = gradients.view(real.shape[0], -1)
    # https://github.com/igul222/improved_wgan_training/blob/master/gan_language.py
    slopes = torch.sqrt(torch.sum(gradients ** 2, dim=1) + 1e-12)
    gradient_penalty = ((slopes - 1.) ** 2).mean() * LAMBDA
    return gradient_penalty
```

FIGURE 3.13: Implementing the gradient penalty in Python [10][9]

This caused the range of losses to fall between -1 to 1, and showed some improvement. WGAN loss with gradient penalty helped converge the GAN, however the NLL loss of the model only decreased slightly after 20 epochs, which suggests very the model was still not effectively able to predict new tokens. We hypothesized that this may be since the sequence length of training was much shorter than our literature review, as our models are only training on 20 tokens at a time, compared to 512-2048 tokens trained for the EMOPIA transformer and the Transformer-GAN [7][10]. Unfortunately, REMI suffers from exploding the corpus size, as the 900+ tokens to represent one clip. Hence,

we moved on to other representations to help reduce the average number of tokens per clip.

CP Word Encoding

CP word or compound word is another representation for musical events, as described in section 2.2.1 [2]. We tokenised the music clips using the MidiTok library. CP Word reduced the average sequence length of the tokens per clip to 428 tokens. This nearly halves the number of tokens compared to REMI, however this creates eight dimensions (Family, Bar, Pitch, Velocity, Duration, Chord, Rest, Tempo) to represent a single musical event. This expands the model size, as each type of token is embedded individually, creating $8 + 1$ (emotions) embeddings. In order to reduce the parameters of the model so it could be trained, the internal size (for the model dimensions between the embedding and encoder layer), was reduced from slightly.

To adapt the model to train with CP word tokens, all elements of processing notes in the model were expanded by eight + one dimensions. Hence, there were nine embeddings in the discriminator and generator, nine projections and outputs, the gradient penalty is calculated for each token type, and the NLL loss was calculated for the prediction of each token type (nine in total).

When generating with CP word, predicting the next token by returning the token with maximum multinomial probability was not very effective, moreover, the generated samples do not always capture the relationships between the tokens. For example, if the Family token is set to “note”, MidiTok only accepts tokens that do not have “ignore” or “none” tokens for Pitch, Velocity and Duration [25]. Similarly, ”metric” family tokens require a non-null value for tempo [25].

Hence, we added temperatures to the weighted sampling, and manually mitgated the “ignore”/”none” tokens where needed. One approach can be to continue with the last value of the token type. For eg. if the current Pitch is ”none”, we set it to the last value it was set at. However, this defeats the purpose of independent generation, as we would be directly manipulating the generated values, which would not reflect what the model has learnt. Instead we opted for selecting only the ”good” tokens, which are tokens that satisfy the dependency of the token types with each other. Hence, the predicted output would contain only these tokens. This issue was not discussed explicitly in the EMOPIA Transformer model which also uses CP words, however their implementation suggests a similar approach [7]. Adding the temperatures also encouraged diversity. This is especially effective for the rest tokens, which is an explicit token type for MidiTok

library but not with the custom CP word representation for the EMOPIA Transformer [25] [7].

Propagating NLL losses

Early on in the training, we noted that converged models perform poorly in retaining musicality and emotion. This meant that although the model was converging, it was suffering mode collapse. Mode collapse is a common problem with GAN, where the model predicts the tokens similar to the same class or mode [41]. This was also observed when the NLL loss did not change after every epoch. Although literature suggests that conditioned inputs can improve mode collapse [41], our results suggested otherwise. Shahbazi et al. also found that mode collapse still occurs with limited data [72]. Hence, we proposed propagating NLL loss with the generator to improve the NLL loss, and effectively improve feature matching of the predicted outputs to the final targets. Our results were fruitful, as we were able to reduce the REMI Transformer-GAN NLL loss.

Overconfidence in discriminator

During training, we noted that after a few epochs, the discriminator would grow overconfident, and the generator would not be able to catch up to the discriminators performance. This can be explained as the discriminator has to learn much fewer features with the combined representation of all the embeddings, into a single binary classification. This hurts the performance of the generator, as the discriminator classified fake samples with a very high probability (over 90%). We tried to reduce the overconfidence using the following ways

Intuitively, we expected the generator to pick up if we trained it twice or thrice as much as the discriminator at every run, however this was not the case as the generator loss still depends on the discriminator weights, which are already confident from a single training in every run. We could not pretrain the generator as that defeats the purpose of a GAN, which aims to train the generator using the discriminator. Hence, we attempted to penalise the discriminator by using reducing the model size, increasing learning rate, and increasing drop out. This proved to have little effect on the model, as the discriminator still grew overconfident sooner or later. Eventually we attempted two regularisation techniques.

The first was to introduce one-sided label smoothing, which is a regularisation technique for GANs to penalise labels predicting real images with high confidence [11]. This can be done by reducing the label 1 for real music to a smoothed range between 0.8 - 1.2.

The implementation can be viewed in fig.3.14. Since WGANs work with the fixed labels 0 and 1 (fake and real) respectively, we used a binary cross entropy loss, that allows the model to learn from smoothed targets.

```
# Cross Entropy loss with label smoothing
# https://arxiv.org/pdf/1606.03498.pdf
# https://github.com/NVIDIA/DeepLearningExamples
def label_smoothing_loss(self, x, is_real, smoothing = 0.5):

    if is_real:
        # real labels are smoothened from 1 to a range between (0.8, 1.2)
        # One Sided Label Smoothing (Real Label [0.8,1.2]
        target = torch.tensor(random.randrange(8, 12) / 10)
    else:
        target = torch.tensor(0.0)

    target =  target.expand_as(x).to(device)

    return self.criterion(x, target)
```

FIGURE 3.14: One sided label smoothing [11]

The second step was to introduce some noise for the discriminator, by training it with the wrong labels after every few batches. Label flipping is a common technique to penalise confidence of predicting real values with the discriminator [73]. This helps create more confusion in the discriminator, which allows the generator to catch up with stronger performance. We implemented label flipping after every four batches.

Chapter 4

Results and Evaluation

In this chapter we will dwell deeper into the results and evaluation from the final models, and compare it to existing implementations, as well as our models on different datasets. Finally, we will conclude how the music performs in the subjective human listening test, and conclude with the justification of current results and future works.

4.1 Evaluation Strategy

In this section we define the strategy used for evaluating the performance and requirements of our model. Since our project uses both a classification and a generation model, we apply different metrics for classification and music generation. The key strategies covered here are Confusion Matrix and its metrics, BLEU score, other music metrics and Human Subject Evaluation

4.1.1 Confusion Matrix and its Metrics

Confusion matrices are popular performance metrics for classification models as they provide a comparison for the predicted labels with the ground truth. For binary classifications, a 2 x 2 matrix represents four essential information. Clockwise from the top-left, they are:

- True Positives (TP): Number of records that were predicted positive and are positive
- False Positives (FP): Number of records that were predicted positive but are not positive

- True Negatives (TN): Number of records that were predicted negative and are negative
- False Negatives (FN): Number of records that were predicted negative but are not negative

This is used in further performance metrics such as:

- **Accuracy** $\frac{TP+TN}{TP+TN+FP+FN}$: The ratio of correct predictions with total predictions. Accuracy by itself does not account for imbalance in data
- **Precision** $\frac{TP}{TP+FP}$: The ratio of correct positive predictions with the total positive predictions.
- **Recall** $\frac{TP}{TP+FN}$: The ratio of correct positive predictions with the total positive labels.
- **F-score** $2 * \frac{Precision * Recall}{Precision + Recall}$: Used to seek balance with precision and recall. Since it covers different aspect of the confusion matrix and is not swayed by the imbalance of data, it is used in place of Accuracy as a better metric.

4.1.2 BLEU Score

Bilingual Evaluation Understudy (BLEU) score is used to compare predicted and original sequences [74]. It compares consecutive phrases in the generated sequence with the phrases in the original sequence, independent of the position of where they begin. The scores range between 0 and 1, and higher BLEU scores imply higher similarity, while lower BLEU scores imply originality.

4.1.3 Other objective metrics

Apart from the BLEU score, we can look toward the notes and the features in the generated and real music samples to calculate metrics. These metrics are also found in some papers surveyed in our literature review, specifically in [27] [7] [10], and are discussed below:

- **Pitch Range (PR)**: The range of pitch used. The more similar it is to the training data, the better it is
- **Unique Pitch Class (NPC)**: Number of unique pitch classes. Also depends on the NPC of the training data

- **Polyphonic (POLY)**: Number of simultaneous notes (like chords), higher POLY indicates complex harmony
- **Empty Beat Rate (EBR)**: Ratio of rest notes played over total notes. This gives us an idea of the silence in the generated music
- **Negative Log-Likelihood (NLL)**: Measures the loss of the predicted note from the probability distribution

The first four metrics can be extracted using Python toolkit MusePy [62]. The negative log-likelihood can be calculated as the loss function during the training.

4.1.4 Human Subject Evaluation

As our end goal is to create music for human beings, it is critical that this model can create music that is actually usable and likeable by humans. Hence, human subject evaluation is absolutely necessary in evaluating the performance of the model.

We plan to study the human evaluation of the model by proposing an online questionnaire to a set of participants. We plan to design a within-subject experiment where participants will be given a set of short musical pieces, and will be asked questions such as their opinion on the music and whether they believe the music is generated by human composers or AI, as well as be asked to categorise the music as an emotion class. The emotion classes follow Russell's 4Q [31], which will be explained to them. Although participants are not expected to have a background in music, they will be required to have a brief understanding of artificial intelligence. Participants will not be asked to download any software, instead the data will be collected using HWU-approved third party apps such as the Microsoft Office package. All data will be anonymised under GDPR requirements.

A consent form will be handed out to the participants to inform them of the purpose, process and the usage of the survey, present in Consent Form. They will also be informed about their right to withdraw from the survey at any point, at which their data will be deleted. They will also be asked their consent on sharing the results in a publication. The study will take place as following steps:

- **Pre-survey**: Gathering demographic information, experience with music, and the knowledge of Russell's 4Q. They will be briefed about the classes of 4Q, and will be given sample of music from each class to familiarise themselves with each quadrant. The form is available in Pre-Survey

- **Questionnaire**: The participants will be asked to listen to a few short clips of music. After listening to each clip, they will be asked a few questions. Between each clip, a break or a neutral sound will be played to not project opinions from the previous clip. The questionnaire is available in Questionnaire

Overall, this evaluation will give us an insight on how well can this model's work be perceived by a human audience, in case it is deployed for public use.

4.2 Results from Literature

In this section, we will look into some results from the reviewed models for music generation and music emotion recognition. This will help us understand the performance of our final model.

4.2.1 Baseline Music Generation: Music Transformer

The Music Transformer is one of the first few attempts at symbolic music generation transformer model, hence it serves as our baseline [12]. The model was trained on two datasets: one on Bach's chorales and the other was a database of piano-arrangements in MIDI. The piano arrangements were represented using MIDI-Like encoding, which can serve as a comparison of the model. The model was trained on sequences of length 2048 tokens with six layers. They also used a global attention mechanism to improve performance. The final results reported the validation NLL as 1.84. A direct comparison cannot be made since the model is trained on different datasets, but the performance of the model gives us an idea with respect to the Muhamed et al.'s Transformer-GAN discussed in section 4.2.3. The results from the Music Transformer are highlighted in fig. 4.1

Model variation	Validation NLL
PERFORMANCE RNN (LSTM) (3L, 1024hs)	1.969
LSTM with attention (3L, 1024hs, 1024att)	1.959
Transformer (TF) baseline (6L, 256hs, 512att, 2048fs, 1024r, 8h)	1.861
TF with local attention (Liu et al., 2018) (8L, 1024fs, 512bs)	1.863
TF with relative global attention (our efficient formulation) (6L, 2048fs, 1024r)	**1.835**
TF with relative local attention (ours) (6L, 1024fs, 2048r, 512bs)	**1.840**

FIGURE 4.1: Results from the Music Transformer [12]

4.2.2 Baseline Conditioned Generation: EMOPIA Transformer

The EMOPIA Transformer is the closest implementation to our research question, hence it serves as our baseline model [7]. The parameters used in the EMOPIA transformers are shown in fig. 3.3. The results of the model indicated how their model shared closer proximity in musical metrics with the original dataset. There was no information provided on the NLL loss on the dataset as well. The results are displayed in fig.4.2.

Model	Objective metrics						Subjective metrics		
	PR	NPC	POLY	4Q	A	V	Humanness	Richness	Overall
EMOPIA (i.e., real data)	51.0	8.48	5.90	—	—	—	—	—	—
LSTM+GA [10]	59.1	9.27	3.39	.238	.500	.498	2.59±1.16	2.74±1.12	2.60±1.07
CP Transformer [35]	53.4	9.20	3.48	**.418**	**.690**	.583	2.61±1.03	2.81±1.03	2.78±1.03
CP Transformer w/ pre-training	**49.6**	**8.54**	**4.40**	.403	.643	**.590**	**3.31±1.18**	**3.22±1.23**	**3.26±1.15**

FIGURE 4.2: Results from the EMOPIA Transformer [7]

4.2.3 Baseline Transformer-GAN: Transformer-GAN

Muhamed et al.'s unconditioned Transformer-GAN follows our architecture closely, albeit with heavier generator and discriminator. The parameters used in their model is shown in tab. 3.4. The results of the model are compared against the original dataset, MAESTRO MIDI V1 [75]. The results are displayed in fig. 4.3. An interesting result is the performance of the Music Transformer compared to the Transformer-GAN. Muhamed et al. reported that the Transformer-GAN could only improve on Music Transformers performance for next token prediction by 0.04.

	Discriminator	NLL ↓	Sampling	CA ↓	PLL ∼	PCU ∼	ISR ∼	PRS ∼	TUP ∼	PR ∼	APS ∼	IOI ∼
Training Set	–	–	–	–	2.0203	7.810	0.586	0.399	65.28	67.34	11.531	0.133
Music Transformer	–	1.79	Random	0.8443	2.5666	7.214	0.599	0.465	54.95	62.035	**11.619**	0.113
Transformer-XL	–	**1.74**	Top32	0.8377	2.1531	7.045	0.572	0.284	52.95	60.39	11.117	0.107
WGAN-GPen	CNN	1.75	Random	0.8401	2.3087	6.95	0.608	**0.327**	52.28	59.37	10.832	**0.119**
WGAN-GPen	Pretrained BERT	1.75	Random	**0.8179**	**2.1020**	7.19	**0.585**	0.277	**55.56**	**63.23**	11.935	0.145
PPO-GPen	Pretrained BERT	1.75	Random	0.8213	2.3549	6.932	0.598	0.298	52.31	59.245	10.808	0.163
RSGAN-GPen	Pretrained BERT	1.75	Random	0.8307	2.2766	**7.285**	0.578	0.304	54.11	62.83	11.461	0.136
RSGAN	Pretrained BERT	1.75	Random	0.8615	2.1084	6.56	0.607	0.192	48.165	55.62	11.256	0.082

FIGURE 4.3: Results from the unconditioned Transformer-GAN [7]

4.3 MER Results

We first examine our MER model, which returned an accuracy of 58% on Midi-Like and 61% on REMI representation. It seems that training with sequences of length 50 is not enough to compute the features in the BERT effectively, as the Midi-BERT Classifier reached 67% accuracy using 512 tokens in a sequence (see tab. 3.2) [16]. From

the confusion matrix in fig. 4.4, the model performs better when valence and arousal are on opposite extremes, i.e. HVLA and LVHA (Q4 and Q2). Explicitly encoding chords in the valence improved the overall accuracy and performance on low valence. The confusion matrix of the MER model is shown in fig. 4.4. Our results follow the results of the EMOPIA classifier and Midi-BERT classifier [7][16], however we saw better performance with high valence (Q1,Q4) in average compared to low valence (Q2, Q3). The model suffered confusion with its neighbouring quadrants, especially among the high arousal (Q1, Q2) group and low arousal (Q3, Q4). We saw a jump in the accuracy from 51% to 61% simply by using 50 tokens in a sequence instead of 20. However, this was our computation limit. Based on our results, we believe the model is still in the process of learning and requires longer training sequences to perform better.

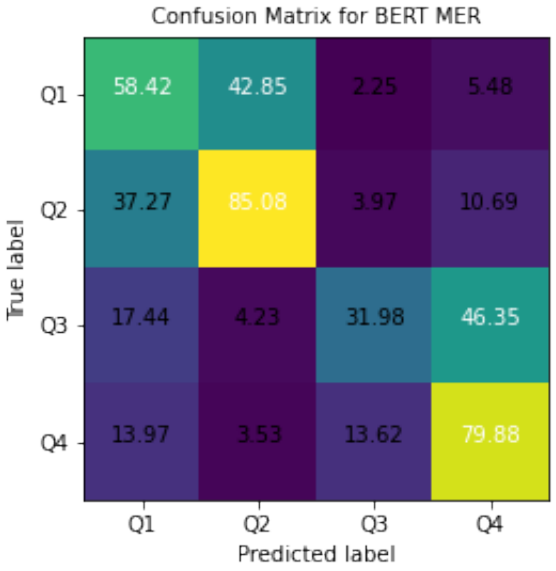


FIGURE 4.4: The confusion matrix for our BERT MER model

4.4 Music Generation Results

This section highlights the final results to understand our performance and compare our results with the baseline models. For ease of reference, our Conditioned-Transformer GAN model will also be referred to as C-TransGAN model. During the calculation of the results we came across a pecularity of computing the BLEU score with all generated samples, as always returning a score of 0. This peculiarity was confirmed when the generated samples from the baseline models returned the same results. At first, it was intuitive to believe that matching 4-grams in the generated samples and true samples can calculate the proximity of the generated samples with the true samples, however that was not the case in practice. We believe this may be from two reasons: (1) The order

of the events do not always stay the same in the tokenisation. For example a sequence of tokens could contain [...,112,112,...] which could denote two rest tokens, each of 1 second, or it could contain [...,113,...] which denotes one rest token of 2 second. (2) The BLEU score loses its meaning with CP Word Encoding. In CP Word Encoding, each token is represented with at most eight token types. Given the family type tokens, some token types could carry meaning and others could be ignored. Hence, there is no explicit dependency on the succeeding tokens, as a "metric" family token could follow a "note" family token or vice-versa without losing its meaning. Hence, we discarded our results on the BLEU score.

For a fairer comparison, we retrained the EMOPIA transformer with our hyperparamters, using the same model dimensions and number of layers. The model architecture with its recurrrence mechanism and the implicit handling of the data and representation was kept intact. The results of all the objective metrics on the original and our models is provided in tab. 4.1.

Dataset	**Representation**	**Model**	**PARAMs**	**NLL**	**PR**	**NPC**	**POLY**	**EBR**	**Accuracy**
EMOPIA					50.9	8.48	5.90	0.005	61%
VIGIMIDI					48.2	10.47	2.94	0.01	33%
PianoMIDI	MIDI-Like	Music Transformer	N/A	1.84	N/A	N/A	N/A	N/A	N/A
EMOPIA	CP Word	Transformer (Original)*	39M+	N/A	**53.4**	**9.20**	3.48	N/A	N/A
EMOPIA	CP Word	Transformer (less hyperparameter)	20M+	**1.2**	46.2	11.5	4.6	0.17	33%
MAESTRO	REMI	Transformer-GAN (Original)*	123M+	1.75	63.23	7.19	N/A	0.975	N/A
EMOPIA	Midi-Like	Our Transformer	20M+	3.5	62.7	4.7	1.0	0.98	4%
EMOPIA	REMI	Our Transformer	20M+	3.39	83.0	11.4	**4.9**	0.09	16.7%
EMOPIA	REMI	C-TransGAN	11M+	5.75	78.0	11.5	1.2	0.49	4%
EMOPIA	CP Word	C-TransGAN	11M+	3.69	82.8	12.0	18.2	0.04	11%
EMOPIA	REMI	C-TransGAN + WGAN GP	11M+	3.5	84.1	11.4	7.01	0.04	4%
EMOPIA	CP Word	C-TransGAN + WGAN GP	11M+	1.5	81.2	11.5	19.0	**0.02**	**33.3%**
EMOPIA	CP Word	C-TransGAN + label smooth	11M+	**1.2**	81.7	11.5	19.8	**0.02**	20.8%
VGMIDI	CP Word	C-TransGAN + label smooth	11M+	**1.11**	81.8	12.0	10.8	0.05	16%

TABLE 4.1: The final results of the base line and our models [7] [10]. NLL = Negative Log Likelihood loss, PR = Pitch Range, NPC = Number of Pitch Classes, POLY = Polyphony, EBR = Empty Beat Rate, Accuracy = Prediction accuracy by our MER model. The starred models are results from the original paper. N/A means that the results were not given in the literature or not calculated. The bolded results are the best performance in that metric

Transformer architectures

Our transformer-only architectures is not comparable alone to the EMOPIA transformer model with the same hyperparameters. The results indicate that the model achieves NLL of 1.2 given the same number of epochs, and an accuracy of 33.3% on the emotion of the generated models. However, it is important to note that, with the same hyperparameters, the EMOPIA transformer also suffers from long pauses and empty beats, with an EBR of 0.17 compared to our 0.09 with the REMI transformer. Moreover, the REMI transformer gains the closest proximity to the polyphony in the original dataset, with EMOPIA's transformer close behind. Both models also use the similar number of

pitch classes, although the EMOPIA transformer uses a smaller pitch range at the same level. The results indicate that given a similar hyperparameter space and model size, the model can perform better by increasing the sequence length during training, similar to the 1024 tokens used to train the EMOPIA transformer per sequence. The large EBR in the EMOPIA transformer can occur when the model is not trained on the rest beats in the original dataset. We paid care to this and used MidiTok's tokenisation, which offers explicit encoding for rest, chord, and tempo tokens across all representations [25]. This is fruitful as seen in the results of Polyphony (chords) and EBR (empty beat rate). To improve the emotion classification, we can use recurrence mechanisms to improve the memory of the model as seen in the EMOPIA transformer, relying on a global memory instead of local attention-heads.

4.4.1 Conditioned Transformer-GAN vs Baseline

Our conditioned Transformer-GAN achieves comparable results to the baseline papers, using roughly 4-10 times lesser parameters. Our CP word conditioned Transformer-GAN with WGAN-GP loss outperforms the EMOPIA transformer by a margin in the classification accuracy, and has the lowest EBR, suggesting the music is rich with notes. Muhamed et al. achieved a NLL lower than the baseline Transformer-XL model. However, we were able to outperform other Transformer-only models by achieving a NLL loss of 1.2. We achieved this by carefully training the GAN by using optimum loss functions, decaying the learning rate, penalising discriminator overconfidence with gradient penalty, clipping gradients early, propagating the NLL loss, and initialising weights to avoid randomisation. Hence, we were successfully able to generate comparable performance while learning only 20 tokens per sequence, without using recurrence mechanisms or global attention heads. Hence, we find that the Transformer-GAN architecture is worth exploring for conditioned generation, with substantial improvements expected using more complex transformers as generators and discriminators, and longer training sequences.

4.4.2 Representations

Our results indicate that symbolic music representation plays an important role in the performance of the model, where CP word outperforms all other representations across all metrics. This can be attributed to (1) The vocabulary size: models train faster, and hence, converge faster, with shorter vocabulary. (2) The number of tokens required to represent a sequence: 1-D representations such as REMI and Midi-Like explode the sequence length for the same musical events compared to CP word, as they are forced to

use one dimension to represent the all events. CP Word on the other hand can reduce the token length by half by increasing to k-dimensions. (3) The model has to learn all types of events in the same embedding space with REMI and Midi-Like. Midi-Like also has no explicit tokens for chords, which only worsens the performance. In comparison, We were able to train CP Word with different embeddings for different types of tokens, which improves the next token prediction. This is proven as the NLL of CP-word is always lesser than the NLL of other encodings. However, CP Word also encourages polyphony since the next token is sampled with different temperatures for each token type. Explicitly using chords with CP word and rest beats allowed us to improve the accuracy and EBR of the generated sequences. Though, there should be a more careful evaluation of the sampling, which can result in erratic polyphonies.

4.4.3 GAN Loss Functions

When training GANs, the adversarial loss function can play an important role in the performance. We find that the label smoothing + label flipping converges the model to a smaller NLL, however the generated sequences lose the discrimination between the classes. In comparison, WGAN-GP both stabilises training as well as penalises the discriminator for over confidence better than label-flipping. We suspect that label flipping may corrupt the internal representations of each class in the model, which leads to poorer classification accuracy at 20.8%. Other metrics remain in close range, which suggests that model uses the same vocabulary space, but performs better predictions on the model.

4.4.4 Datasets

We also trained the model on two different datasets, EMOPIA and VGMIDI [7] [8]. The model outperforms with the EMOPIA dataset by a large margin. This can be attributed to the dataset size, as the EMOPIA dataset (1078 sample) is 5 times bigger than the VGMIDI dataset (200 samples). This suggests that GANs perform better when trained on larger datasets. With limited datasets, it is favourable to pre-train the models to learn the language model before performing specific tasks [10][7].

4.5 Subjective Test

Since music is a subjective art, we also deployed a subjective human test to understand the perception of the generated music with the general public. 12 pieces of generated music were sampled from the baseline EMOPIA Transformer (as given in the paper), our EMOPIA C-TransGAN and the VGMIDI C-TransGAN models. From each model one sample of 7 seconds from one emotion was used and shuffled to the participants. It is important to note that we chose a sample piece of 7 seconds as that is approximately 20 CP Word tokens during training. We wanted to only use the amount of music the models learnt from, to understand how the length of music affects the perception. The participants were informed the music may or may not be machine generated, but no indication was given if they were. The clip numbers and the classes, as well as the complete result is provided in Appendix H Subjective Results. A total of 15 subjects participated. The results are described as follows:

4.5.1 Demographics

Most of the subjects belonged to the age-group 18-24, as seen in fig.4.5 and listened to music daily, as seen in fig.4.6. Five of the participants had experience in music production, however only one of them was familiar with the Russell's 4Q theory before the survey. They all believed that music has an emotional impact.

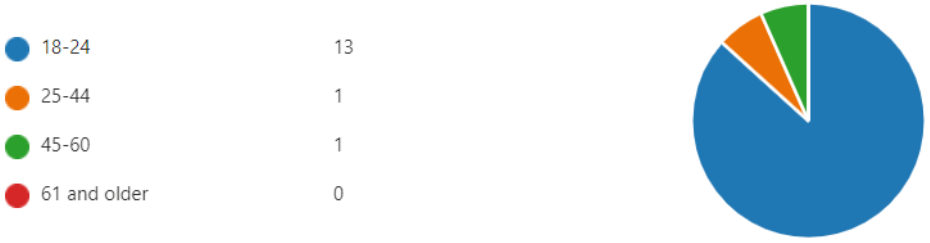


FIGURE 4.5: Age group of the participants

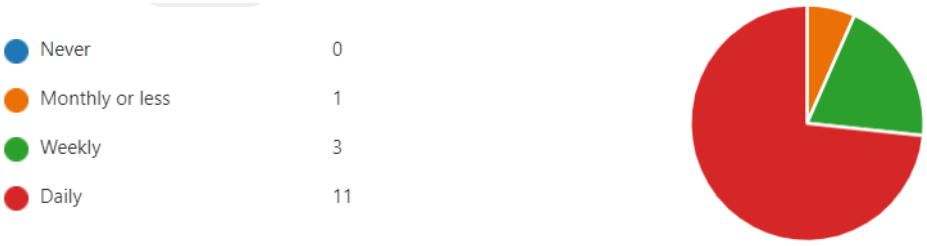


FIGURE 4.6: Participants' music listening habits

4.5.2 Musicality Results

Participants were given open ended questions to rate the pleasantness and interestingness of the music to allow us to understand subjectively how the music was interpreted. Our results are shown in tab. 4.2. We see here how the music generated from our models falls behind the state of the art in achieving pleasant music. Our results suggest that further the model strays away from the musical metrics with the original dataset, the less musical it sounds. One way to improve this is to match the hidden embeddings and features with the target features as well. This could help improve the internal representation, and encourage the generator to generate music inclined to the real dataset. Another way, we can suggest is to pre-train the discriminator on a larger dataset, which would help the discriminator understand the feature map of the music, which can enforce the generator to learn those features quicker as well, as seen in [7].

Model	**Pleasant**	**Interesting**
EMOPIA Transformer	4.75	4.5
EMOPIA C-TransGAN	2.5	4.25
VGMIDI C-TransGAN	2.25	4.75

TABLE 4.2: Results on musical questions

It is interesting to note that although the VGMIDI music was the least pleasant, it was also the most interesting. This contradictory results can stem from the interest improv music encourages, since it sounds more creative than the samples close to real music. However, there is a thin-line between human-improvised music and machine-improvised music, as we see that the VGMIDI and EMOPIA model could easily be distinguished as machine-generated music in fig. 4.7.

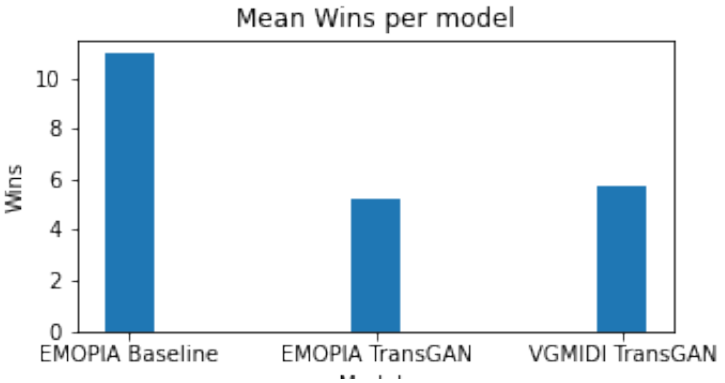


FIGURE 4.7: Here, a win is when the subject classifies the sample as human-generated music

Finally, we also looked into the mean of over all opinion provided for samples from each models, and we found that subjects had a higher opinion on the music provided by the

EMOPIA Transformer. Even while testing with a length of 7 seconds, the EMOPIA transformer outperformed our music. We suspect that with longer training sequences, comes more realistic and pleasant patterns of music. VGMIDI and EMOPIA generated music shared similar scores throughout the musical tests, even though the generated samples were reported with different musical metrics in tab. 4.1. This could further enforce the idea that the generated samples should follow the real dataset closely to generate well-received music, much like the metrics of the state-of-the-art EMOPIA Transformer.

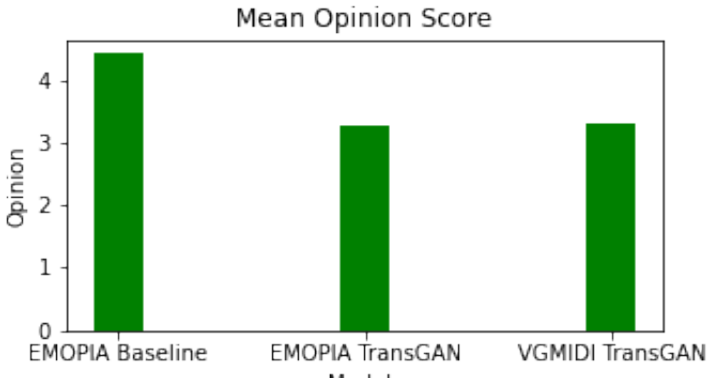
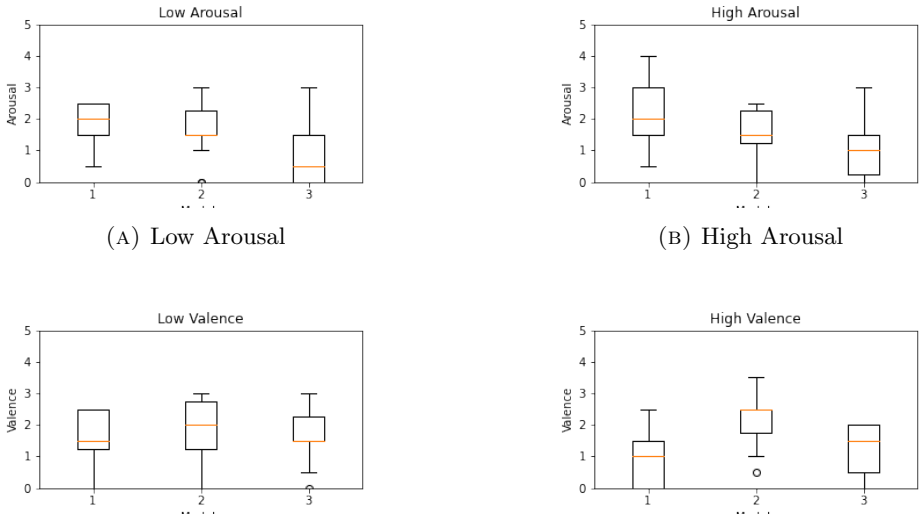


FIGURE 4.8: Mean Opinions reported per model

4.5.3 Emotion Results

We plotted the average valence and arousal given to each piece of sample, as shown in fig. 4.9. We found some really interesting results, especially for valence. Even though our model was lacking the musical cohesiveness, the subjective tests show that the state-of-the-art samples and our samples perform similar in emotion classification. For arousal, overall the EMOPIA and VGMIDI C-TransGAN generated samples perform better in capturing low arousal.

Moreover, our generated samples performed well for high valence classification. This is encouraging as this supports our intuition that chord information can enhance valence in the music. Subjects classified our samples as more positive than the ones generated by the EMOPIA Transformer. The model also shares a similar space for low valence as the state-of-the-art. Our results suggests that the model is capable of learning and producing music conditioned on emotions, which shows great potential for future work.



(A) Low Arousal

(B) High Arousal

(C) Low Valence

(D) High Valence

FIGURE 4.9: The box plot of the subjective scores for different arousal and valence, where model (1) is the EMOPIA Transformer baseline, (2) is the EMOPIA C-TransGAN and (3) is the VGMIDI C-TransGAN

Chapter 5

Conclusion

Through this project, we implemented an Emotion conditioned music generation model, that can generate sequences of any length given an input emotion as the condition. We looked into the current literature, and combined the learnings from different models such as [7], [26],[10], and [12] and came up with our proposal, to create a conditional Transformer-GAN. Our model uses an auto-regressive transformer as a generator and a transformer classifier as a discriminator. We experimented with different music representations, architectures, loss functions, and more. After carefully implementing a GAN to avoid its common pitfalls [65], **our model was able to generate music with a NLL loss of 1.2**. We also found how different representations affect our model, and concluded that **CP Word Encoding can effectively produce music with closer musical metrics than other representations**. Given our computation limits, we aimed to build a hybrid model that can generate music and is still comparable to the existing implementations, while using minimum model size and training length sequence. From our results, we find that our model is comparable to models 2-10 times its size, as **our model achieved a higher classification accuracy on the emotions (33.3%) while achieveing lower NLL compared to Music Transformer and unconditioned Transformer-GAN [12] [10]**. We believe that the approach shows great scope for future work and implementation, where we can truly harness the power of Transformer-GANs

5.1 Requirements Validation

Based on our initial review, we were able to complete all the "must" Functional requirements for the project, as highlighted below:

FR01 Encode MIDI Files - M

- **DONE**

FR02 Implement a Model for Music Generation - M

- **DONE**

FR03 Implement a Model for Emotion-Encoded Music Generation - M

- **DONE**

FR04 Generate Emotion-Encoded Music with Implemented Model - M

- **DONE**

FR05 Conduct Comparative Analysis - M

- **DONE**

FR06 Explore Datasets - S

- **DONE**

FR07 Implement an In-painting Model - C

- **NOT DONE**

FR08 Implement a web interface - C

- **NOT DONE**

5.2 Contributions

Based on our literature review, this is the first attempt of conditioned Transformer-GANs for generating music. This project can serve as a guideline for implementing such models and future steps. We saw our computation limit as an opportunity to explore the effects of the different parts of the implementation on the overall performance. We were able to surpass Muhamed et al.'s Transformer-GAN [10] in their NLL loss, and achieve similar results from our model in the emotion classification accuracy and the subjective tests with the EMOPIA transformer [7]. We also discovered the relationship between training sequence length, music representation, GAN loss functions, and model architectures with the performance. We conclude with the hope that our contributions can bring light to more interesting and complex behaviour in AI-generated music.

5.3 Challenges faced

Our most important challenge was the implementation of the model in the given computation space. Initially, we were expecting to develop the code on Google Colab as it offers a 16GB GPU. However, we discovered early on that the training RAM on Google Colab was not as predictable, as loading the model itself took up nearly 8 GB. Moreover, we also faced difficulty for training the model for longer periods in Google Colab, as the session timed out. One solution for avoiding session time outs is training on a local Jupyter Instance by calling the Google Colab API, however we could not solve the issue regarding the RAM used for building the model. Eventually, we trained on the 8GB RAM in the lab computers, as the memory consumption was more predictable albeit half in quantity. This also forced us to scope-down our project, both in model size and computation time. Hence, our focus was building models with minimum parameters that could still produce comparable results, we also turned our focus on the optimisation of building complex models such as GANs.

Between the month of March and April, there was also a system lockdown on Keyserver. This posed a great challenge since the implementation was done remotely. In total, the lockdown set the project back by two weeks, hence, we were not able to train and refine our model as expected. Nevertheless, we attempted to continue with our implementation, and focused on faster results. This gave us an opportunity of learning tricks and tips for modeling with GANs and Transformers.

5.4 Future Work

Although the model poses potential, there is still much to do in the future work to surpass the current state-of-the-art by a large margin. Our first step would be to use stronger computers, or parallelising the code on multiple GPUs, as we have largely exhausted our options within the current development space.

Based on our results, we propose the following solutions to improve the performance, given there is no explicit computation limit. The first would be to increase sequence length and observe the effect of training on longer sequences in the musicality of the model. We also found lack of long term dependency (largely due to the short training sequences). We can resolve issues of long-term dependency and relative positional encoding with Transformer-XL [66], as the implementation was too heavy for our current computation space. Traditional transformers recalculate the memory (internal states) at each run, whereas recurrent Transformers like Transformer-XL reuse previous memory, which accumulates with each run and requires more GPU RAM. We also want to try

using pre-trained Discriminators, especially BERT used by [10]. A fun implementation would be to use multiple generators for each token type of the CP Word encodings, and then use the same discriminator to classify. We expect further issues with the dependency of the tokens by separating the generator, which we hypothesize we can try to mitigate by using a generator for each token family instead, which would reduce the learning space for the generator, making it easier for it to learn quickly.

Overall, the field of music generation with AI still requires a lot more research, especially when it comes to our emotional attachment and identification to music. We hope that this project inspires someone to pick up an instrument, or sit down to code, to really understand and effectively use AI-music generation in the current world of pop music.

Appendix A

Background Details

A.1 Musical Notation

The leftmost symbol placed at the beginning of the piece in fig. A.1 defines the *key* of the music. *A musical key* is the group of predefined notes beginning at a specific note, namely the *home note*. In fig A.2, we see the C major key, where the home note is C. In a piano, each black and white key is referred to as a half step, so two consecutive piano keys are referred to as a whole step. Musical keys usually follow either of the two notation:

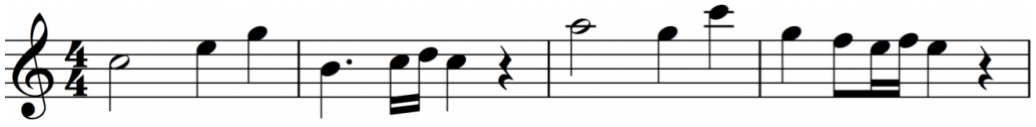


FIGURE A.1: A piece of the score from Mozart – Piano Sonata No. 16, K. 545

Major Notation $W,\ W,\ h,\ W,\ W,\ W,\ h$

Minor Notation $W,\ h,\ W,\ W,\ h,\ W,\ W$

Where W is a whole step and h is a half step.

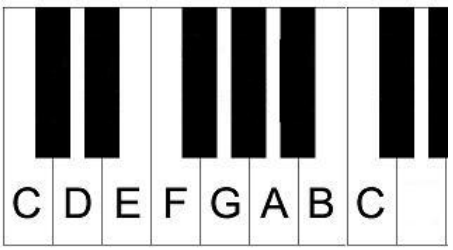


FIGURE A.2: The C Major Key pattern

A.2 Artificial Neural Networks

ANN are fundamental in deep learning. They are biologically inspired algorithms that imitate human neural networks. Similar to how human brains pass information through neurons, ANN pass information through layers of nodes. A human neuron and an artificial node are compared in fig. A.3 and fig. A.4 The three types of layers are: (1) Input layer - Which receives the input data directly, (2) Hidden layer - which learns the information and (3) Output layer - which produces a decision or output on the information. Deep Neural Networks consist of more than 1 hidden layer, so they can model non-linear problems.

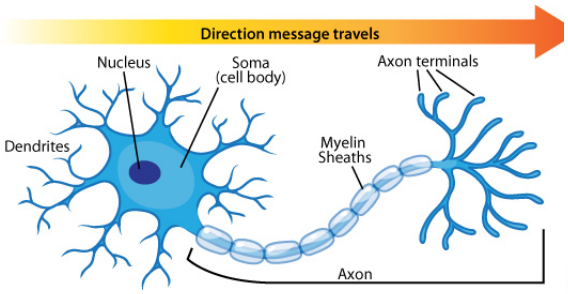


FIGURE A.3: A human neuron. A message from a neuron is input through the dendrite, and output from the axon terminals to other neurons. [13]

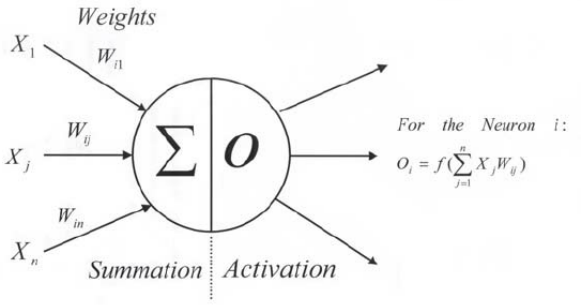


FIGURE A.4: An artificial node. Weights and bias are applied on the input, and the activation function over it for the output [14]

Some key concepts in neural networks are:

- ***Weights and Bias***: Weights (W) define the strength of connection, by affecting the value of the input. The bias (b) is unaffected by the input, and adds the layer's own offset to the input. These values are usually initialised randomly, then learnt over each pass of information
- ***Activation functions***: After applying the weight and bias, the values are summed from each connection and fed into an activation function. Activation functions are mathematical functions that normalise the output and decide whether to "fire" a node. In direct terms, they decide whether the output is relevant or not. Some popular types of activation functions are sigmoid, reLU, and tanh, illustrated in fig. A.5

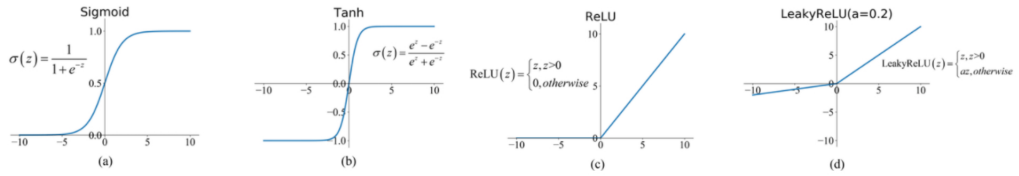


FIGURE A.5: Different Activation functions [15]

- ***Feed Forward***: A single pass of information from the input node to the output node. It is the simplest type of neural network, where data only moves in one direction
- ***Loss Functions***: Once we have an output from a feed forward pass, we evaluate the loss of information with loss functions. Popular loss functions include cross entropy (which is used in classification algorithms).
- ***Back propagation***: To actually learn from our loss, we use back propagation to go backwards and update the weights and bias to reduce loss over time. This optimisation is usually done with a gradient descent algorithm.

Appendix B

Consent Form

Consent to Act as a Subject in an Experimental Study

The purpose of this document is to request your consent in an experimental study for evaluating music generated with artificial intelligence. To protect you from undesirable circumstances, we advice you to NOT consider participation if you: (1) identify as a vulnerable participant (including using learning support), (2) Are sensitive to audio or face any hearing difficulty, and (3) Are sensitive to questions around emotions, specifically stemming from music. All of your information will be kept confidential in a password-protected computer directory. No identifiable information will be collected. Your participation will not affect your relationship with the university in any way. You have the right to decline or withdraw from the study at any point. Such a decision will not adversely affect your status with the university in any way. In this study, you will be asked to:

- Listen to several short clips of music
- Give your opinion on the music clip, and categorise as different emotions
- Categorise each as composed by humans or artificial intelligence

Voluntary consent:

I certify that I have read the preceding text and that I understand its contents. I understand I can ask any questions related to the study to the email of the investigator: ms374@hw.ac.uk. My signature below certifies that I have freely and willingly agreed to participate in the study. Lastly, I consent to the publication and distribution of the

results of the survey, for scientific and research purposes only, as long as my personal identity is not revealed.

Subject Signature: _____

Appendix C

Pre-Survey

1. What age group do you belong in?

   O 18 - 24

   O 25 - 44

   O 45 - 60

   O 61 and older

2. How often do you listen to music?

   O Never

   O Once a month or less

   O Once a week

   O Several times a week

   O Once a day

   O Several times a day

3. Do you have any experience in producing music?

   O Yes

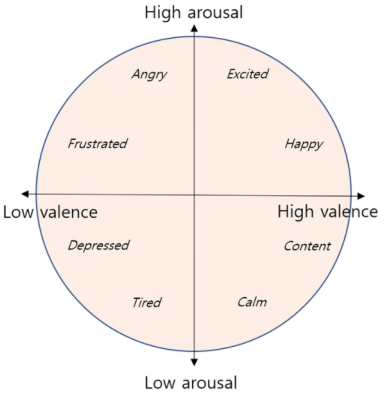
   O No

4. Do you believe music has an emotional impact on you?

   O Yes

   O No

5. Rusell's 4Q is a model to classify emotions on a two dimensional plane. The dimensions are arousal, which describes how intense the emotion is, and valence, which describes how pleasant the emotion is. Increasing arousal or valence increase the intensity and pleasantness respectively. Emotions can be categorised in one of four quadrants in the plane, namely: HVHA (High Valence High Arousal), HVLA (High Valence Low Arousal), LVHA (Low Valence High Arousal) and LVLA (Low Valence Low Arousal). An idea of the placement of common emotions is given below. Do you have any experience with this categorisation?



O Yes

O No

6. In this study, you will be asked to score each clip on its valence and arousal. To familiarise you with the emotions associated with each, we will be giving you samples of music from the different quadrants. You can refer to the samples and diagram at any point in the survey.

Appendix D

Questionnaire

Questionnaire

Now you will be given a few short samples of music. For each clip, please answer the following questions:

1. Is the music pleasant to hear? Here one is not at all pleasant and 5 is very pleasant

   O 1 O 2 O 3 O 4 O 5

2. Is the music interesting to hear? Here one is not at all interesting and 5 is very interesting

   O 1 O 2 O 3 O 4 O 5

3. Please score the music on the valence or positivity, where one is lowest or not at all positive and 5 is the highest or completely positive:

   O 1 O 2 O 3 O 4 O 5

4. Please score the music on the arousal or intensity, where one is lowest or not at all intense and 5 is the highest or very intense:

   O 1 O 2 O 3 O 4 O 5

5. How is the overall quality of the music? Here one is not at very bad and 5 is very good

   O 1 O 2 O 3 O 4 O 5

6. Do you this music is composed by a human being or a machine?

   O Human being

   O A Machine

Appendix E

Appendix E Methodology

This chapter looks into the project planning, risk management and the ethical and legal risks involved in the project

E.1 Development Methodology

For this project, we plan to develop the project with SCRUM framework in Agile methodology. SCRUM follows an iterative approach, which gives us a flexibility of developing in small increments called sprints. The project is managed by the SCRUM master, who also conducts the Sprint Review meetings. This meeting reviews the progress, monitor risks and plans ahead at the end of a sprint. Our main advantages of using SCRUM framework compared to other traditional frameworks are as follows:

- **More control over development**: We can achieve greater control over the entire project by reducing them into digestible tasks over a short sprint.
- **Monitoring risk**: We can use the Sprint Review meeting to assess any risks identified weekly
- **Communicating Progress**: We can discuss the project progress with the supervisor in the Sprint Review meetings, which gives us a direct and established line of communication for assessing the increments and discussing further development.

E.2 Project Plan

Based on our development methodology, the following Gantt chart in fig.E.1 was created as our project plan.

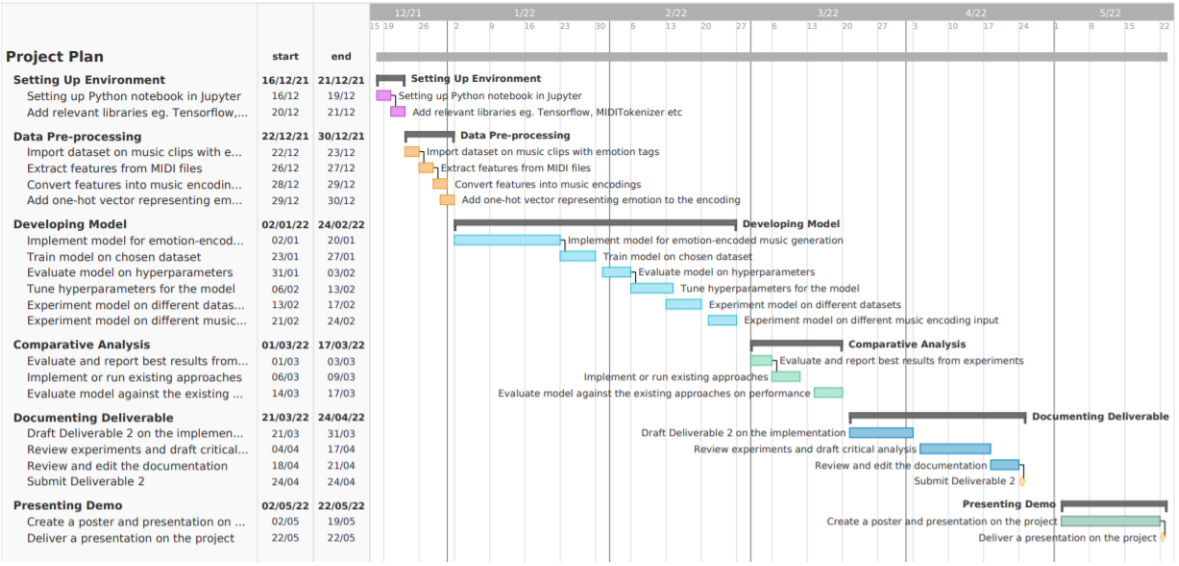


FIGURE E.1: Project Plan

E.3 Risk Management

Risk management is necessary to avoid risks affecting our project negatively. Four key steps are covered in risk management to do so:

- **Identification**: Identifying and categorising risks for the project. The following risk categories are used: Technology, People, Tools, Organisational
- **Analysis**: Analysing and prioritising the risks using the likelihood and impact. The likelihood and impact can be either low, medium or high
- **Planning**: Planning to cover mitigation and contingencies for the risks
- **Monitoring**: Identifying when the risks happen by monitoring and avoiding them.

Based on these steps, we developed the following risk management plan for the project as seen in table E.1

ID	Risk, Type, Priority	Monitoring and Avoidance	Contingency and Mitigation
01	**Tool or library discontinued** Type: Tools Likelihood Impact	**M**: Keep track of tool versions **A**: Use open source tools	**C**: Use alternative tools **M**: Check versions of alternative open sourced tools and use
02	**Cannot implement baseline models from lit. review** Type: Technology Likelihood Impact	**M**: No implementation or documentation found **A**: Use papers with well-explained models	**C**: Use alternative papers of similar architectures **M**: Use alternative approach for undocumented steps
03	**Music representation does not work on the model** Type: Technology Likelihood Impact	**M**: Poor performance metric **A**: Use tried and tested representations in the lit. review	**C**: Modify encoding to fit the model or vice-versa **M**: Follow encoding used in similar models
04	**Dataset does not perform well on the model** Type: Technology Likelihood Impact	**M**: Poor performance metric **A**: Use evaluated datasets	**C**: Use a different dataset from lit. review **M**: Use a subset of the data or alternate dataset
05	**Proposed architecture cannot effectively generate emotion-conditioned music** Type: Technology Likelihood Impact	**M**: Poor performance metric **A**: Make similar architecture decisions as in the lit. review	**C**: Use other architectures seen in lit. review **M**: Research similar models, test alternative steps where possible
06	**Generated music is not the input emotion** Type: Technology Likelihood Impact	**M**: Input and Training sample are classified differently **A**: Pre-train the MER discriminator to recognise emotions well	**C**: Use alternative approach to emotion-conditioned music **M**: Follow existing architecture as in the lit. review
07	**Data Loss** Type: Technology Likelihood Impact	**M**: Cannot recover data **A**: Use version-control and back up files	**C**: Redo the implementation **M**: Use documentation for the re-implementation
08	**Project plan lags** Type: Organisational Likelihood Impact	**M**: Progress does not match project plan **A**: Start early and keep track	**C**: Review and revise the plan **M**: Review current circumstance before revising the plan
09	**Unfamiliar with project tools** Type: Tools Likelihood Impact	**M**: Cannot use tools effectively **A**: Test the tools before implementation	**C**: Use alternative, familiar tools **M**: Follow tool documentation
10	**Not enough computation power** Type: Tools Likelihood Impact	**M**: Computation need exceeds machine limit **A**: Implement computationally inexpensive models	**C**: Use HWU key-server **M**: Reduce parameters in the model

TABLE E.1: Risk Management Table for the Project

E.4 Professional, Legal, Ethical and Social Issues

1. **Professional and legal issues**: Research papers and code used in the project will be referenced. Our model might pick up the styles of some artists in the

dataset, however the MIDI files give us explicit permission to redistribute and edit the material under the CC-BY 4.0 license. Any music generated by our proposed model will also be under the CC-BY license. The source code will be published under a GNU-LGPL license to allow distributions and modifications with open and closed source software, in hopes of commercial or public interest The music for subjective tests will be from open sourced datasets such as [7] [8]. Any data collected will be under the compliance of GDPR and stored safely.

2. **Ethical and Social Issues**: Before our survey, we will hand out consent forms to define the purpose of the study results and their rights, including but not limited to: (1) their right to withdraw from the study at any point, and (2) their right to ask any questions related to the study. Given the sensitivity of emotion and music to some people, we will avoid any vulnerable participants including those hard of hearing by specifying the risks in the consent form. The collected data will be aggregated and used in the analysis. We address privacy by anonymising the data. We follow the BCS code of conduct, specifically the first key principle.

Appendix F

Appendix F Requirements Analysis

This chapter follows the research question and the requirements analysis of the project

F.1 Research Questions

At the end of the project, we will answer the following questions.

1. Can generative models successfully build a music generation model where the output is conditioned on an emotion input by the end-user?
2. How does different music representation and source affect the quality of generated music?
3. How does a hybrid approach of generative models for conditioned music generation compare to existing models?

F.2 Requirements Analysis

Requirements Analysis is critical to capture the interests of a project. They define the project objectives as actionable tasks and can be prioritised accordingly. The requirements are classified as functional and non-functional, and prioritised as follows:

- Must Have (M) : Requirements that are absolutely necessary for the completion of the project

- Should Have (S) : Requirements that are essential to the project, however do not affect the completion.
- Could Have (C) : Requirements that are not necessary nor essential, but they could add additional benefit to the project

F.2.1 Functional Requirements

The functional requirements of the project define what the system should or should not do. For our project, our functional requirements include reasonably achieving the technical implementations. This is necessary as this carries out the evaluation of the results. To further explore the research questions, we also discuss the requirements for the comparative analysis and what the model can do. We have come up with the following functional requirements and their evaluation strategy

FR01 Encode MIDI Files - M

Pre-process music (MIDI) files, extract features and encode based on the research discussed on music representation in chapter 2

Evaluation: Compare different representations using classification accuracy of valence and arousal

FR02 Implement a Model for Music Generation - M

Implement a model trained on music samples in different representations and predict the next token, using the research conducted in chapter 3 and train based on the primary dataset chosen in section 2.5

Evaluation: Compare the training loss of our model to our baseline LSTM, transformer and transformer-GAN.

FR03 Implement a Model for Emotion-Encoded Music Generation - M

Implement a model to input emotion and output a piece of emotional music, using the research conducted in chapter 3 and train based on the primary dataset chosen in section 2.5

Evaluation: Compare the training loss of our model to our baseline LSTM, transformer and transformer-GAN.

FR04 Generate Emotion-Encoded Music with Implemented Model - M

Generate samples of music with the implementation of our proposed model with an input emotion

Evaluation: Evaluate the generated music using BLEU score, music metrics, confusion matrix and human subject Evaluation as discussed in section 4.1

FR05 Conduct Comparative Analysis - M

Compare the proposed model with existing approaches, specifically [7], [10], [12].

Evaluation: Compare the generated samples of music by the models using BLEU scores and music metrics (ref. 4.1)

FR06 Explore Datasets - S

Train the proposed model on different datasets discussed in 2.5

Evaluation: Compare the generated samples of music by training with different datasets using BLEU scores and music metrics (ref. 4.1)

FR07 Implement an In-painting Model - C

Implement a model to "in-paint" or fill in between two dis-joint pieces of music based on the emotions of both pieces

Evaluation: Evaluate the generated samples by comparing the music metrics on pitch and beats to the input defined in section 4.1

FR08 Implement a web interface - C

Develop a web interface for the project that can be used by end users to input an emotion and output a music sequence

Evaluation: Conduct a user experience study on the interface

F.2.2 Nonfunctional Requirements

Nonfunctional requirements define the constraints and standards of the project. They define the quality of the system. Our quality of the system is based on the guidance and standards for maintaining technical implementations. They also define other quality measures such as expected output. In brief, they describe *how* the system works. We have come up with the following non functional requirements for our project as described below:

NFR01 Comply with GDPR - M

Comply and follow GDPR guidelines for training dataset and conducting human subject evaluation

Evaluation: Check if the dataset is open source and submitted an ethical approval will for the human listening test. The generated music will be under CC-BY license. More details on licensing and ethics is available in section E.4

NFR02 Use Version Control - M

Ensure no previous versions are lost, and changed files are recoverable with a reliable coding platform

Evaluation: Check the coding platform allows version control, and verify the code is backed up every week on GitHub.

NFR03 Write Readable Code - S

Implement the code with adequate comments, proper variable names, and modular approach that can be reused to optimise the code

Evaluation: Since the language of choice for this project is Python, validate the script with the PEP 8 standard style guidelines [76].

NFR04 Produce at least 14 seconds of music - M

The model should be able to produce at leas 14 seconds of music, as the minimum sample length from the EMOPIA transformer is 14 seconds.

Evaluation: During inference, the number of tokens will be controlled to what the desired length.

NFR05 Can run on the given hardware - M

The current hardware is a 8GB GPU from the university computers. The model should be able to complete training and inference within the given limits

Appendix G

Appendix G Architecture

```
  BERTClassifier(
(embedding): Embedding(267, 512)
(in_linear): Linear(in_features=512, out_features=252, bias=True)
(bert): BertModel(
  (embeddings): BertEmbeddings(
    (word_embeddings): Embedding(267, 252, padding_idx=0)
    (position_embeddings): Embedding(50, 252)
    (token_type_embeddings): Embedding(2, 252)
    (LayerNorm): LayerNorm((252,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (encoder): BertEncoder(
    (layer): ModuleList(
      (0): BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
            (query): Linear(in_features=252, out_features=252, bias=True)
            (key): Linear(in_features=252, out_features=252, bias=True)
            (value): Linear(in_features=252, out_features=252, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
            (distance_embedding): Embedding(99, 42)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=252, out_features=252, bias=True)
            (LayerNorm): LayerNorm((252,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
```

```
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=252, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=252, bias=True)
      (LayerNorm): LayerNorm((252,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (1): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=252, out_features=252, bias=True)
        (key): Linear(in_features=252, out_features=252, bias=True)
        (value): Linear(in_features=252, out_features=252, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (distance_embedding): Embedding(99, 42)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=252, out_features=252, bias=True)
        (LayerNorm): LayerNorm((252,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=252, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=252, bias=True)
      (LayerNorm): LayerNorm((252,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (2): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=252, out_features=252, bias=True)
```

```
          (key): Linear(in_features=252, out_features=252, bias=True)
          (value): Linear(in_features=252, out_features=252, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (distance_embedding): Embedding(99, 42)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=252, out_features=252, bias=True)
          (LayerNorm): LayerNorm((252,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=252, out_features=3072, bias=True)
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=252, bias=True)
        (LayerNorm): LayerNorm((252,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (3): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=252, out_features=252, bias=True)
          (key): Linear(in_features=252, out_features=252, bias=True)
          (value): Linear(in_features=252, out_features=252, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (distance_embedding): Embedding(99, 42)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=252, out_features=252, bias=True)
          (LayerNorm): LayerNorm((252,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=252, out_features=3072, bias=True)
      )
      (output): BertOutput(
```

```
            (dense): Linear(in_features=3072, out_features=252, bias=True)
            (LayerNorm): LayerNorm((252,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=252, out_features=252, bias=True)
      (activation): Tanh()
    )
  )
  (out_linear): Linear(in_features=252, out_features=4, bias=True)
)
```

The Transformer Model: Music Generation

```
    REMITransformer(
  (criterion): CrossEntropyLoss()
  (embedding_notes): Embedding(218, 128)
  (embedding_emotion): Embedding(4, 128)
  (in_linear): Linear(in_features=256, out_features=512, bias=True)
  (pos_encoder): PositionalEncoding(
    (dropout): Dropout(p=0.2, inplace=False)
  )
  (linear): Linear(in_features=256, out_features=512, bias=True)
  (encoder): TransformerEncoder(
    (layers): ModuleList(
      (0): TransformerEncoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): _LinearWithBias(in_features=512, out_features=512, bias=True)
        )
        (linear1): Linear(in_features=512, out_features=2048, bias=True)
        (dropout): Dropout(p=0.2, inplace=False)
        (linear2): Linear(in_features=2048, out_features=512, bias=True)
        (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.2, inplace=False)
        (dropout2): Dropout(p=0.2, inplace=False)
```

```
  )
  (1): TransformerEncoderLayer(
    (self_attn): MultiheadAttention(
      (out_proj): _LinearWithBias(in_features=512, out_features=512, bias=True)
    )
    (linear1): Linear(in_features=512, out_features=2048, bias=True)
    (dropout): Dropout(p=0.2, inplace=False)
    (linear2): Linear(in_features=2048, out_features=512, bias=True)
    (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (dropout1): Dropout(p=0.2, inplace=False)
    (dropout2): Dropout(p=0.2, inplace=False)
  )
  (2): TransformerEncoderLayer(
    (self_attn): MultiheadAttention(
      (out_proj): _LinearWithBias(in_features=512, out_features=512, bias=True)
    )
    (linear1): Linear(in_features=512, out_features=2048, bias=True)
    (dropout): Dropout(p=0.2, inplace=False)
    (linear2): Linear(in_features=2048, out_features=512, bias=True)
    (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (dropout1): Dropout(p=0.2, inplace=False)
    (dropout2): Dropout(p=0.2, inplace=False)
  )
  (3): TransformerEncoderLayer(
    (self_attn): MultiheadAttention(
      (out_proj): _LinearWithBias(in_features=512, out_features=512, bias=True)
    )
    (linear1): Linear(in_features=512, out_features=2048, bias=True)
    (dropout): Dropout(p=0.2, inplace=False)
    (linear2): Linear(in_features=2048, out_features=512, bias=True)
    (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (dropout1): Dropout(p=0.2, inplace=False)
    (dropout2): Dropout(p=0.2, inplace=False)
  )
  (4): TransformerEncoderLayer(
    (self_attn): MultiheadAttention(
```

```
      (out_proj): _LinearWithBias(in_features=512, out_features=512, bias=True)
    )
    (linear1): Linear(in_features=512, out_features=2048, bias=True)
    (dropout): Dropout(p=0.2, inplace=False)
    (linear2): Linear(in_features=2048, out_features=512, bias=True)
    (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (dropout1): Dropout(p=0.2, inplace=False)
    (dropout2): Dropout(p=0.2, inplace=False)
  )
  (5): TransformerEncoderLayer(
    (self_attn): MultiheadAttention(
      (out_proj): _LinearWithBias(in_features=512, out_features=512, bias=True)
    )
    (linear1): Linear(in_features=512, out_features=2048, bias=True)
    (dropout): Dropout(p=0.2, inplace=False)
    (linear2): Linear(in_features=2048, out_features=512, bias=True)
    (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (dropout1): Dropout(p=0.2, inplace=False)
    (dropout2): Dropout(p=0.2, inplace=False)
  )
  (6): TransformerEncoderLayer(
    (self_attn): MultiheadAttention(
      (out_proj): _LinearWithBias(in_features=512, out_features=512, bias=True)
    )
    (linear1): Linear(in_features=512, out_features=2048, bias=True)
    (dropout): Dropout(p=0.2, inplace=False)
    (linear2): Linear(in_features=2048, out_features=512, bias=True)
    (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (dropout1): Dropout(p=0.2, inplace=False)
    (dropout2): Dropout(p=0.2, inplace=False)
  )
  (7): TransformerEncoderLayer(
    (self_attn): MultiheadAttention(
      (out_proj): _LinearWithBias(in_features=512, out_features=512, bias=True)
    )
    (linear1): Linear(in_features=512, out_features=2048, bias=True)
```

```
        (dropout): Dropout(p=0.2, inplace=False)
        (linear2): Linear(in_features=2048, out_features=512, bias=True)
        (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.2, inplace=False)
        (dropout2): Dropout(p=0.2, inplace=False)
      )
    )
  )
  (project_notes): Linear(in_features=512, out_features=218, bias=True)
  (project_emo): Linear(in_features=512, out_features=4, bias=True)
)
```

The Conditioned Transformer GAN model:

```
    MidiTransGAN(
  (generator): Generator(
    (criterion): CrossEntropyLoss()
    (embedding_family): Embedding(3, 32)
    (embedding_bar): Embedding(35, 128)
    (embedding_pitch): Embedding(85, 512)
    (embedding_velocity): Embedding(32, 128)
    (embedding_duration): Embedding(66, 256)
    (embedding_chord): Embedding(16, 64)
    (embedding_rest): Embedding(7, 64)
    (embedding_tempo): Embedding(3, 32)
    (embedding_emotion): Embedding(4, 512)
    (in_linear): Linear(in_features=1728, out_features=256, bias=True)
    (pos_encoder): PositionalEncoding(
      (dropout): Dropout(p=0.2, inplace=False)
    )
    (encoder): TransformerEncoder(
      (layers): ModuleList(
        (0): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
            (out_proj): _LinearWithBias(in_features=256, out_features=256, bias=True)
          )
          (linear1): Linear(in_features=256, out_features=2048, bias=True)
```

```
    (dropout): Dropout(p=0.2, inplace=False)
    (linear2): Linear(in_features=2048, out_features=256, bias=True)
    (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    (dropout1): Dropout(p=0.2, inplace=False)
    (dropout2): Dropout(p=0.2, inplace=False)
  )
  (1): TransformerEncoderLayer(
    (self_attn): MultiheadAttention(
      (out_proj): _LinearWithBias(in_features=256, out_features=256, bias=True)
    )
    (linear1): Linear(in_features=256, out_features=2048, bias=True)
    (dropout): Dropout(p=0.2, inplace=False)
    (linear2): Linear(in_features=2048, out_features=256, bias=True)
    (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    (dropout1): Dropout(p=0.2, inplace=False)
    (dropout2): Dropout(p=0.2, inplace=False)
  )
  (2): TransformerEncoderLayer(
    (self_attn): MultiheadAttention(
      (out_proj): _LinearWithBias(in_features=256, out_features=256, bias=True)
    )
    (linear1): Linear(in_features=256, out_features=2048, bias=True)
    (dropout): Dropout(p=0.2, inplace=False)
    (linear2): Linear(in_features=2048, out_features=256, bias=True)
    (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    (dropout1): Dropout(p=0.2, inplace=False)
    (dropout2): Dropout(p=0.2, inplace=False)
  )
  (3): TransformerEncoderLayer(
    (self_attn): MultiheadAttention(
      (out_proj): _LinearWithBias(in_features=256, out_features=256, bias=True)
    )
    (linear1): Linear(in_features=256, out_features=2048, bias=True)
    (dropout): Dropout(p=0.2, inplace=False)
    (linear2): Linear(in_features=2048, out_features=256, bias=True)
    (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
```

```
          (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
          (dropout1): Dropout(p=0.2, inplace=False)
          (dropout2): Dropout(p=0.2, inplace=False)
        )
      )
    )
    (project_family): Linear(in_features=256, out_features=3, bias=True)
    (project_bar): Linear(in_features=256, out_features=35, bias=True)
    (project_pitch): Linear(in_features=256, out_features=85, bias=True)
    (project_velocity): Linear(in_features=256, out_features=32, bias=True)
    (project_duration): Linear(in_features=256, out_features=66, bias=True)
    (project_chord): Linear(in_features=256, out_features=16, bias=True)
    (project_rest): Linear(in_features=256, out_features=7, bias=True)
    (project_tempo): Linear(in_features=256, out_features=3, bias=True)
    (project_emo): Linear(in_features=256, out_features=4, bias=True)
    (proj_cat): Linear(in_features=288, out_features=256, bias=True)
  )
  (discriminator): Discriminator(
    (embedding_family): Embedding(3, 32)
    (embedding_bar): Embedding(35, 128)
    (embedding_pitch): Embedding(85, 512)
    (embedding_velocity): Embedding(32, 128)
    (embedding_duration): Embedding(66, 256)
    (embedding_chord): Embedding(16, 64)
    (embedding_rest): Embedding(7, 64)
    (embedding_tempo): Embedding(3, 32)
    (embedding_emotion): Embedding(4, 512)
    (linear): Linear(in_features=1728, out_features=256, bias=True)
    (pos_encoder): PositionalEncoding(
      (dropout): Dropout(p=0.5, inplace=False)
    )
    (encoder): TransformerEncoder(
      (layers): ModuleList(
        (0): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
            (out_proj): _LinearWithBias(in_features=256, out_features=256, bias=True)
          )
          (linear1): Linear(in_features=256, out_features=128, bias=True)
          (dropout): Dropout(p=0.5, inplace=False)
```

```
    (linear2): Linear(in_features=128, out_features=256, bias=True)
    (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    (dropout1): Dropout(p=0.5, inplace=False)
    (dropout2): Dropout(p=0.5, inplace=False)
  )
  (1): TransformerEncoderLayer(
    (self_attn): MultiheadAttention(
      (out_proj): _LinearWithBias(in_features=256, out_features=256, bias=True)
    )
    (linear1): Linear(in_features=256, out_features=128, bias=True)
    (dropout): Dropout(p=0.5, inplace=False)
    (linear2): Linear(in_features=128, out_features=256, bias=True)
    (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    (dropout1): Dropout(p=0.5, inplace=False)
    (dropout2): Dropout(p=0.5, inplace=False)
  )
  (2): TransformerEncoderLayer(
    (self_attn): MultiheadAttention(
      (out_proj): _LinearWithBias(in_features=256, out_features=256, bias=True)
    )
    (linear1): Linear(in_features=256, out_features=128, bias=True)
    (dropout): Dropout(p=0.5, inplace=False)
    (linear2): Linear(in_features=128, out_features=256, bias=True)
    (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    (dropout1): Dropout(p=0.5, inplace=False)
    (dropout2): Dropout(p=0.5, inplace=False)
  )
  (3): TransformerEncoderLayer(
    (self_attn): MultiheadAttention(
      (out_proj): _LinearWithBias(in_features=256, out_features=256, bias=True)
    )
    (linear1): Linear(in_features=256, out_features=128, bias=True)
    (dropout): Dropout(p=0.5, inplace=False)
    (linear2): Linear(in_features=128, out_features=256, bias=True)
    (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
```

```
          (dropout1): Dropout(p=0.5, inplace=False)
          (dropout2): Dropout(p=0.5, inplace=False)
        )
      )
    )
    (classifier): Linear(in_features=256, out_features=2, bias=True)
  )
  (criterion): BCEWithLogitsLoss()
)
```

Appendix H

Appendix H Subjective Results

Clip Number	Origin	Quadrant
1	EMOPIA baseline	Q1
2	EMOPIA ours	Q2
3	EMOPIA ours	Q4
4	VGMIDI ours	Q3
5	VGMIDI ours	Q1
6	EMOPIA ours	Q3
7	VGMIDI ours	Q2
8	EMOPIA baseline	Q4
9	EMOPIA ours	Q1
10	EMOPIA baseline	Q2
11	VGMIDI ours	Q4
12	EMOPIA baseline	Q3

TABLE H.1: The true clips and quadrant values

H.1 Results

Since we could not fit the entire table, screenshots are added instead

ID	I certify that I have read the preceding text and that I understand its contents. I understand I can ask any questions related to the study to the email of the investigator: ms374@hw.ac.uk. My agr...	What age group do you belong in?	How often do you listen to music?	Do you have any experience in producing music? (Including singing training, writing, creating music)	Do you believe music has an emotional impact on you?	Rusell's 4Q is a model to classify emotions on a two dimensional plane. The dimensions are arousal, which describes how intense the emotion is, and valence, which describes how pleasant the emotio...	Is the music pleasant	Is the music interesting?	Is the music positive? (Low vs High Valence
1	Yes	18-24	Daily	No	Yes	No	Little bit	Very much	Little bit
2	Yes	18-24	Monthly or less	No	Yes	No	Little bit	Little bit	Not much
3	Yes	18-24	Weekly	Yes	Yes	No	Very much	Very much	Little bit
4	Yes	45-60	Daily	No	Yes	No	Little bit	Very much	Little bit
5	Yes	18-24	Daily	No	Yes	No	Little bit	Very much	Little bit
6	Yes	18-24	Daily	Yes	Yes	Yes	Very much	Little bit	Little bit
7	Yes	18-24	Daily	No	Yes	No	Little bit	Not much	Little bit
8	Yes	18-24	Daily	Yes	Yes	No	Neither	Neither	Neither
9	Yes	18-24	Weekly	No	Yes	No	Little bit	Little bit	Little bit
10	Yes	18-24	Weekly	No	Yes	No	Very much	Very much	Very much
11	Yes	18-24	Daily	Yes	Yes	No	Neither	Very much	Not much
12	Yes	25-44	Daily	No	Yes	No	Little bit	Very much	Little bit
13	Yes	18-24	Daily	No	Yes	No	Not much	Little bit	Not much
14	Yes	18-24	Daily	No	Yes	No	Neither	Little bit	Little bit
15	Yes	18-24	Daily	Yes	Yes	No	Little bit	Little bit	Little bit

Is the music intense? (Low vs High Arousal	What is the overall quality of the music	Do you this music is composed by a human being or a machine?	Is the music pleasant?2	Is the music interesting?2	Is the music positive? (Low vs High Valence	Is the music intense? (Low vs High Arousal	What is the overall quality of the music?	Do you this music is composed by a human being or a machine?	Is the music pleasant?3
Very much	4	Human being	Neither	Very much	Neither	Very much	3	Machine	Little bit
Not much	5	Human being	Not at all	Very much	Not much	Very much	5	Machine	Not much
Not much	5	Human being	Not at all	Not much	Not at all	Little bit	3	Machine	Neither
Very much	5	Human being	Very much	Very much	Very much	Very much	5	Human being	Very much
Very much	3	Human being	Not much	Not much	Not much	Very much	3	Machine	Neither
Not much	3	Human being	Not at all	Very much	Not at all	Very much	2	Machine	Not much
Not much	4	Machine	Not much	Little bit	Little bit	Not much	3	Machine	Little bit
Not at all	4	I can't guess	Not much	Little bit	Not much	Little bit	4	I can't guess	Not at all
Not much	4	Machine	Neither	Little bit	Neither	Little bit	3	Machine	Not much
Little bit	5	Human being	Not at all	Not much	Not at all	Not at all	2	Machine	Little bit
Little bit	4	Human being	Not at all	Not much	Not at all	Very much	1	Human being	Not at all
Not much	4	Machine	Not much	Little bit	Not much	Very much	4	Machine	Little bit
Little bit	3	Human being	Not at all	Not much	Not at all	Not much	1	Machine	Not at all
Neither	5	Human being	Not much	Little bit	Not much	Very much	2	Machine	Neither
Not at all	3	Machine	Not much	Neither	Neither	Very much	3	Machine	Neither

Is the music interesting?3	Is the music positive? (Low vs High Valence)2	Is the music intense? (Low vs High Arousa	What is the overall quality of the music?	Do you this music is composed by a human being or a machine?	Is the music pleasant?4	Is the music interesting?4	Is the music positive? (Low vs High Valence)3	Is the music intense? (Low vs High Arousal	What is the overall quality of the music?
Little bit	Little bit	Very much	3	Machine	Little bit	Very much	Neither	Very much	4
Very much	Not much	Little bit	5	Machine	Not much	Very much	Not much	Little bit	5
Neither	Not at all	Very much	3	Human being	Little bit	Little bit	Little bit	Neither	4
Very much	Little bit	Little bit	5	Human being	Not much	Very much	Little bit	Very much	5
Little bit	Not much	Very much	3	Machine	Not much	Little bit	Not much	Little bit	3
Not much	Not much	Little bit	2	Human being	Not at all	Not much	Neither	Neither	2
Little bit	Little bit	Little bit	3	Human being	Not much	Not much	Little bit	Not much	2
Little bit	Not much	Little bit	4	I can't guess	Neither	Very much	Not at all	Very much	4
Neither	Not much	Little bit	4	I can't guess	Not much	Not much	Neither	Very much	3
Little bit	Neither	Not much	3	Machine	Neither	Little bit	Not much	Not much	2
Not much	Not at all	Little bit	1	Human being	Not at all	Not at all	Not much	Little bit	1
Very much	Little bit	Very much	5	Human being	Not much	Not much	Not much	Little bit	4
Not at all	Not at all	Not at all	1	Machine	Not at all	Neither	Not much	Little bit	2
Neither	Not much	Very much	2	Machine	Neither	Neither	Neither	Little bit	3
Little bit	Neither	Little bit	3	Machine	Very much	Neither	Not at all	Neither	3

Do you this music is composed by a human being or a machine?	Is the music pleasant?5	Is the music interesting?5	Is the music positive? (High Valence)	Is the music intense? (High Arousal)	What is the overall quality of the music?	Do you this music is composed by a human being or a machine?	Is the music pleasant?6	Is the music interesting?6	Is the music positive? (High Valence)2
I can't guess	Neither	Very much	Little bit	Little bit	3	Human being	Not much	Very much	Little bit
Machine	Not much	Very much	Not much	Little bit	5	Machine	Not much	Very much	Not much
Machine	Little bit	Little bit	Little bit	Not much	4	Human being	Little bit	Little bit	Little bit
Human being	Not much	Very much	Not much	Very much	5	Human being	Not much	Very much	Not much
Machine	Not much	Little bit	Little bit	Very much	3	Human being	Little bit	Very much	Little bit
Human being	Neither	Neither	Neither	Little bit	3	Machine	Not much	Little bit	Not much
Machine	Little bit	Little bit	Little bit	Neither	4	Machine	Not much	Little bit	Neither
I can't guess	Not much	Little bit	Neither	Little bit	4	I can't guess	Not at all	Little bit	Not at all
Machine	Little bit	Neither	Neither	Little bit	4	I can't guess	Not much	Little bit	Not much
Human being	Little bit	Little bit	Not much	Little bit	2	Machine	Not much	Little bit	Not much
Human being	Not much	Not much	Not much	Neither	2	Human being	Not at all	Not at all	Not much
Machine	Very much	Very much	Little bit	Little bit	5	Human being	Very much	Very much	Little bit
Machine	Not much	Very much	Not much	Very much	3	Human being	Not much	Very much	Very much
Machine	Not much	Neither	Neither	Little bit	2	Machine	Neither	Little bit	Neither
Machine	Little bit	Very much	Very much	Very much	4	Machine	Very much	Very much	Very much

Is the music positive? (High Valence)2	Is the music intense? (High Arousal)2	What is the overall quality of the music?	Do you this music is composed by a human being or a machine?	Is the music pleasant?7	Is the music interesting?7	Is the music positive? (High Valence)3	Is the music intense? (High Arousal)3	What is the overall quality of the music?	Do you this music is composed by a human being or a machine?
Little bit	Very much	4	Human being	Little bit	Little bit	Little bit	Little bit	3	Machine
Not much	Little bit	5	Human being	Neither	Very much	Neither	Little bit	5	Human being
Little bit	Neither	4	Human being	Very much	Very much	Very much	Neither	5	Machine
Not much	Very much	5	Human being	Very much	Very much	Very much	Little bit	5	Human being
Little bit	Little bit	4	Human being	Neither	Little bit	Not much	Not much	3	Human being
Not much	Little bit	3	Human being	Not at all	Neither	Not much	Little bit	2	Machine
Neither	Little bit	4	Machine	Little bit	Neither	Little bit	Not much	3	Machine
Not at all	Little bit	4	Machine	Not much	Not much	Not much	Neither	4	Machine
Not much	Little bit	3	Human being	Neither	Not much	Not much	Little bit	2	Machine
Not much	Little bit	3	Machine	Not much	Neither	Little bit	Little bit	3	Machine
Not much	Little bit	1	Human being	Not much	Not much	Not much	Little bit	1	Human being
Little bit	Not at all	5	Human being	Very much	Very much	Not much	Neither	5	Machine
Very much	Very much	4	Human being	Not much	Little bit	Very much	Very much	2	I can't guess
Neither	Very much	2	Machine	Neither	Little bit	Neither	Little bit	3	Human being
Very much	Very much	5	Human being	Neither	Neither	Neither	Little bit	3	Machine

Is the music pleasant?8	Is the music interesting?8	Is the music positive? (High Valence)4	Is the music intense? (High Arousal)4	What is the overall quality of the music5	Do you this music is composed by a human being or a machine?	Is the music pleasant?9	Is the music interesting?9	Is the music positive? (High Valence)5	Is the music intense? (High Arousal)5
Very much	Little bit	Very much	Not much	5	Human being	Not much	Little bit	Neither	Very much
Very much	Very much	Very much	Not much	5	Human being	Not much	Very much	Not at all	Little bit
Very much	Very much	Very much	Not at all	5	Human being	Neither	Neither	Not much	Little bit
Not much	Little bit	Little bit	Not much	5	Human being	Little bit	Very much	Not much	Little bit
Very much	Very much	Very much	Not at all	5	Human being	Not much	Neither	Not much	Not much
Very much	Very much	Very much	Not much	5	Human being	Not much	Very much	Not much	Very much
Very much	Not much	Little bit	Not much	4	Human being	Not at all	Not much	Neither	Little bit
Very much	Little bit	Little bit	Neither	4	I can't guess	Not much	Not much	Not much	Little bit
Little bit	Not much	Little bit	Not much	4	Machine	Not much	Not much	Not much	Little bit
Very much	Very much	Very much	Not much	5	Human being	Not much	Very much	Not much	Very much
Very much	Very much	Very much	Not at all	4	Human being	Not at all	Not at all	Not at all	Very much
Very much	Little bit	Little bit	Not at all	3	Machine	Very much	Very much	Little bit	Very much
Very much	Very much	Very much	Not at all	5	Machine	Not much	Very much	Not much	Very much
Very much	Little bit	Little bit	Neither	5	Human being	Not much	Neither	Neither	Little bit
Very much	Very much	Not much	Not at all	5	Human being	Not at all	Not at all	Neither	Little bit

What is the overall quality of the music5	Do you this music is composed by a human being or a machine?	Is the music pleasant?10	Is the music interesting?10	Is the music positive? (High Valence)6	Is the music intense? (High Arousal)6	What is the overall quality of the music1	Do you this music is composed by a human being or a machine?	Is the music pleasant?11	Is the music interesting?11
3	Machine	Very much	Little bit	Very much	Not much	5	Human being	Little bit	Little bit
5	Machine	Very much	Very much	Very much	Not much	5	Human being	Not much	Very much
2	Machine	Very much	Very much	Very much	Not at all	5	Human being	Not much	Little bit
5	Human being	Little bit	Little bit	Neither	Not much	5	Human being	Not much	Little bit
3	Machine	Very much	Very much	Very much	Very much	5	Human being	Little bit	Little bit
4	Human being	Very much	Very much	Little bit	Neither	5	Human being	Not at all	Not much
3	Machine	Little bit	Little bit	Little bit	Little bit	4	Human being	Not much	Little bit
4	I can't guess	Very much	Little bit	Very much	Neither	4	I can't guess	Neither	Not much
2	Machine	Little bit	Little bit	Little bit	Neither	4	I can't guess	Neither	Neither
4	Human being	Very much	Very much	Very much	Not much	5	Human being	Not much	Not much
1	Human being	Little bit	Neither	Little bit	Not at all	3	Human being	Not much	Not much
5	Human being	Little bit	Not much	Neither	Not at all	3	Machine	Very much	Very much
3	I can't guess	Very much	Very much	Very much	Not much	4	Machine	Not much	Little bit
2	Machine	Very much	Very much	Very much	Little bit	5	Human being	Little bit	Little bit
3	Machine	Very much	Very much	Very much	Very much	5	Human being	Not at all	Neither

Is the music positive? (High Valence)7	Is the music intense? (High Arousal)7	What is the overall quality of the music2	Do you this music is composed by a human being or a machine?	Is the music pleasant?12	Is the music interesting?12	Is the music positive? (High Valence)8	Is the music intense? (High Arousal)8	What is the overall quality of the music3	Do you this music is composed by a human being or a machine?
Little bit	Little bit	4	I can't guess	Very much	Very much	Very much	Not much	5	Human being
Not much	Little bit	5	Human being	Very much	Very much	Neither	Not much	5	Human being
Neither	Neither	4	Machine	Very much	Very much	Very much	Not at all	5	Machine
Not much	Little bit	5	Human being	Not at all	Little bit	Not much	Not much	5	Human being
Not much	Little bit	3	Human being	Very much	Very much	Very much	Not much	5	Human being
Not much	Little bit	2	Machine	Little bit	Little bit	Neither	Not at all	4	Human being
Neither	Little bit	3	Human being	Very much	Neither	Little bit	Not much	4	Human being
Not much	Not much	3	Machine	Little bit	Very much	Not much	Not much	4	Human being
Little bit	Little bit	4	Machine	Little bit	Neither	Little bit	Little bit	4	I can't guess
Not much	Little bit	2	Machine	Very much	Very much	Not much	Not much	5	Human being
Not much	Little bit	2	Human being	Little bit	Little bit	Not much	Not much	3	Human being
Little bit	Not at all	5	Human being	Little bit	Not much	Not much	Very much	1	Machine
Not much	Very much	3	Machine	Very much	Very much	Very much	Not at all	4	Human being
Neither	Little bit	4	Human being	Very much	Very much	Very much	Neither	5	Human being
Neither	Very much	3	Machine	Very much	Very much	Neither	Not much	5	Human being

Bibliography

[1] Y.-S. Huang and Y.-H. Yang, "Pop music transformer: Beat-based modeling and generation of expressive pop piano compositions," in *Proceedings of the 28th ACM International Conference on Multimedia*, 2020, pp. 1180–1188.

[2] 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 preprint arXiv:2101.02402*, 2021.

[3] H.-W. Dong, W.-Y. Hsiao, and Y.-H. Yang, "Pypianoroll: Open source python package for handling multitrack pianoroll," *Proc. ISMIR. Late-breaking paper;[Online] https://github. com/salu133445/pypianoroll*, 2018.

[4] M. S. Lee, Y. K. Lee, D. S. Pae, M. T. Lim, D. W. Kim, and T. K. Kang, "Fast emotion recognition based on single pulse ppg signal with convolutional neural network," *Applied Sciences*, vol. 9, no. 16, p. 3355, 2019.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[6] F. Nejati, N. Ravikumar, S. Vesal, F. Kemeth, M. Struck, and A. Maier, "The effect of data augmentation on classification of atrial fibrillation in short single-lead ecg signals using deep neural networks," 02 2020.

[7] H.-T. Hung, J. Ching, S. Doh, N. Kim, J. Nam, and Y.-H. Yang, "Emopia: A multi-modal pop piano dataset for emotion recognition and emotion-based music generation," *arXiv preprint arXiv:2108.01374*, 2021.

[8] L. N. Ferreira and J. Whitehead, "Learning to generate music with sentiment," *arXiv preprint arXiv:2103.06125*, 2021.

[9] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*. PMLR, 2017, pp. 214–223.

[10] A. Muhamed, L. Li, X. Shi, S. Yaddanapudi, W. Chi, D. Jackson, R. Suresh, Z. C. Lipton, and A. J. Smola, "Symbolic music generation with transformer-gans," in *35th AAAI Conference on Artificial Intelligence, AAAI 2021*, 2021.

[11] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *Advances in neural information processing systems*, vol. 29, 2016.

[12] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, N. Shazeer, C. Hawthorne, A. Dai, M. Hoffman, and D. Eck, "Music transformer: Generating music with long-term structure (2018)," *arXiv preprint arXiv:1809.04281*, 2018.

[13] B. Szymik, "Nerve cell," 2011. [Online]. Available: https://askabiologist.asu.edu/neuron-anatomy

[14] A. Hasan, H. Al-Assadi, and A. Azlan, *Neural Networks Based Inverse Kinematics Solution for Serial Robot Manipulators Passing Through Singularities*. IntechOpen, 04 2011.

[15] J. Feng, X. He, Q. Teng, C. Ren, H. Chen, and Y. Li, "Reconstruction of porous media from extremely limited information using conditional generative adversarial networks," *Physical Review E*, vol. 100, 09 2019.

[16] Y.-H. Chou, I.-C. Chen, C.-J. Chang, J. Ching, and Y.-H. Yang, "MidiBERT-Piano: Large-scale pre-training for symbolic music understanding," *arXiv preprint arXiv:2107.05223*, 2021.

[17] D. Hesmondhalgh, *Why music matters*. John Wiley & Sons, 2013.

[18] L. B. Meyer, *Emotion and meaning in music*. University of chicago Press, 2008.

[19] P. N. Juslin, J. A. Sloboda *et al.*, "Music and emotion," *Theory and research*, 2001.

[20] C. L. Krumhansl, "Music: A link between cognition and emotion," *Current directions in psychological science*, vol. 11, no. 2, pp. 45–50, 2002.

[21] L. B. Meyer, *Style and music: Theory, history, and ideology*. University of Chicago Press, 1996.

[22] G. Loy, "Musicians make a standard: the midi phenomenon," *Computer Music Journal*, vol. 9, no. 4, pp. 8–26, 1985.

[23] S. Oore, I. Simon, S. Dieleman, D. Eck, and K. Simonyan, "This time with feeling: Learning expressive musical performance," *Neural Computing and Applications*, vol. 32, no. 4, pp. 955–967, 2020.

[24] A. Huang and R. Wu, "Deep learning for music," *arXiv preprint arXiv:1606.04930*, 2016.

[25] N. Fradet, J.-P. Briot, F. Chhel, A. E. F. Seghrouchni, and N. Gutowski, "Miditok: A python package for midi file tokenization," in *Extended Abstracts for the Late-Breaking Demo Session of the 22nd International Society for Music Information Retrieval Conference*, 2021.

[26] D. Makris, K. R. Agres, and D. Herremans, "Generating lead sheets with affect: A novel conditional seq2seq framework," *arXiv preprint arXiv:2104.13056*, 2021.

[27] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, "Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

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

[29] P. Ekman, "Facial expressions of emotion: New findings, new questions," 1992.

[30] ——, "Basic emotions," *Handbook of cognition and emotion*, vol. 98, no. 45-60, p. 16, 1999.

[31] J. A. Russell, "A circumplex model of affect." *Journal of personality and social psychology*, vol. 39, no. 6, p. 1161, 1980.

[32] R. Plutchik, "The nature of emotions: Human emotions have deep evolutionary roots, a fact that may explain their complexity and provide tools for clinical practice," *American scientist*, vol. 89, no. 4, pp. 344–350, 2001.

[33] A. Mehrabian and J. A. Russell, *An approach to environmental psychology.* the MIT Press, 1974.

[34] E. Alpaydin, *Introduction to machine learning.* MIT press, 2020.

[35] S. Russell and P. Norvig, "Artificial intelligence: a modern approach," 2002.

[36] L. R. Medsker and L. Jain, "Recurrent neural networks," *Design and Applications*, vol. 5, pp. 64–67, 2001.

[37] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning.* PMLR, 2013, pp. 1310–1318.

[38] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[39] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[40] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, "Musegan: Demonstration of a convolutional gan based model for generating multi-track piano-rolls," *ISMIR Late Breaking/Demos*, 2017.

[41] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[42] R. E. S. Panda, R. Malheiro, B. Rocha, A. P. Oliveira, and R. P. Paiva, "Multi-modal music emotion recognition: A new dataset, methodology and comparative analysis," in *10th International Symposium on Computer Music Multidisciplinary Research (CMMR 2013)*, 2013, pp. 570–582.

[43] J.-C. Wang, H.-Y. Lo, S.-K. Jeng, and H.-M. Wang, "Mirex 2010: Audio classification using semantic transformation and classifier ensemble," in *Proc. of The 6th International WOCMAT & New Media Conference (WOCMAT 2010)*, 2010, pp. 2–5.

[44] D. Bogdanov, M. Won, P. Tovstogan, A. Porter, and X. Serra, "The mtg-jamendo dataset for automatic music tagging," *ICML2019*, 2019.

[45] Y.-H. Yang and H. H. Chen, "Machine recognition of music emotion: A review," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 3, no. 3, pp. 1–30, 2012.

[46] M. Malik, S. Adavanne, K. Drossos, T. Virtanen, D. Ticha, and R. Jarina, "Stacked convolutional and recurrent neural networks for music emotion recognition," *arXiv preprint arXiv:1706.02292*, 2017.

[47] J.-H. Su, T.-P. Hong, Y.-H. Hsieh, and S.-M. Li, "Effective music emotion recognition by segment-based progressive learning," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2020, pp. 3072–3076.

[48] D. Turnbull, L. Barrington, D. Torres, and G. Lanckriet, "Semantic annotation and retrieval of music and sound effects," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 16, no. 2, pp. 467–476, 2008.

[49] S. Hizlisoy, S. Yildirim, and Z. Tufekci, "Music emotion recognition using convolutional long short term memory deep neural networks," *Engineering Science and Technology, an International Journal*, vol. 24, no. 3, pp. 760–767, 2021.

[50] W. Zhao, Y. Zhou, Y. Tie, and Y. Zhao, "Recurrent neural network for midi music emotion classification," in *2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. IEEE, 2018, pp. 2596–2600.

[51] J. Grekow, *From content-based music emotion recognition to emotion maps of musical pieces*. Springer, 2018, vol. 747.

[52] J. H. Juthi, A. Gomes, T. Bhuiyan, and I. Mahmud, "Music emotion recognition with the extraction of audio features using machine learning approaches," in *Proceedings of ICETIT 2019*. Springer, 2020, pp. 318–329.

[53] C. Moruzzi, "Creative ai: Music composition programs as an extension of the composer's mind," in *3rd Conference on" Philosophy and Theory of Artificial Intelligence*. Springer, 2017, pp. 69–72.

[54] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," *arXiv preprint arXiv:1901.02860*, 2019.

[55] M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy, "Spanbert: Improving pre-training by representing and predicting spans," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 64–77, 2020.

[56] G. Hadjeres, F. Pachet, and F. Nielsen, "Deepbach: a steerable model for bach chorales generation," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1362–1371.

[57] H. H. Mao, T. Shin, and G. Cottrell, "Deepj: Style-specific music generation," in *2018 IEEE 12th International Conference on Semantic Computing (ICSC)*. IEEE, 2018, pp. 377–382.

[58] K. Zhao, S. Li, J. Cai, H. Wang, and J. Wang, "An emotional symbolic music generation system based on lstm networks," in *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*. IEEE, 2019, pp. 2039–2043.

[59] G. Athanasopoulos, T. Eerola, I. Lahdelma, and M. Kaliakatsos-Papakostas, "Harmonic organisation conveys both universal and culture-specific cues for emotional expression in music," *Plos one*, vol. 16, no. 1, p. e0244964, 2021.

[60] L. Bonetti and M. Costa, "Musical mode and visual-spatial cross-modal associations in infants and adults," *Musicae Scientiae*, vol. 23, no. 1, pp. 50–68, 2019.

[61] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[62] H.-W. Dong, K. Chen, J. McAuley, and T. Berg-Kirkpatrick, "Muspy: A toolkit for symbolic music generation," *arXiv preprint arXiv:2008.01951*, 2020.

[63] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[64] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.

[65] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[66] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers are rnns: Fast autoregressive transformers with linear attention," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5156–5165.

[67] [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

[68] H. Yao, D.-l. Zhu, B. Jiang, and P. Yu, "Negative log likelihood ratio loss for deep neural network classification," in *Proceedings of the Future Technologies Conference*. Springer, 2019, pp. 276–282.

[69] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[70] X. Chen, P. Cai, P. Jin, H. Wang, X. Dai, and J. Chen, "A discriminator improves unconditional text generation without updating the generator," *arXiv preprint arXiv:2004.02135*, 2020.

[71] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[72] M. Shahbazi, M. Danelljan, D. P. Paudel, and L. Van Gool, "Collapse by conditioning: Training class-conditional gans with limited data," *arXiv preprint arXiv:2201.06578*, 2022.

[73] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," *arXiv preprint arXiv:1701.04862*, 2017.

[74] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[75] C. Hawthorne, A. Stasyuk, A. Roberts, I. Simon, C.-Z. A. Huang, S. Dieleman, E. Elsen, J. Engel, and D. Eck, "Enabling factorized piano music modeling and generation with the maestro dataset," *arXiv preprint arXiv:1810.12247*, 2018.

[76] G. Van Rossum, B. Warsaw, and N. Coghlan, "Pep 8: style guide for python code," *Python. org*, vol. 1565, 2001.