DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Application of CTG Methods in Music Generation

Hakan Akyürek

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Application of CTG Methods in Music Generation

Anwendung von CTG-Methoden zur Musikerzeugung

Author:	Hakan Akyürek
Supervisor:	Prof. Dr. Georg Groh
Advisor:	M.Sc. Tobias Eder
Submission Date:	15.05.2023

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.05.2023 Hakan Akyürek

Acknowledgments

I would like to express my sincere appreciation to M.Sc. Tobias Eder for his invaluable guidance and support throughout this study. His advice, input and feedback were significant in shaping my research, and I am deeply grateful for his assistance.

Moreover, I am thankful to him for providing me with the opportunity to pursue my own ideas in this study. His trust in my abilities has been a tremendous source of motivation, and I am fortunate to have had such a dedicated and knowledgeable mentor.

Abstract

Music has long been regarded as a language; with its own syntax, grammar, and vocabulary, though it tends to be rather repetitive than a natural language. This is especially clear when musical pieces are represented in a symbolic way. Accordingly, applications of sequence models such as LSTM's [HS97] and Transformers [Vas+17] have been proven to achieve compelling results in not just natural language generation but music generation as well. Additionally, the control of such generative models is as important as the generations themselves. While there are studies on controlling the instruments, the tempo, or conditioning the model on a specific artist through different methods, applications of Controllable Text Generation (CTG) have not been explored to same extend in music generation. It is important to investigate whether these techniques can also be used in symbolic music generation in order to achieve greater control over the music that is produced.

In this study, our main objective is both to implement and evaluate controllability on music generation using CTG techniques on the recently developed Music Transformer [Hua+18]. Our proposed approaches implement some of the CTG techniques on the architecture in question and try to condition the generative models on the tonality. In order to be effective, we regard that the controllability features need to successfully condition the generative models on a control code without staggering the generation performance of the original networks. The performance of these techniques along with the replication of the baseline architecture is evaluated on a custom dataset and with quantitative metrics to measure the generations' structural and musical quality and effects of the adaptation of CTG techniques to the music generation domain as well as further evaluation of the baseline network on the downstream classification tasks.

Ultimately, our initial results showed promise in terms of music generation. Building on top of that, our work has contributed to the advancement and adaptation of CTG techniques on music generation by also highlighting the importance of more musical aspects such as musical encoding and evaluation as well as music theory in music generation tasks. From our evaluations, we infer various results on controlling tonality and applications of CTG on music generation. Accordingly, we identify several areas for future research, including possible future approaches to symbolic music encoding, music generation metrics and applications of CTG on symbolic music generation.

Contents

1 Introduction

Music has always been an important part of human life. It helps us to express our emotions and ideas, used commonly in communication and entertainment alike. With the recent advancements on technology writing musical pieces is much more common than ever and it is used more commonly than ever as well. However, writing such pieces still provides a huge challenge and still is a daunting task even with these advancements. The pieces written usually are limited by the artists' skill and creativity and we see that specific artists develop specific styles and artists often deal with writer's block[Wik23k]. Of course, since music is an art-form, the artists do not necessarily rely on software tools to assist their creativity.

Nevertheless, with the recent development on deep learning and later Large Language Models (LLMs) such as ChatGPT[Ope21] there has been a growing interest in generative models and their capabilities. It has been seen that such models can provide exceptionally good results in text generation tasks and often it can be seen that the text is often indistinguishable from a well-written human written text. Such Language Models (LMs) are also used for creative tasks such as poem-generation or story generation and require minimal input from users. We also see such generative models' application in music generation, the generation results in a musical piece rather than a text. There are multiple approaches to music generation from different perspectives as music can be represented in multiple forms: Symbolic and waveform.

Accordingly, we can classify music generation under two categories: Symbolic music generation and waveform music generation. Symbolic music generation is in summary generating the musical scores or the note sheets of the musical pieces, whereas waveform music generation deals with directly generating audio itself. Symbolic generation in particular follows the same principles with text generation: The model is tasked with predicting the next token depending on the previous tokens. On the other hand we see waveform generation not only on music generation, but other tasks such as audio synthesis. In comparison, even though waveform generation outputs better sounding results, it is more computationally heavy and more complex with respect to symbolic music generation. One of the main differences is that in the respective data for audio generation, the interments are not specifically separated. All of the instruments in a musical piece are represented in a continuous form. However, for symbolic music generation the different instruments are separated into multiple tracks or scores, while this opens possibility to other solutions, it also adds a new dimension to the problem.

That being said, the generation task itself is very open to controllability. While the baseline LMs sample from a probability distribution that is only conditioned on previous part of the text or the input, Class-Conditional Language Models (CC-LMs) sample from a distribution that is also conditioned on a specific class. Accordingly, we also see much work done in the field of Controllable Text Generation(CTG). The research done aims on controlling the output of such generative language models towards safer for usage, more exciting, or simply more controllable towards a specific topic or sentiment. As in text generation, controllability is also a challenging task in music generation as well. While the tasks such as text-to-audio is controllable by nature, as the ground truth for the datasets of these tasks have in detail description of the musical pieces, the application of controllability in music remain partially undiscovered. The study conducted on this topic does not concentrate on regulating or controlling the generation process. More precisely, the area of controllable music generation is not extensively investigated. There are some studies done of course, for instance [Zhe+21] aims to control the emotions of the music generation via a pitch histogram [TEC03], where such a histogram gives a probability distribution for the scale of the generation. However, this pitch histogram does not exactly match the CTG techniques we discuss in Section 3.2.

Motivated by such work, we aimed to make music generation more controllable in terms of tonality[Wik23j] using such CTG techniques for monophonic[Wik23f] piano pieces. By successfully developing a controllable music generation model, we can empower artists and enthusiasts with a powerful tool that can assist them in writing musical pieces. Such a model, depending on its complexity, can assist with different aspects of music such as it's melody, tempo, rhythm or tonality allowing them to get inspired and tailor their music to their preferences.

1.1 Objectives

Before outlining the goals of the thesis, we note certain gaps that exist in the literature review:

- CTG techniques are usually applied to natural languages. However, their application on other language-like data is not available. Therefore, we focus on another language-like data: music. While music has similarities with natural languages, for instance a note's sentiment is dependent on the previous notes, music is much more repetitive by nature. In other words it means that we cannot punish a model on repeating the same sequence as it is done in text generation, as in text the essential objective is information passing so repetitiveness is not considered a good practice. However, music is still considered a language-like sequence because of its other characteristics.
- While the generative models for text-to-audio tasks apply controllability through the input prompts, controllability with under other categories of CTG remain undiscovered especially in other music generation tasks. Therefore, one of our objectives is to apply other CTG techniques on symbolic music generation task.
- Tonality (explained in Section 2.1) remains one of the concepts that remain not controlled in music generation. Often the controllability in music generation aims to condition

the models with another melody, artist, genre or instruments. Accordingly, we aim to control the tonality in music generation by conditioning the model on a tonality as much as possible.

1.2 Contributions

Based on such objectives, we provide the following contributions in this study:

- We replicated the work done in [Hua+18] and provide a working code base and weights based on this work.
- We combined numerous piano MIDI datasets into a single dataset that can be used in future work.
- We adapted multiple CTG techniques into music generation field and conducted experiments to compare their performances.
- We explored a novel approach on controllability of tonality in generative music models.
- We studied the general music understanding of [Hua+18] via training the networks on downstream tasks.

1.3 Structure

The thesis structure is as follows:

Background

In the Chapter 2, we provide relevant background information on the key concept of this study, such as about music theory, transformers, and language models (LMs).

Tasks and Related Work

Chapter 3 includes the information about the tasks that we worked on in this study as well as their related works. We also talk about our selection of controllable text generation methods and music generation models. Lastly, we talk about applying such generative networks to downstream tasks.

Methodology

In Chapter 4, background knowledge about music theory and transformer architectures as well as the model architecture used for music generation and different data embeddings are represented. Later, how the CTG techniques used for controllable music generation are discussed. And finally, the quantitative metrics used to evaluate the models and the tasks are presented.

Evaluation and Discussion

Later on, we continue the study with Chapter 5, where we discuss the datasets for both generation and downstream tasks, and the evaluation of the tasks. Additionally, we present information about how the training is done with some metrics, and lastly we discuss the results of the experiments.

Future Work

Chapter 6 talks about future work of this study. We explore in what areas the problems possibly lie and how we can solve such problems.

Conclusion

In the last chapter, Chapter 7 we conclude the work done and provide a quick summary of the thesis.

2 Background

In this chapter, we provide relevant background information about music theory, transformers, and language modelling. We discuss the what is tonality and key in music theory, the concept of closely related keys and how does one can figure out closely related keys of a given key in Section 2.1. In Section 2.2 we provide in depth information about transformer architecture, the attention mechanism, and the concept of positional embeddings. Lastly, the background for language models is discussed in Section 2.3. We provide information about how language models are used to generate in an auto-regressive manner, how they are used for downstream tasks, and the concept of class-conditional language models.

2.1 Music Theory Background

Tonality in music theory means structuring the piece around a certain pitch or tonic. The tonic serves as a resting point throughout the piece. The root note of the tonic chord forms the name given to the tonality or the key. For instance, in key of A minor the root of the the tonic chord, A, is given as the name.

Keys in music theory is a way of labeling the tonality. In other words, each key corresponds to a specific tonic. A piece that is written in a key is said to be using the scales and notes that belong to that key. For instance, the key of C major is based on the note C as the tonic and uses a scale consisting of the notes C, D, E, F, G, A, and B. Accordingly, all of the 12 notes in western music can be used as a tonic and the relationship between the tonic and the other notes in the scale creates a unique tonal flavour or mood, that is different in each key and a characteristic feature of such key.

As each tonic and the notes used create a unique tonality, different keys that are used through the piece with modulation [Wik23e], changing keys, can also create unique tonality. However, it is important to understand that using a key that is not related to the tonic is not a common practice. Often, the key is switched to one of the closely related keys [Wik23b] when modulation is used. It is also important to understand that between closely related keys, there is a high correlation between the notes (Almost all of them), however, the usage of the notes differ from key to key.

A common way to identify closely related keys is by using the Circle of Fifths [Wik23a], which can be seen in Figure 2.2. Through the Circle of Fifths, one can see the closely related keys by looking at the adjacent keys in the figure, where the artist can choose from major

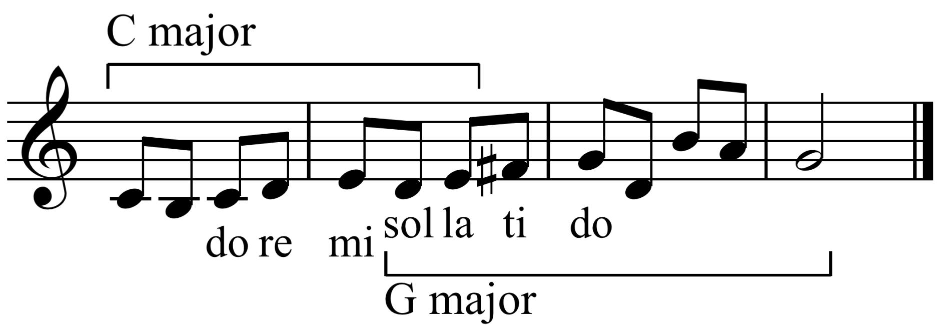


Figure 2.1: Example of modulation in a note sheet. Taken from [Wik23e]

keys [Hut21a] and minor keys [Hut21b]. Of course, since music is a creative art form, the artists are not forced to obey these rules; however, it is common practise.

One other concept that is important and relevant to this study in music theory is harmony [Wik23c]. Harmony in simple terms refers to the combination of notes, chords, or other musical elements that are played simultaneously, creating a pleasing or meaningful sound. In other words, harmony can be described as "intact" when the chords and notes used in a piece are in agreement with the underlying tonality and musical structure of the piece. Harmony is a fundamental concept that affects the quality of the piece heavily. A piece that doesn't have intact harmony usually doesn't sound pleasing and structurally incorrect.

However, one should note that the notes played in a song or a piece do not necessarily need to be exactly in the key of the song. One of the common concepts that is used in composing is borrowed chords [Wik23h]. Borrowed chords are, in simple terms, chords that actually belong to another tonality or key, but usage of such chords and the notes that belong to these chords do not break the current tonality of the song.

2.2 Transformers

Transformers, introduced by [Vas+17], revolutionized the way LM's work with their exceptional performance. In terms of long term understanding they outperform older models such as LSTM's [HS97] due to avoiding problems such as vanishing gradients[1]. Accordingly, the model used in this work is based on Transformer architecture.

2.2.1 Positional Embeddings

The way the Transformers handle positional information is via the position embeddings. Unlike older LM's, Transformers do not try to understand the token order by sequentially processing the tokens in the sequence. Instead, the tokens are summed with an embedding

[1]https://en.wikipedia.org/wiki/Vanishing_gradient_problem

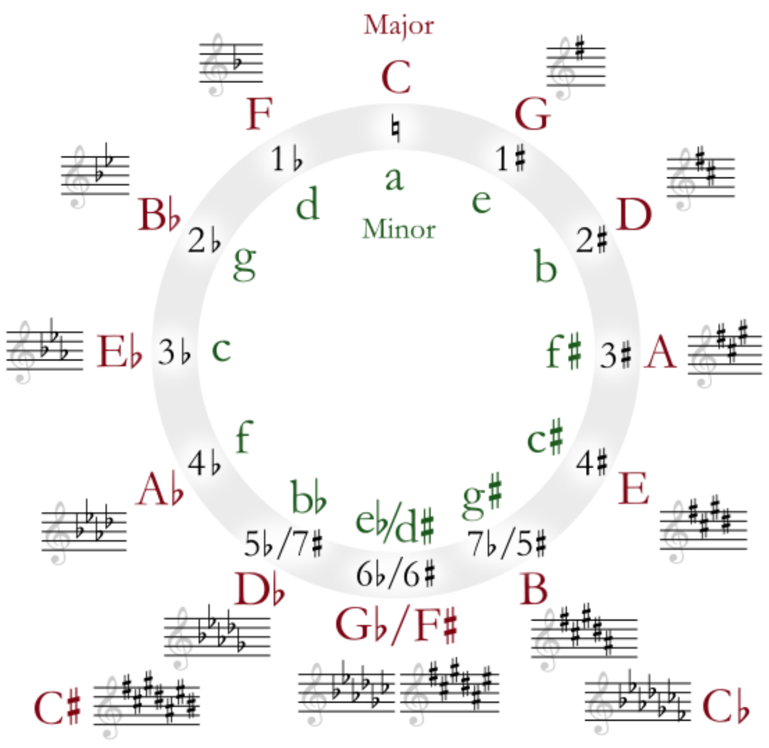


Figure 2.2: Circle of fifths showing major and minor keys as well as their key signatures in the note sheet. The figure also shows the number of flat and sharp notes for every key. In total there are 24 major and minor keys that we can control and label. Taken from [Wik23a]

that represents their position in the sequence in order to specify their location inside the sequence. The positional embeddings can be learned during training, however, as pointed out in [Vas+17], the results are nearly identical to static Positional Embeddings.

Positional Embeddings derived from the following equation:

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(pos/10000^{2i/d_{\text{model}}}\right) \\ PE_{(pos,2i+1)} &= \cos\left(pos/10000^{2i/d_{\text{model}}}\right) \end{aligned} \tag{2.1}$$

In the end the final input embeddings are the summation of token embeddings and positional embeddings:

$$E_{x_i} = x_i + PE_{x_i} \tag{2.2}$$

2.2.2 Self-Attention Mechanism

The main improvement of Transformers is the usage of Self-Attention mechanism. Attention, originally introduced by [LPM15], which is also called Scaled-Dot-Product Attention, is derived by the following equation:

$$Z = \text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.3}$$

, where the input consists of Queries (Q) and Keys (K) with dimension d_k and values (V) with dimension d_v. However, the difference in [Vas+17] is that instead of using a single head attention, projecting the queries, keys, and values linearly h times using distinct linear projections that are learned, with dimensions of d_k, d_k, and d_v, respectively. Later, the Attention function is applied in parallel, and at the end each head concatenated, resulting in outputs with d_v dimensions. The Multi-Head Attention equations is as follows:

$$Z^h = \text{Attention}\left(Q^h, K^h, V^h\right) = \text{Softmax}\left(\frac{Q^h K^{h\top}}{\sqrt{D_h}}\right)V^h \tag{2.4}$$

In this context, the queries, keys, and values are represented by $Q = XW^Q$, $K = XW^K$, and $V = XW^V$, respectively. The input sequence X is transformed into l vectors with D dimensions, denoted as $X = \{x_1, x_2, ..., x_l\}$. The matrices W^Q, W^K, and W^V are square matrices with dimensions of DxD, where $D = d_k = d_v$. Each lxD query, key, and value vector is then split into H separate vectors with dimensions of lxD_h, where D_h is calculated as $D_h = \frac{D}{H}$ and indexed by h.

Such application of Attention, is especially beneficial in sequences. Since, this way the models is able to attend to multiple positions in the sequence instead of only one token.

The Multi-Head Attention can be made to attend to a different sequence in tasks such as translation, however, in this work the tasks, described in Chapter 3, do not require on conditioning on another sequence. Instead the mechanism called Self-Attention, where the attention is not on another sequence, on the original input sequence.

2.2.3 Transformer Blocks

Transformer blocks are the main building blocks of a decoder-only Transformer[2]. As it can be seen in Figure 2.4, Transformer blocks are made of a Self-Attention layer and a Linear (Feed Forward) layer, each of them are followed by a normalisation layer.

To elaborate further, the normalization layer ensures that the values in the hidden layers remain within a range of unit Gaussian. This helps to stabilise the model during training. Furthermore, the residual connections enable the lower-level layers to access the gradients of

[2]While there is an encoder-decoder architecture available, this work primarily concentrates on the auto-regressive decoder architecture. The reason being, [Hua+18] has also utilized the same architecture.

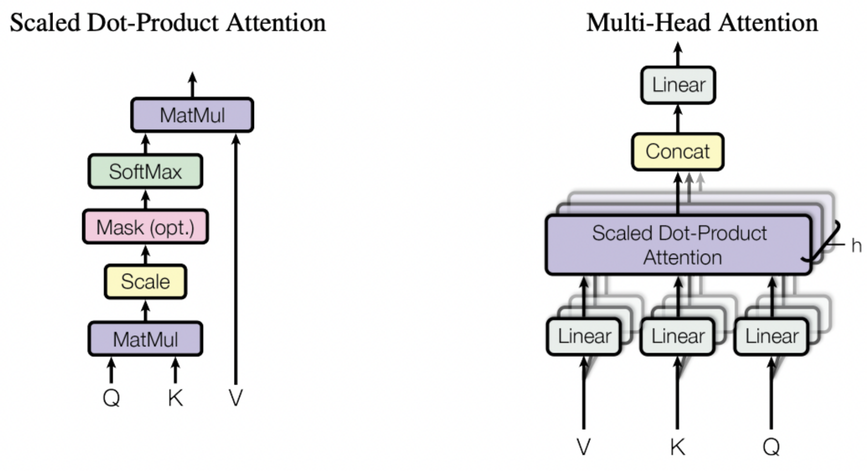


Figure 2.3: Scaled-Dot-Attention and Multi-Head Attention. Taken from [LPM15]

the higher-level layers. This specific architecture has been implemented and utilized in this work.

Accordingly, to get the output probabilities for the next token, the output of N Transformer Blocks o_t is passed to Softmax.

$$p\left(x_t, x_{:t-1}\right) = \text{Softmax}\left(o_t\right) \tag{2.5}$$

2.3 Language Models

The basic idea behind a LM is to use probability to predict the likelihood of a particular word or sequence of words appearing in a given context. Accordingly, to predict the likelihood, the models have to learn inter-token relationships in the training data. Given an input sequence $X = \{x_1, x_2, x_3, ...x_l\}$, the purpose of LM is to learn $p(X)$. An autoregressive model with parameters θ assigns a probability to the sequence X by factorising it with the chain rule:

$$p_\theta(X) = \prod_{i=1}^{n} p_\theta\left(x_i \mid x_{<i}\right) \tag{2.6}$$

the model in question can later assign probabilities to sequences by iteratively predicting the token x_i given the tokens $x_{<i}$.

2.3.1 Language Models for Downstream Tasks

Downstream tasks refer to a range of NLP applications that use language models as a basis for solving specific problems, such as text classification, named entity recognition, machine translation, and others. Given a PLM, it can be fine-tuned to a downstream task and can

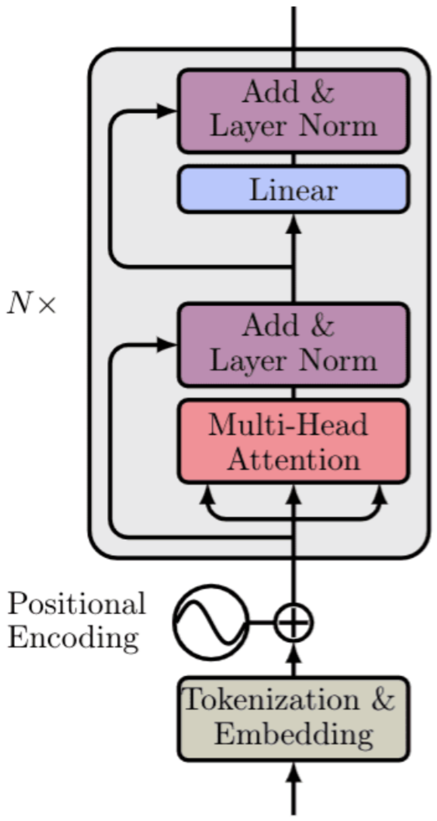


Figure 2.4: Transformer Block with Self-Attention (Multi-Head Attention) and a Feed Forward layer, each layer followed by layer normalization. Taken from [Sum19]

achieve good performance depending on the models shape and size and the tasks themselves [Tay+21] via leveraging the relationships between token that are learned already by the LM. This fine-tuning involves updating the weights of the model for the specific task as well as making some architectural changes.

Specifically, a LM can be used for a downstream task with changing its head. In an Auto-regressive generative model such as the Music Transformer, the head originally tasked with a sequence-to-sequence task: Predicting the next token depending on the previous tokens by assigning a probability to the sequences. However, the head can be modified to predict a class instead of next token. Specifically, in this concept, the number of classes for the prediction change to the number of classes from the original vocabulary. So, instead of assigning probabilities to the possible sequences, the model assigns probabilities to the class-sequence pairs.

2.3.2 Class-Conditional Language Models

Class-conditional Language Models (CC-LM) are LM's that are conditioned on a control code c in order to generate sequences conditioned towards a specific class. The probability distribution of the sequence still factorized via the chain rule as follows:

$$p_\theta(x \mid c) = \prod_{i=1}^{n} p_\theta\left(x_i \mid x_{<i}, c\right) \tag{2.7}$$

The control code can be an attribute or another sequence as well. For example, a sentiment-conditioned language model would be trained on a corpus of text that has been labeled as either positive or negative in sentiment. The model would then be able to generate text that is specifically tailored to a given sentiment, such as positive reviews or negative comments.

CC-LM's learn the probability distribution $p_\theta\left(x_i \mid x_{<i}, c\right)$ by various techniques ranging from direct concatenation to modifying and re-training of the LM. The performance and the use cases of such approaches differ according to the technique.

2.3.3 Generation

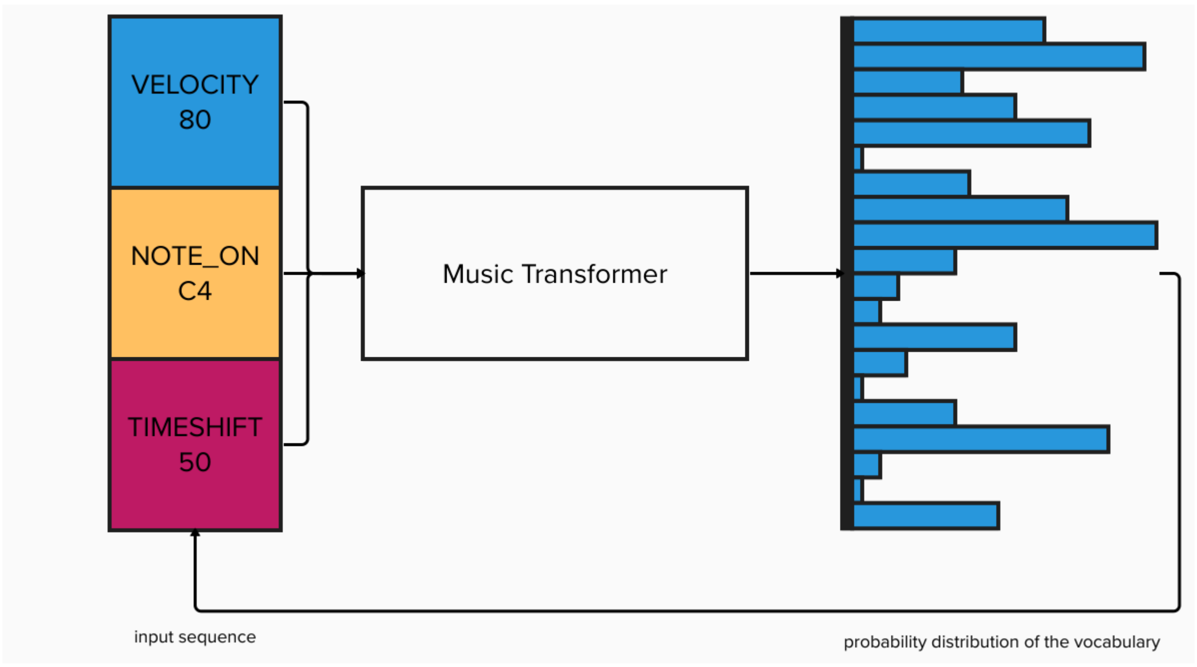


Figure 2.5: Auto-regressive generation with Music Transformer: The selected token from the output distribution is concatenated to the input prompt and later the updated input is fed to the network for further generation until the desired limit reached or the network decides to end the sequence.

With an Auto-regressive model, the generation of a new sequence is done iteratively. It is required to sample from $p_\theta\left(x_i \mid x_{<i}\right)$ for the next token. Accordingly, one of the tokens with higher probabilities will most likely be chosen as the next token. Later, concatenating the sampled token x_i with $x_{<i}$ and feeding the new sequence to the model. In this work, the tokens are not limited by notes, but also have the velocity and timeshift events as well. The model is tasked to generate the tokens from 4 different classes.

Temperature Setting

In the context of generating text, or a sequence with a LM, temperature refers to a hyper-parameter that determines the randomness or creativity of the generated sequence. The temperature of a LM controls how much it deviates from the most likely next word or sequence of words (in our case notes), as determined by the model's training data and internal probabilities.

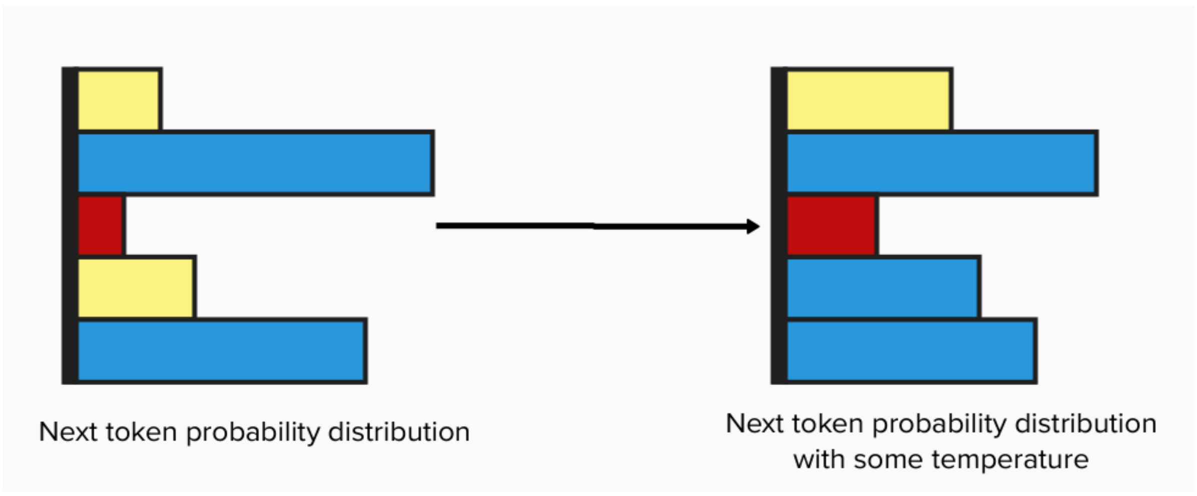

Next token probability distribution
Next token probability distribution with some temperature

Figure 2.6: Temperature setting illustration.

A higher temperature will cause the model to generate more diverse and creative songs, as it will be more willing to choose notes that are less likely according to its internal probabilities. A lower temperature, on the other hand, will cause the model to generate more predictable and conservative songs, as it will stick more closely to the most likely next words. The temperature setting is applied on the logits. The formula is basically as follows:

$$z = z/t + e^{-9} \tag{2.8}$$

where, z is the logit and t is the temperature. The temperature t is summed with e^{-9} in case of 0 temperature. Accordingly, when used with softmax:

$$\sigma\left(z_i\right) = \frac{e^{\frac{z_i}{t+e^{-9}}}}{\sum_{j=0}^{N} e^{\frac{z_j}{t+e^{-9}}}} \tag{2.9}$$

As it can be seen, low temperature makes the more probable notes even more probable while high temperature makes them less probable.

3 Tasks and Related Work

This chapter of the thesis outlines key concepts and the tasks in this study and their related works in detail. We start by explaining different music generation approaches in Section 3.1. Specifically, we divide music generation approaches in two: Symbolic and Waveform and then we discuss which approach we choose. Afterwards, we continue with providing information about controllable generation task. We discuss controllable text generation(CTG) in detail with previous approaches and point out the techniques we tried in this study in Section 3.2. Finally, the chapter ends with Section 3.4, where the discussion of application of the generative model to downstream tasks and why we chose to apply it on a downstream task.

3.1 Music Generation

Automatic music generation has been around for more than 60 years. While the first works do not use the deep learning methods today, they can still be considered automatic music generation. During this long period various methods were used, including defining new grammars, probability models, and deep learning models.

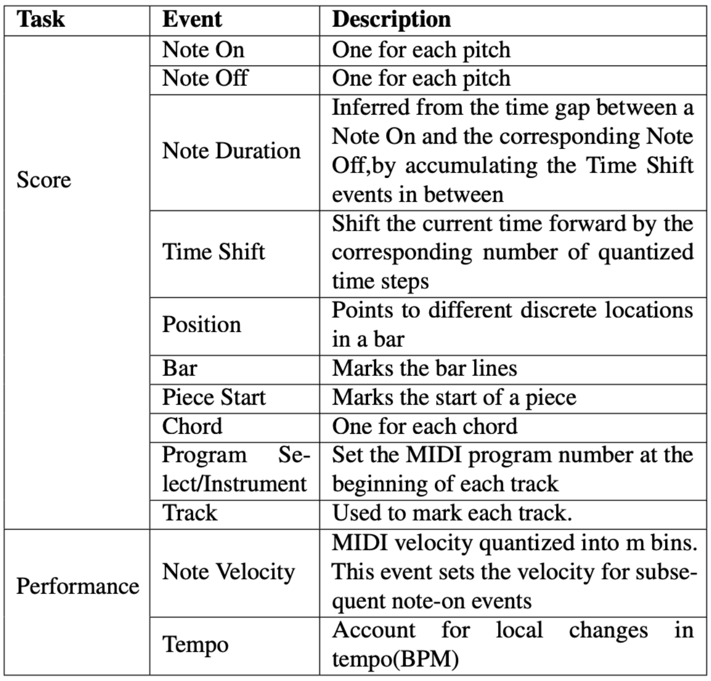
In general, we divide music generation into two:

- Symbolic Generation: The musical data is represented in a discrete language-like structure.
- Waveform Generation: 1D or 2D audio signals are used to represent musical data.

3.1.1 Symbolic Music Generation

Firstly, symbolic music representation is using symbolic music blocks such as musical notation to represent music. This kind of representation uses readable musical symbols such as notes and time signatures to represent the musical information of the piece, such as rhythm, harmony or the melody in a discrete way. The advantage of this approach lies in its precision and detail of the musical representation. However, the disadvantage is the availability of the correct symbolic musical notation for deep learning training.

Accordingly, a way to generate music is generating the note score, or a Musical instrument digital interface(MIDI) format [Wik23d]. Generating, music in such format is called symbolic music generation. These methods often thread the symbolic representation as a language and train sequence models to generate musical pieces. The difference is that in the text



Task	Event	Description
Score	Note On	One for each pitch
	Note Off	One for each pitch
	Note Duration	Inferred from the time gap between a Note On and the corresponding Note Off,by accumulating the Time Shift events in between
	Time Shift	Shift the current time forward by the corresponding number of quantized time steps
	Position	Points to different discrete locations in a bar
	Bar	Marks the bar lines
	Piece Start	Marks the start of a piece
	Chord	One for each chord
	Program Select/Instrument	Set the MIDI program number at the beginning of each track
	Track	Used to mark each track.
Performance	Note Velocity	MIDI velocity quantized into m bins. This event sets the velocity for subsequent note-on events
	Tempo	Account for local changes in tempo(BPM)

Figure 3.1: Possible events in a midi file. It is divided between score and performance, where the events related to the score are directly related to the note sequences and the performance related events are more related to other dimensions of the notes such as the overall speed of the tracks or note velocities. Taken from [JLY20]

the main building blocks are words and sentences, while in the symbolic music they are notes. However, the notes also hold information such as it's duration and velocity, which also can easily affect their sentiment. Similarly, their sentiment is heavily dependent on the neighbouring notes. However, one main difference that music has with respect to a natural language is the information flow. With any kind of text generation task the main objective revolves around information, either seeking or giving. To that extend, in text generation repetitiveness is not desirable, while in music the melody or rhythm can easily be repeated to some extend depending on the genre. For instance, when we repeat same sentences in a story it makes the story boring, however, when we repeat the same drum rhythm in a song that repetitiveness doesn't necessarily make the song boring.

The MIDI format can be encoded in different ways; 1D, 2D:

- **1D**: One way of encoding a MIDI file can be as close to the original MIDI format as possible [Oor+18a], where the MIDI format can be seen in Figure 3.1. Another approach is to simplify the MIDI like representation in order to make learning easier for sequence models [HY20]. Specifically, the main difference is that they use note duration instead of note on and note of events, it is more bar based instead of absolute time.
- **2D**: Another music representation method draws the notes played over time as a 2D matrix by sampling time. However, increasing the dimension of the representation increases the complexity respectively. Since these methods based on sampling from time, in a poly-rhythm or complex rhythm cases the sampling will need to be more precise, and accordingly the time dimension of the representation will be bigger. One of the examples that uses this representation is [MSC18]. They also include velocity information in the matrix as well for representing.

Symbolic music generation task with deep learning saw one of the first use cases with RNN's with [Tod89]. The task itself is considered as a variation of text generation. The goal of the generation is from a symbolic music input the rest of the sheet to be generated with a neural network. As another example, Performance RNN [SO17] tries to generate polyphonic music in form of MIDI with LSTM's. Building on that Music Transformer [Hua+18] tries to solve the same objective with a transformer based network and a relative position-based attention mechanism in order to capture long-term attention easier. Additionally, they introduce a memory efficient version of such mechanism and show it's performance on symbolic music generation. Whereas, Museformer [Yu+22] uses a different attention mechanism, so called Fine- and Coarse-Grained Attention in order to capture long term attention even better. There are also other architectures such as MusicBERT [Zen+21], where suggest a bar-level encoding of the MIDI files and bar-level masking in in BERT training. MusicBERT is able to generate latent MIDI encodings for music understanding tasks.

In symbolic music generation, when worked with polyphonic music, other instruments also come in to the vision as separate tracks. The models should keep track of multiple sequences at the same time and should keep track of their relationships. since, the sequences from different instruments should follow the same tonality and they need to harmonize with each other. However, when worked with monophonic music, the single instrument should only harmonize with itself. For instance, the melodic notes need to follow the bass tonality, but they are present in the same sequence.

3.1.2 Waveform Music Generation

On the other hand, we can also represent the musical pieces in waveform. With contrast to symbolic representation, waveform representation is continuous, which has much more detail in acoustic information such as timbre, texture, and the effects. Accordingly, these methods often involve objectives such as generating the audio with instruments, noise, effects, lyrics etc. The disadvantage, however, would lie in it's complexity, since multiple instruments playing at the same time is not even easily done by human ears. There are much research done in identifying the instruments in a piece under the term of Musical Information Retrieval(MIR), for instance one such use case is converting audio to MIDI data [Bit+22]. However, this complexity and continuity makes it harder to control the output of a potential generative model.

Waveform representation can be divided into two types of representations:

- **1D Representation**: Time information is held in the horizontal axis and the vertical axis holds the information about the signals. This can be considered as a direct approach to represent audio itself, as it can be easily converted into sound.

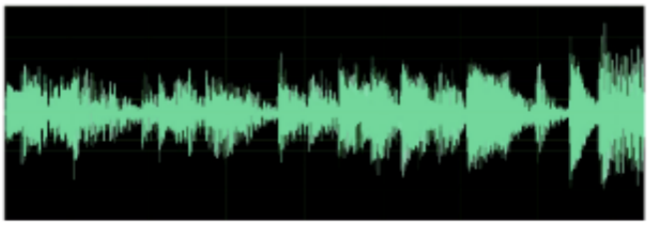


Figure 3.2: 1D representation, taken from [JLY20]

- **2D Representation**: To obtain frequency data, the audio is converted into the frequency domain through the use of Fourier transforms on windows that overlap, and this data is displayed in relation to time, frequency, and amplitude.

As one of the first breakthroughs WaveNET [Oor+16] expanded the horizon in the synthesis task. It was used in tasks such as text-to-speech and music generation. It utilized convolution layers to generate waveforms. Later, SynthNET [**synthnet**] expanded upon WaveNET, able to learn direct relations between audio and notes. They focused on a MIDI-to-Audio task. They present an encoder-decoder architecture with LSTM's, the encoder learn to create a latent

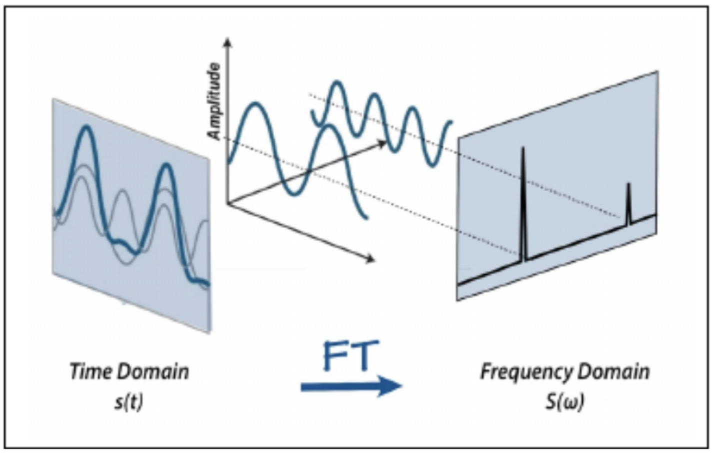


Figure 3.3: 2D representation, taken from [MBM21]

space between MIDI notes and audio while the decoder is able to generate audio from such latent representations.

Since it's release GAN [Goo+14] achieved success in generating images in high quality. We, of course, saw its application on music generation as well. WaveGAN [DMP18] can be considered one of the first applications of GAN usage in music generation. They worked on a unsupervised learning task. Its generative network generates waveform in the time domain, while the discriminator learned to distinguish between real and generated waveform data. Later, we also see modeling with time-frequency information

3.2 Controllable Generation with Pre-trained Language Models

Controllable Text Generation (CTG) is the task of controlling the generated text with respect to the users' preferences by either taking an additional control prompt or including the preference in the input prompt. Depending on the task, the generative model can be controlled in various ways. For instance, in the case of safe text generation, the generative model can be guided in the direction of less offensive text, if the users' preference is in the direction of a topic or a sentiment it can be also controlled. Accordingly, controlling the generative models that are tasked with music generation is also possible. In tasks such as text-to-waveform generation, the model can be constrained with instruments, genres or tempo.

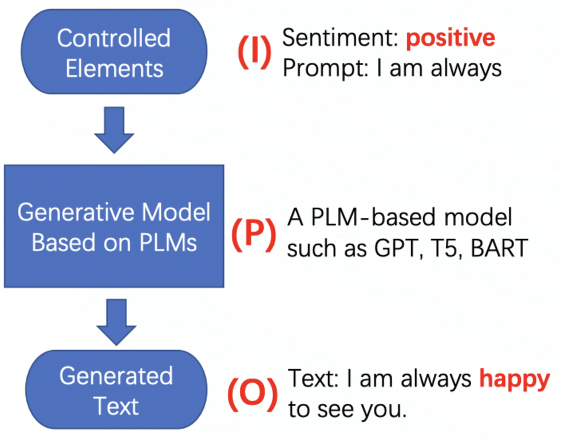

Controlled Elements
(I)
Sentiment: positive
Prompt: I am always
Generative Model Based on PLMs
(P)
A PLM-based model such as GPT, T5, BART
Generated Text
(O)
Text: I am always happy to see you.

Figure 3.4: The IPO illustration of controlled text generation. A typical CTG system consist of three components: the controlled element alongside the input sequence (I), the generative model (pre-trained language model) as the process (P), and the generated text satisfying the input control condition as output (O). Taken from [Zha+22]

Accordingly, Controllable Music Generation(CMG), is the task of controlling the music piece generated by additional constraints. In this concept, we try to utilize CTG methods in order to control the generated musical piece's more abstract features (See Section 3.3) by passing an additional control prompt to the generative model.

To this extend, CTG techniques can be classified in 3 main branches:

- **Fine-tuning Techniques**
- **Retraining and Refactoring Techniques**
- **Post-processing Techniques**

3.2.1 Fine-tuning Techniques

As a straightforward way, fine-tuning adjusts part or all of the LM for a specific task. In terms of flexibility the model needs to be fine-tuned specifically for the sentiment or the topic. Such down-streaming of a general language model for a specific task can result in better quality at a low cost. Nonetheless, because these methods involve adjusting the models to suit particular content, the level of control is sub-optimal. Fine-tuning techniques can be expanded into more advanced methods as such:

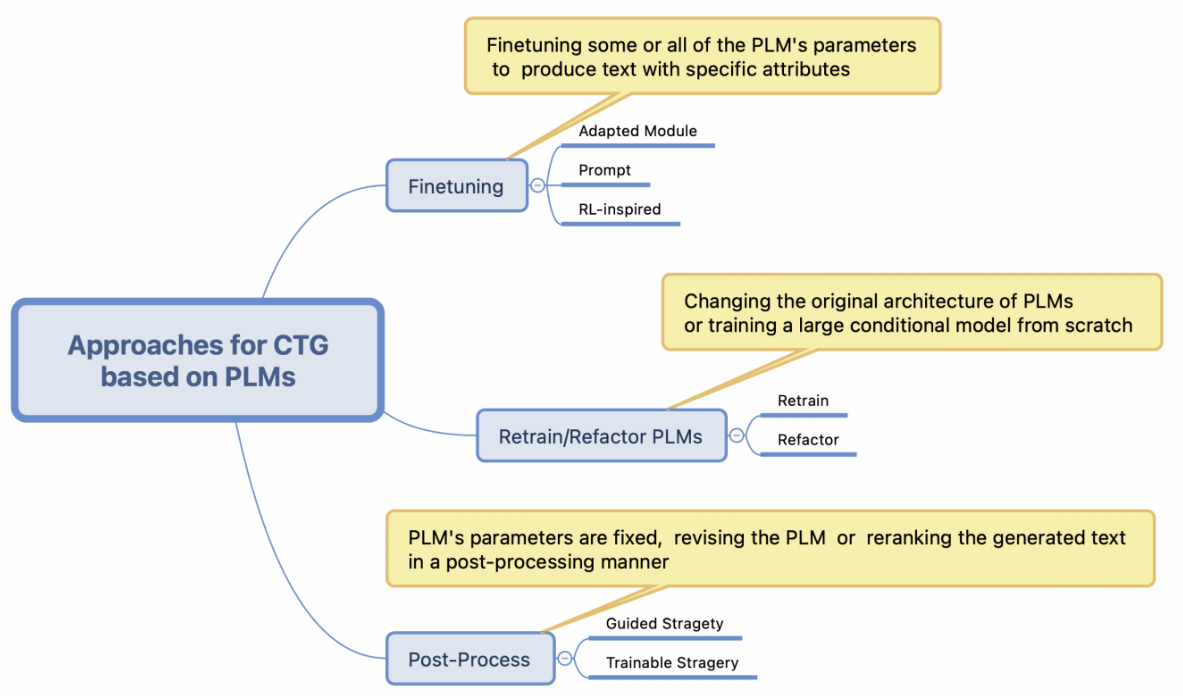

Approaches for CTG based on PLMs
Finetuning
Finetuning some or all of the PLM's parameters to produce text with specific attributes
Adapted Module
Prompt
RL-inspired
Retrain/Refactor PLMs
Changing the original architecture of PLMs or training a large conditional model from scratch
Retrain
Refactor
Post-Process
PLM's parameters are fixed, revising the PLM or reranking the generated text in a post-processing manner
Guided Stragety
Trainable Stragery

Figure 3.5: Categorization of CTG techniques under 3 branches: Fine-tuning, Retraining/Refactoring, Post-processing techniques. Taken from [Zha+22]

Adapted Module

This method aims to construct an adapted module for a specific context around a PLM and then it is trained with the PLM similar to direct fine-tuning. One of the latest examples, namely DialGPT [Zha+19], implements an adapted module around the GPT [Rad+18].

Prompt Based

One of the ways is to having the objective of the fine-tuning consistent with the original training [GFC21]. The main work done in this approach is including a set of templates in the input prompts in the training data. For instance, with templates like: "No reason to watch. **It was [MASK]**". Accordingly, the training objective is to predict the [MASK].

Reinforcement Learning

At the same time other fine-tuning methods leverage Reinforcement Learning(RL). The motivation with these approaches is to reward the model depending on how well the control code objectives are achieved. For instance, [Zie+19] uses rewards from human preferences, while [Tam+19] introduces a new reward method that rewards the model at intermediate time steps for story generation.

3.2.2 Retraining and Refactoring techniques

Another approach for CTG is retraining or refactoring the model for better controllability. Such methods often require much more development and training time and computational resources than other techniques as run additional training on the original PLM.

As an early attempt CTRL [Kes+19] suggests concatenating the control code at the start of the input text without any modifications on the PLM. Training with this approach lets the model learn the conditioning on various control codes. Additionally, they also introduce top-k sampling for text generation:

$$p_i = \frac{\exp\left(x_i/(T \cdot I(i \in g)\right)}{\sum_j \exp\left(x_j/(T \cdot I(j \in g)\right)} I(c) = \theta \text{ if c is True else 1,} \tag{3.1}$$

where, g is the generated list of tokens, p_i is the probability of the next token, and $I(c)$ gives a repeat penalty.

Unlike it's predecessor CoCon [Cha+20] suggests injecting a control block inside the PLM in order to have more precise control over the topic or the condition. In the work, they inject a CoCon Block inside a pre-trained GPT-2 model and train the new model. Additionally, it is suggested in the paper training a LM with additional losses such as; Self Reconstruction Loss, Null Content Loss, Cycle Reconstruction Loss, and Adversarial Loss can be beneficial to train a Class-Conditional Language Model(CC-LM).

3.2.3 Post-processing Techniques

As the number of parameters increase with the number of content to memorize for a LM, the computational complexity both during training and inference increase. To tackle this problem certain methods propose achieving controllability with influencing the outputs of a smaller LM.

For instance, Plug&Play [Dat+19] provides a simple approach to controllable language models. In their approach they use a simple discriminator, such as a bag of words, in order to guide the output of a PLM. Whereas, GEDI [Kra+20] builds on top of CTRL [Kes+19]. They assume the existence of a PLM as well as a much smaller CC-LM and basically use the CC-LM to guide the PLM's output in the desired direction by using simply Bayes' theorem. Similarly, Plug&Blend [LR21] works with a similar approach. Instead of just a CC-LM guiding the PLM, they also implement a planner module in order to control the generated text more in detail. Later, Keyword2Text[Pas+21] suggests a non-discriminator approach. Specifically, the control code is considered as a hard constraint by the model and in the score function the shift towards the wanted vocabulary added.

3.3 Chosen Approach

One of the objectives of this study is to apply CTG methods to other language-like data and in this specific case music as mentioned in Section 1.1. To that extend we have the following limitations while selecting the architecture:

- As the CTG methods are mostly applied to transformer based architectures, the model we need to choose should be also based on transformers. We do not want to face additional challenges and limitations on CTG technique selection.
- To reduce the complexity we also constraint the tasks with monophonic music generation, where we generate music with only one instrument.
- We also wish to apply multiple CTG techniques. Accordingly, the input prompts should be either text or other language-like structure.
- Audio generation is computationally expensive [JLY20]. Given the time constraints we wish to work on symbolic representations.

Given such challenges and limitations, we chose Music Transformer [Hua+18], since it meets the requirements for this study.

With the CTG methods, on the other hand, as mentioned before in Section 1.1, we aimed to apply multiple techniques in order to compare their performances. Given the comparison between CTG techniques in Table 3.1, we chose to implement CTRL [Kes+19] and CoCon [Cha+20] for our case. We chose these techniques as they can be both classified as different types of techniques, retraining and refactoring, and also should result in better controllability and quality.

Method	Main characteristics
Fine-tuning	standard training; efficient inference; higher text quality; weaker controllability
Refact/Retrain	computationally expensive training; higher text quality; better controllability
Post-Process	efficient training; computationally expensive inference; lower text quality; better controllability

Table 3.1: Comparison of CTG Techniques. Taken from [Zha+22]

Chosen the CTG Techniques to apply, which feature of the symbolic generations to condition the model on is another challenging task. We explored numerous aspects to control via such techniques, but we were able to work only on one of them:

- **Chord Progression**: One possible option is to control the chord progressions of the songs. To control the chord progression use would need to give a sequence of chords, however, extracting chord progressions reliably is a challenge and since it is another sequence it might make learning harder.
- **Key/Tonality**: One other possible option is to control the tonality or the key of the songs. Key of the song actually defines the possible chord progressions that can be used within the song and it is easier to label MIDI data with a key. One big challenge with tonality control is modulation, key change during the song.
- **Genre**: Songs that belong to the same genre are usually written in similar keys and chord progressions. However, genres also include certain play-style which might be hard to learn.
- **Artists**: Every artist has his/her own play-style and preferences. Without sufficient data for each artist the trained network will have heavy bias towards certain artists.

Accordingly, we chose to experiment with controlling the tonality of the songs. As songs can reliably labelled with their keys and do not include additional features such as play-style. We assumed there are no modulations in the songs during our experiments.

Additionally, as describe in Section 5.1, we have at least a couple of hundred songs assigned to each key, which makes sampled sequences diverse from each other. On the other hand, for genre and artists, some of the classes only have a couple of examples, which makes the sampled sequences hugely overlap with each other and force the network to memorize such sequences (See Subsection 5.3.4 for more details).

3.4 Downstream Tasks

One other application of LM's is applying them to downstream tasks by conditioning them to in-context dataset. Such PLM's demonstrate exceptional performance in downstream tasks with fine-tuning [Raf+19]. Similarly, we experimented with using LM's for classification in multiple contexts. The reasoning behind this is that by using a PLM that was training on the music data we can measure it's performance in context classification. Depending on its performance on such a classification task, we can gain insight on how such a model would perform on the actual class-conditional generation. To illustrate, when the model shows poor results in a classification task, we anticipate that its performance in related class-conditional generation task will also be unsatisfactory. This is because the poor performance in the classification task suggests that the current architecture, data, and encoding approach are insufficient in learning relevant context information.

To that extend, we experiment with different classification tasks to measure the models performance in various concepts. Namely:

- **Key Classification**: As the main objective of this study is to control the tonality of the music generated by our models, our first priority is to measure the performance of our model with the key classification. As described in Section 5.1, all of the songs in the dataset are labeled by their keys and divided as described.
- **Genre Classification**: Another experiment we conducted is with the genres. With this task, we measure our models understanding on the genres. The genres also hold different sentiments and play style.
- **Artist Classification**: Lastly, we experimented with the artists themselves. Each artist has an unique play style with different tonality preference, genre preference or sentiment preference. Here we aimed to measure the models understanding on a limited set of artists with respect to some of their musical pieces.

Basically, the approach to the tasks described are in parallel with the method described in Subsection 2.3.1: The head of the baseline LM is changed according to the classification task and the transformer blocks are frozen. Then the modified head later trained on the respective dataset on the classification task.

4 Methodology

In this chapter of this study, we discuss the methodology used in this study. We provide a detailed overview of how musical data represented in Section 4.1 and applied such representation on our data. There, we also explain how the labeling is done for the dataset as well. In Section 4.2 we talk about the model architecture and its' differences with the vanilla transformer in detail. Later, we explain how we can applied the controllable text generation methods to the case of this study. We go over briefly how both of the methods work in text generation and how we implemented them for music generation case in Section 4.3.

4.1 MIDI Representation

MIDI format maps each action in the tracks in an event based system. In [Hua+18] follows the method suggested in [Oor+18b] for encoding the following events:

- **NOTE_ON**: note press
- **NOTE_OFF**: note release
- **TIMESHIFT**: time skip
- **VELOCITY**: note velocity[1]

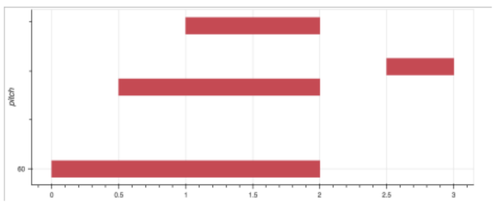
4.1.1 Encoding

Encoding the MIDI data is as follows:

1. Each note in the track is divided into NOTE_ON and NOTE_OFF events
2. Notes are sorted by their absolute time
3. From the NOTE_ON and NOTE_OFF events VELOCITY and TIMESHIFT events are generated:
    - VELOCITY events are created by looking into the current velocity and next events velocity. If the next event is a NOTE_ON it has a velocity and no velocity otherwise. So, when on a NOTE_OFF event the current velocity will be none and since the next event will be NOTE_ON it will have a velocity, by checking this the velocity event is created.

[1]How hard a note key is pressed on piano

- TIMESHIFT events are created by looking at the current time and next note events time. If the next event is a NOTE_ON event and it does not appear directly after the last note it will have a different time than the current time, so a TIMESHIFT event will be created. Otherwise, the algorithm will not create such a event as it is not necessary. But if the next event is a NOTE_OFF event it should have a different time than the previous event (NOTE_ON). So, a TIMESHIFT will be created to represent the duration of the note.



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

Figure 4.1: A small snipped of piano roll serialized. Taken from [Hua+18]

Next, the events are serialized into an integer vector with the following vocabulary:

- 0-127 NOTE_ON events
- 128-255 NOTE_OFF events
- 256-355 TIMESHIFT events
- 356-387 VELOCITY event

Accordingly, this vocabulary allows the model to cover all of the piano board with it's full versatility. Having 100 places reserved in the vocabulary allows the model to represent time shifts from 10ms to 1s.

Unlike natural languages, the tokens in our vocabulary do not contain any previously learned information about their sentiment, relationships with other words, or other structural or sentimental information. In order to implement controllability on tonality we either need to extend the vocabulary or use a existing token, however, the challenge is that our existing tokens do not directly represent keys in any case. Accordingly, for controllable generation task the vocabulary mentioned is extended in order to include the control tokens. In this work all the major and minor keys have been taken into account, totalling in 24 extra indices to the vocabulary. Additionally, for downstream tasks the vocabulary extended further in order to include artists and genres accordingly to their sizes (See Chapter 3).

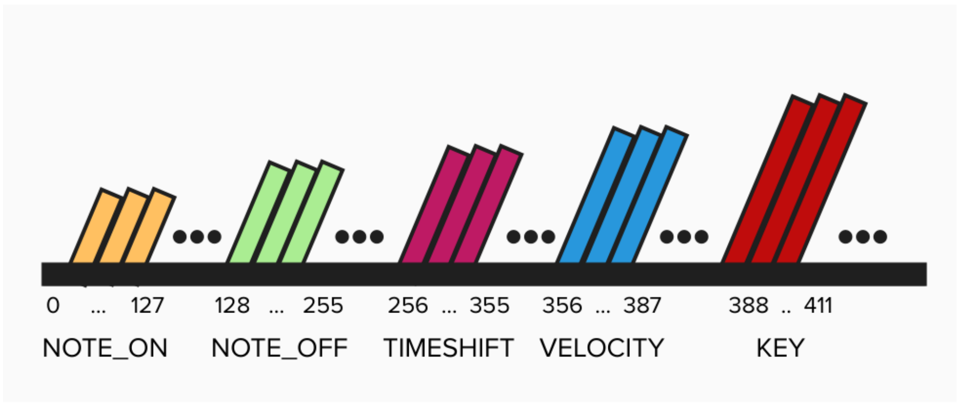


Figure 4.2: The final vocabulary used in the study. On the top of the original vocabulary it also includes the keys as an extension to the itself.

4.1.2 Smart Division of Tracks

Randomly slicing serialized MIDI files proved to have some problems. Since the random slicing is not controlled of the sliced sequence either did not start with proper NOTE_ON events or did not end with NOTE_OFF events. As a result, the model trained on such data may not learn how to end the generation properly. Possibly, not generating NOTE_OFF events at all, resulting in notes pressed until the end. One example generation is presented in Figure 4.3.

To avoid such issues, instead of randomly slicing the encodings, the MIDI data is sliced with respect to the maximum sequence length. First, the approximate time required for l tokens is extracted from a completely encoded MIDI track. This approximated time requirement is later used to slice the MIDI file instead. The reason for that is that slicing the MIDI data directly does not result with the issues mentioned earlier. When the MIDI is directly sliced, any ongoing notes are dropped both at the start of the slice and end of the slice. To avoid losing information at the end of the sequences, the slicing is done with 5% more of the maximum length.

4.1.3 Krumhansl-Schmuckler Algorithm

Krumhansl-Schmuckler algorithm [Tem99] is the state-of-the art algorithm for determining the key of a piece. The algorithm uses a statistical method to analyze the distribution of pitch classes in a piece. Each pitch is assigned with a weight, based on it's frequency in the song. Then the weights are then compared to a set of pre-defined templates, which represent the major and minor keys of Western music. The key that matches the weights most closely is determined to be the key of the piece. However, one of the main drawbacks of this method is that when the piece is atonal[2], it is not able to recognize the piece is atonal. Although, this case is not often seen and in our dataset it is mostly not the case, this situation is something to

[2]An atonal piece lacks the tonal center or a key.

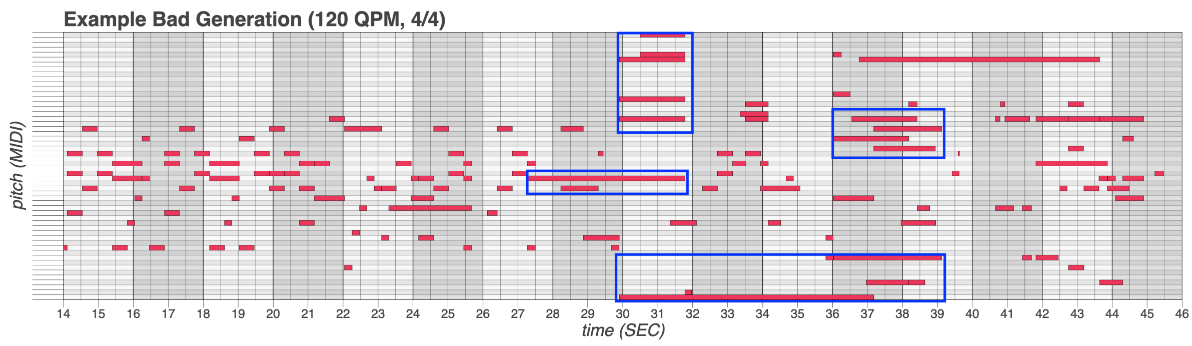


Figure 4.3: Example of a bad generation without smart division. Lot of notes are played much longer than they supposed to be played. Some of them are shown inside the blue squares.

be aware of. Accordingly, in this study we use Krumhansl-Schmuckler algorithm to determine each songs key for controllability tasks.

4.2 Music Transformer

Different form baseline Transformers, Music Transformer [Hua+18] uses a different Attention mechanism. It is because the baseline Attention mechanism does not have relative positional information.

4.2.1 Relative Positional Self-Attention

Relative Positional Self-Attention, as proposed by [SUV18], allows the network to be informed by how far two tokens are apart in the input sequence X. This is especially relevant in music generation since this mechanism can capture structural information such as repetitive patterns like melody or chord progression [Hua+18]. Additionally, we also considered **Masked Self-Attention** case as we don't wish the model to be informed about the future in the training time.

The main difference with the Relative Positional Self-Attention is that learning the tensor S^{rel} with shape (l, l), where l is the sequence length and using it as follows in the Attention equation:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T + S^{\text{rel}}}{\sqrt{D_h}}\right) V \tag{4.1}$$

, the head indices are dropped for clarity.

To learn such a S^{rel} tensor, we firstly require to learn relative positional embeddings E^r in the shape of (H, L, D_h). These embeddings are ordered accordingly to the input sequence. In other words, these embeddings are ordered from $-L+1$ to 0, where 0 being the current token. It holds the pairwise distance information r for each query and key pair i_q and j_k, where $r = j_k - i_q$ and is learned separately for each individual head in the attention block.

Following that, the E^r tensor is reshaped into the R tensor as it can be seen in Figure 4.4. To get the tensor S^{rel}, the R tensor is later multiplied with the query tensor Q, in the original implementation by [SUV18].

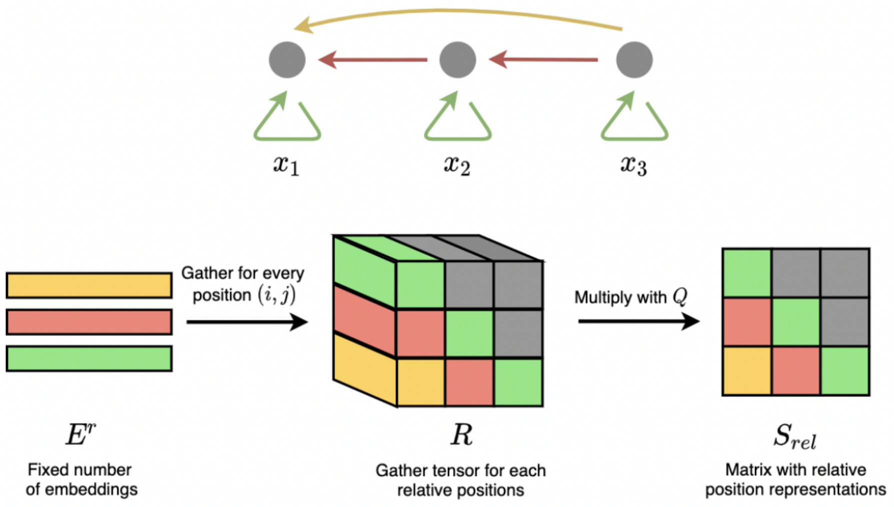

x_1
x_2
x_3
Gather for every position (i, j)
Multiply with Q
E^r
Fixed number of embeddings
R
Gather tensor for each relative positions
S_{rel}
Matrix with relative position representations

Figure 4.4: Original S^{rel} calculation: The green colour represents the token at the 0 (closest), and the yellow is the token at $-L+1$ (furthest). The black colour represents the future tokens, i.e. masked tokens. Taken from [gud20]

Accordingly, the problem with this implementation is its time complexity which is $O(L^2D)$, which makes it's resource consumption high and usage for long sequences relatively challenging.

4.2.2 Memory Efficient Implementation

[Hua+18] suggests an improved implementation of S^{rel} calculation, which we also used in my code. If the number of embeddings in E^r is fixed, the positional embeddings will be the same, depending on the relative position.

With such assumptions, S^{rel} is calculated with the following equation.

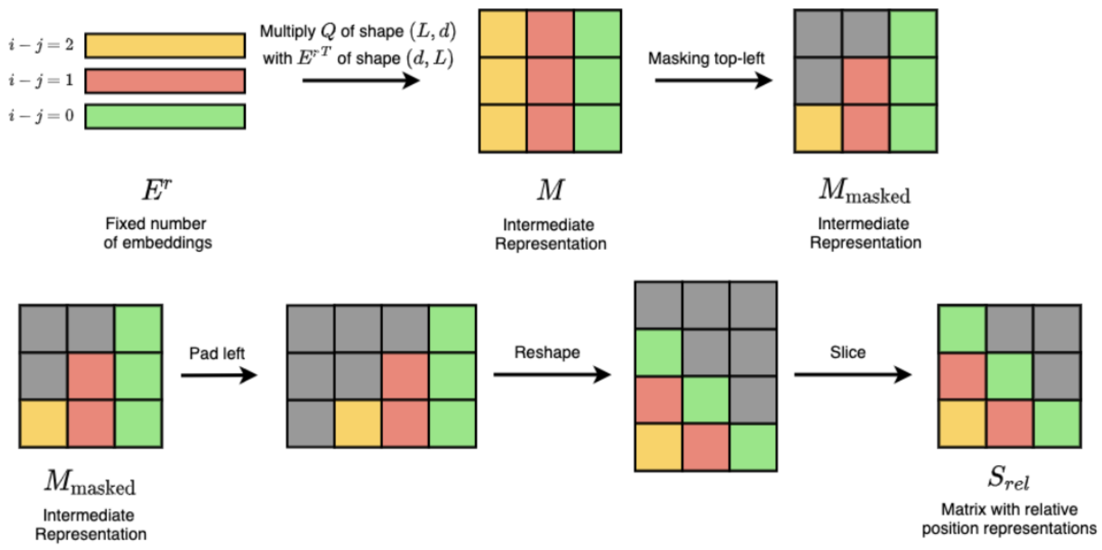
$$\begin{aligned} M &= QE^{r^T} \\ S^{rel} &= Skew(M) \end{aligned} \tag{4.2}$$

, where the $Skew$ function is as follows:

1. Pad column vector M before the leftmost column

2. Reshape M to $(l+1, l)$ shape
3. Slice resulting matrix to retain only the last l rows, resulting in a lower triangular square matrix, which is what is required from S^{rel} in a masked attention case

Accordingly, the main difference is that, instead of using a high dimensional tensor such as R for the whole sequence, applying smaller steps to calculate S^{rel}, which can be also seen in Figure 4.5. With assumptions mentioned earlier, this way of calculating S^{rel} proves to be much more resource efficient, resulting in $O(LD)$ complexity. As a result, this makes such Attention mechanism useful in longer sequences.



$i-j=2$
$i-j=1$
$i-j=0$
Multiply Q of shape (L, d) with E^{rT} of shape (d, L)
Masking top-left
E^r
Fixed number of embeddings
M
Intermediate Representation
M_{masked}
Intermediate Representation
Pad left
Reshape
Slice
M_{masked}
Intermediate Representation
S_{rel}
Matrix with relative position representations

Figure 4.5: Memory efficient Relative Positional Attention, the colour scheme is the same as before. Taken from [gud20]

4.2.3 Relative Positional Local Attention

For very long sequences attending the neighbouring or local tokens instead of all of the tokens in the sequence can be beneficial. Accordingly, another way to use the relative position attention can be in a local way as well.

Specifically, instead of considering the whole input sequence as a whole, the sequence is divided into non-overlapping chunks. Later on, each chunk attends to its own and to the chunk before. Accordingly, the S^{rel} matrix has to include the attention embeddings to the previous chunk as it can be seen in Figure 4.6, and the algorithm to achieve such a tensor is as follows and can be also seen in Figure 4.7:

1. Column vector M masked from top-left and bottom-right

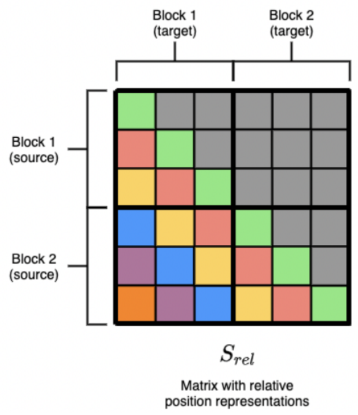


Figure 4.6: S^{rel} in local case. Taken from [gud20]

2. M padded after the rightmost column
3. 1-D flatten M and pad with a mask with length of $l-1$ after the rightmost element
4. Reshape the resulting vector in row-major order
5. Slice the resulting matrix to contain first l rows and last l columns

After following these steps, the missing part S'^{rel} is be obtained.

4.3 Application of CTG methods to Music Transformer

4.3.1 CTRL

CTRL learns the probability distribution $p_\theta(x_i \mid x_{<i}, c)$ via concatenating the control code c with the raw input text. The approach involves completely retraining the LM with the control code concatenated dataset. As a result, the re-trained network is class-conditioned on the classes that are concatenated in the dataset.

In our case, since the key signature of a music piece is not represented as an event in a MIDI file and not present in the original vocabulary, this method does not work as it is for symbolic music generation from MIDI. Instead, the control code is concatenated with the encoded input sequence. With the vocabulary extended for the keys, after the encoding, the token for the keys can be directly concatenated with the encoded MIDI data and fed to the model as input. Specifically, the control code encoding is concatenated at the start of the input prompt.

4.3.2 CoCon

CoCon, on the other hand, proposes a more complex approach. The approach is ingesting a CoCon Block inside a pre-trained decoder-only Transformer model and retraining the

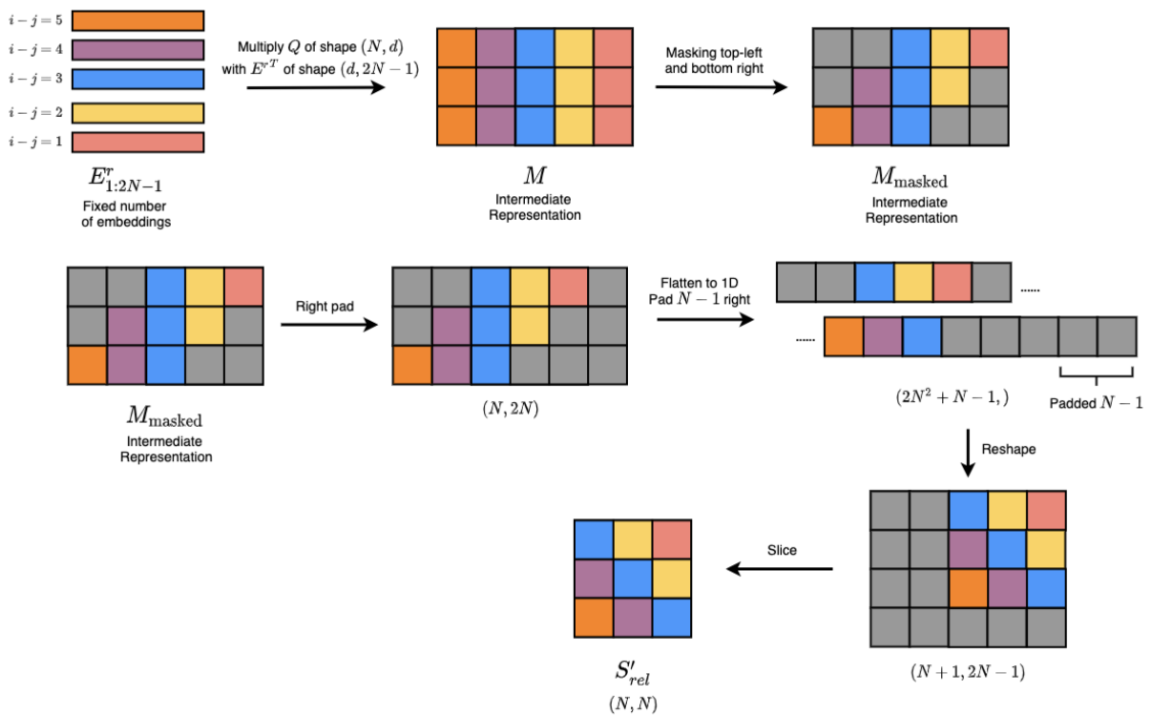

$i-j=5$
$i-j=4$
$i-j=3$
$i-j=2$
$i-j=1$
$E^r_{1:2N-1}$
Fixed number of embeddings
Multiply Q of shape (N,d) with E^{rT} of shape $(d,2N-1)$
M
Intermediate Representation
Masking top-left and bottom right
M_{masked}
Intermediate Representation
M_{masked}
Intermediate Representation
Right pad
$(N,2N)$
Flatten to 1D
Pad $N-1$ right
$(2N^2+N-1,)$
Padded $N-1$
Reshape
$(N+1,2N-1)$
Slice
S'_{rel}
(N,N)

Figure 4.7: Local relative attention. Taken from [gud20]

resulting model. In this concept, CoCon Block is basically an Transformer block: It has one Attention and one Linear layer followed by layer normalization. The role of the CoCon block here is to attend to the control code provided by the user and condition the model on the control code. Ingesting such a block in the model divides the architecture in two: layers before CoCon Block LM_α and layers after the block LM_β. Accordingly, when model is divided as such, one can consider the LM_α as a feature extraction layer for LM_β to calculate logits o_t. The final logit calculation can be seen in Equation 4.3.

$$\mathbf{o}_t = \mathrm{LM}\left(x_{:t-1}\right) = \mathrm{LM}_\beta\left(\mathrm{LM}_\alpha\left(x_{:t-1}\right)\right) = \mathrm{LM}_\beta\left(\mathbf{h}_{:t-1}\right). \tag{4.3}$$

When the LM is divided as such, to calculate the logits we first must calculate the intermediate representations h. These latent representations are calculated by conditioning the input sequences with the control code through the CoCon block with weights ω as:

$$\mathbf{h}'_{:t-1} = \mathrm{CoCon}_\omega\left(\mathbf{h}^{(\mathbf{c})}_{:l_c}, \mathbf{h}_{:t-1}\right) \tag{4.4}$$

where $\mathbf{h}^{(\mathbf{c})}_{:l_c} = LM_\alpha(c)$ is the hidden representation of the control code and l_c is the control code length. In this work the control code is not a sequence but a key, so l_c is always equal to 1.

The CoCon block uses a different attention mechanism in order to attend to the control code:

The CoCon block calculates the Q,K, and V similarly to the vanilla Attention mechanism, however, to attend to the control code, K and V matrices are also calculated with the control code hidden representations from LM_α and concatenated to the K and V calculated from the input sequence.

$$K' = \left[K^{(c)}; K\right], \quad V' = \left[V^{(c)}; V\right] \tag{4.5}$$

$$Z = \text{Attention}(Q, K', V') = \text{Softmax}\left(\frac{QK'^T}{\sqrt{d_k}}\right) V' \tag{4.6}$$

Later the CoCon Block output h'_{t-1} is calculated with the Linear layer inside the block and concatenated with the $h_:t-1$ to be passed to LM_β. Consequently, the probability distribution of the next token is conditioned on the control code c:

$$o_t = \text{LM}_\beta\left(\left[h_{:t-2}; h'_{t-1}\right]\right) \tag{4.7}$$

$$p_{\theta,\omega}\left(x_t \mid c, x_{:t-1}\right) = \text{Softmax}\left(o_t\right) \tag{4.8}$$

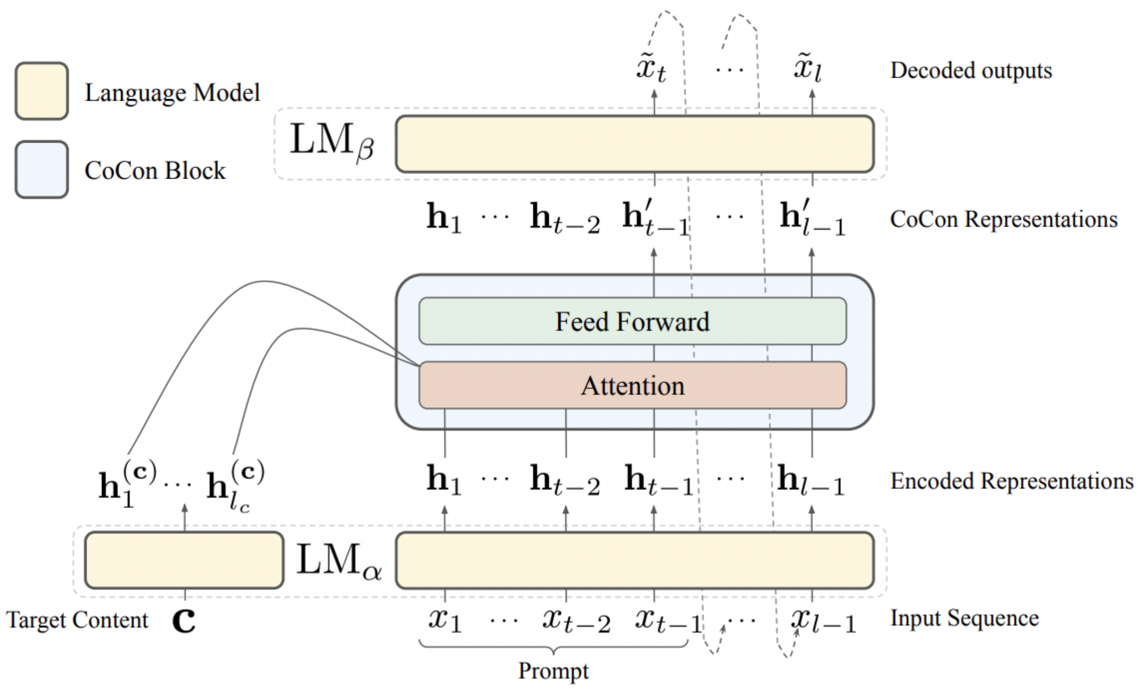

Language Model
CoCon Block
$\tilde{x}_t$ ⋯ $\tilde{x}_l$
Decoded outputs
LM_β
$\mathbf{h}_1 \cdots \mathbf{h}_{t-2}$ $\mathbf{h}'_{t-1}$ ⋯ $\mathbf{h}'_{l-1}$
CoCon Representations
Feed Forward
Attention
$\mathbf{h}_1^{(\mathbf{c})} \cdots \mathbf{h}_{l_c}^{(\mathbf{c})}$
$\mathbf{h}_1 \cdots \mathbf{h}_{t-2}$ $\mathbf{h}_{t-1}$ ⋯ $\mathbf{h}_{l-1}$
Encoded Representations
LM_α
Target Content
$\mathbf{c}$
$x_1 \cdots x_{t-2}$ x_{t-1} ⋯ x_{l-1}
Input Sequence
Prompt

Figure 4.8: Model architecture proposed by the CoCon paper. The content code is also passed through the LM_α and fed to the CoCon block alongside the input sequence. Later the CoCon block's output is fed to LM_β. Taken from [Cha+20]

The original training of CoCon involves a self-supervised approach, meaning that the control codes come from splitting the input data itself. However, as we can already determine the control class of an input data via an algorithm (See Subsection 4.1.3), self-supervised training is not a necessity for the tasks of this study. Accordingly, we drop the self-supervised approach and train the CoCon model with a supervised approach instead, where the control codes are pre-determined by either human annotation or the Krumhansl-Schmuckler algorithm.

4.3.3 GEDI

GEDI generation in essence works based on a PLM and assumes the existence of a smaller CC-LM for guidance. The control code is fed to the CC-LM, the guidance network, instead of the actual generator network and the guidance network tries to guide the generator network. For a two class case, positive and negative, Figure 4.9 holds true.

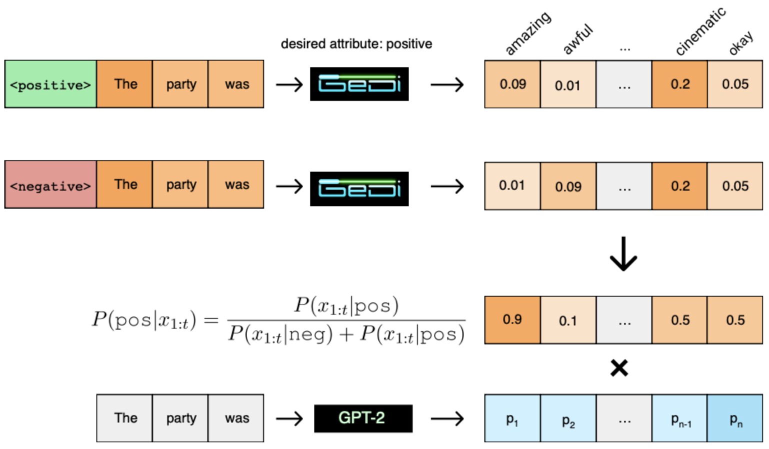


Figure 4.9: Basic intuition behind GEDI. The figure shows the application of GEDI on a single class and its counter class. Taken from [Kra+20]

While, this technique can work on with a independently trained CC-LM, one can also train CC-LM's discriminatively. In order to achieve such training, the generative LM loss L_g is combined with the discriminative loss L_d:

$$L_d = -\frac{1}{N}\sum_{i=1}^{N} \log P_\theta\left(c \mid x_{1:T_i}\right) \tag{4.9}$$

where the posterior distribution P_θ calculated from:

$$P_\theta\left(c \mid x_{1:T_i}\right) = \frac{P(c)P_\theta\left(x_{1:T_i} \mid c\right)^{\alpha/T_i}}{\sum_{c'} P\left(c'\right) P_\theta\left(x_{1:T_i} \mid c'\right)^{\alpha/T_i}}, \tag{4.10}$$

Later generative and discriminative losses combined as:

$$L_{gd} = \lambda L_g + (1-\lambda)L_d \tag{4.11}$$

5 Evaluation and Discussion

In the following chapter, we firstly present the datasets for the tasks and the limitations during collection in Section 5.1. We present the dataset distributions as well as a way to balancing the dataset based on the keys. In Section 5.2 we discuss the metrics to evaluate our approaches. We use a set of quantitative metrics in this study to measure the performance of the class-conditional language models with respect to the baseline language model and the class conditioning performance of the class-conditional language models. We present the evaluation done in Section 5.3, we analyze the performance differences between the models and discuss the evaluation results of the respective models.

5.1 Data

We merged various datasets to train the models used in this work. The merged dataset is divided 80%-10%-10% as train-validation-test datasets respectively. The seed used for such division is kept the same for reproducibility.

5.1.1 Data Collection Limitations

There consists some limitations for data collection because of the description of the tasks. In general, the data used for training and evaluation consists of MIDI file that are polyphonic piano pieces.

The datasets' genre or artists are not considered as main constraints when collecting such data. However, the main constraint is the MIDI files containing piano as main instrument. In the case the files that have multiple piano instruments, the instrument scores are simply merged together and represented as a single instrument.

5.1.2 Datasets

- **Maestro Dataset[1]**: Like [Hua+18], we used also the Maestro Dataset. This dataset is a cleaned up version of Piano-e-comp Dataset[2], where the duplicates are removed. All of the pieces are classical music and are recordings of competitors that played the pieces on a e-piano, so the timings of the notes are not perfect, however, the velocity information enriches the dataset and the data is more natural.

[1]https://magenta.tensorflow.org/datasets/maestro
[2]https://www.piano-e-competition.com/

- **ADL-Piano Dataset**[3]: The ADL-Piano Dataset is a dataset containing polyphonic piano pieces of numerous genres, from classical to rock and from different artists. However, the number of songs per artist is not that high, but the in total there are over 11000 songs in this dataset. Majority of the songs genres are soundtracks and rock.
- **Piano Midi Dataset**[4]: The Piano Midi Dataset, on the other hand, rather small, around 330 songs. All of the pieces are classical from various artists.
- **Vgmidi Dataset**[5]: The Vgmidi Dataset consists of only video game music. Instead of having a separate genre for these music, we considered them as soundtrack as well. In total there are around 4000 songs in the dataset, which are all piano pieces.

Accordingly, the combined dataset consists of above 16000 songs. The songs are sliced accordingly to the method mentioned in Subsection 4.1.2. We created multiple datasets for the tasks described in Chapter 3 under two different umbrella datasets. These umbrella datasets are the original collection or the key balanced version.

5.1.3 Dataset Balancing and Sampling

We sample random sequences from the MIDI files in our dataset. The length of the input sample from each file is a random number between 1 and the maximum sequence length. Accordingly, we also took the distribution balance of the dataset into account as well. The dataset is highly imbalanced in terms of keys. To get a more unbiased towards all classes we had 2 possible options to balance the dataset:

- The songs are transposed[6] to their perfect 4th or 5th[7] and the sample size from each song is 1. The main downside of this approach is that the tonality change of the melodic structures of the songs might not have the same sentiment and the class-conditional training might not work as expected.
- The amount of samples from one file on the other hand depends on the ratio r_i of the class i with the class with the maximum value in the distribution and it is simply calculated by the following formula:

$$r_i = \frac{P(X = i)}{max(P)} \tag{5.1}$$

We selected the second option as means of dataset balancing. Accordingly, the original key distribution can be seen in Figure 5.1, while the distribution after sampling can be seen in Figure 5.2.

[3]https://github.com/lucasnfe/adl-piano-midi

[4]http://www.piano-midi.de/

[5]https://github.com/lucasnfe/vgmidi

[6]Transposing means the key is changed, so all of the notes are modified but the velocity, timing and other variables are the same. The note pitches are moved higher or lower.

[7]Perfect 4th and 5th are the keys on the left and right on the circle of fifths. What is special about them is that even the keys are difference in the notes used is minimal, there is only one note change.

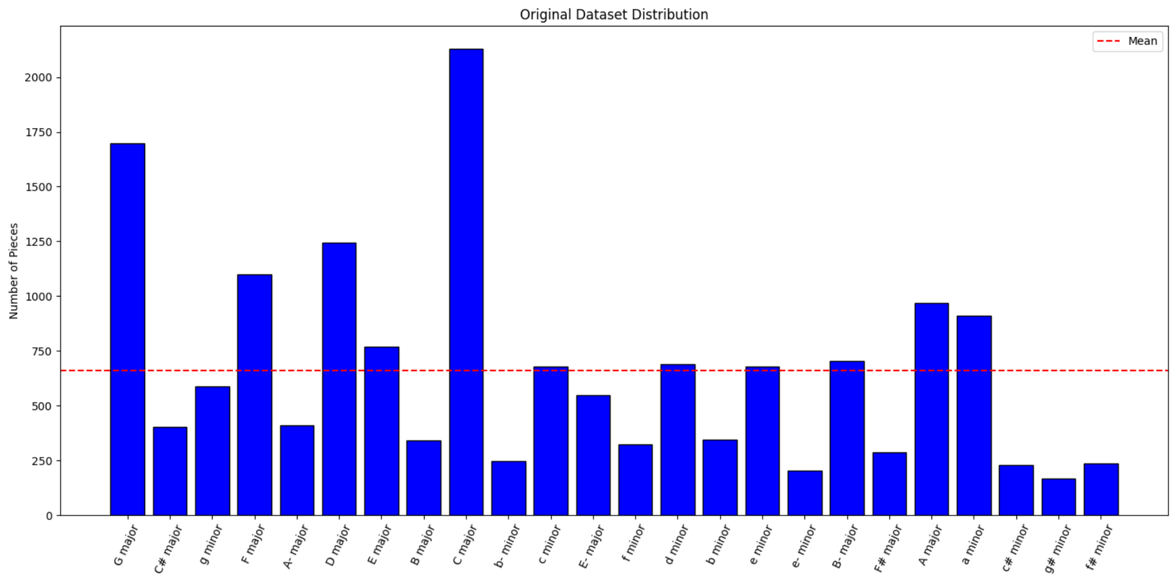


Figure 5.1: Original merged dataset key distribution

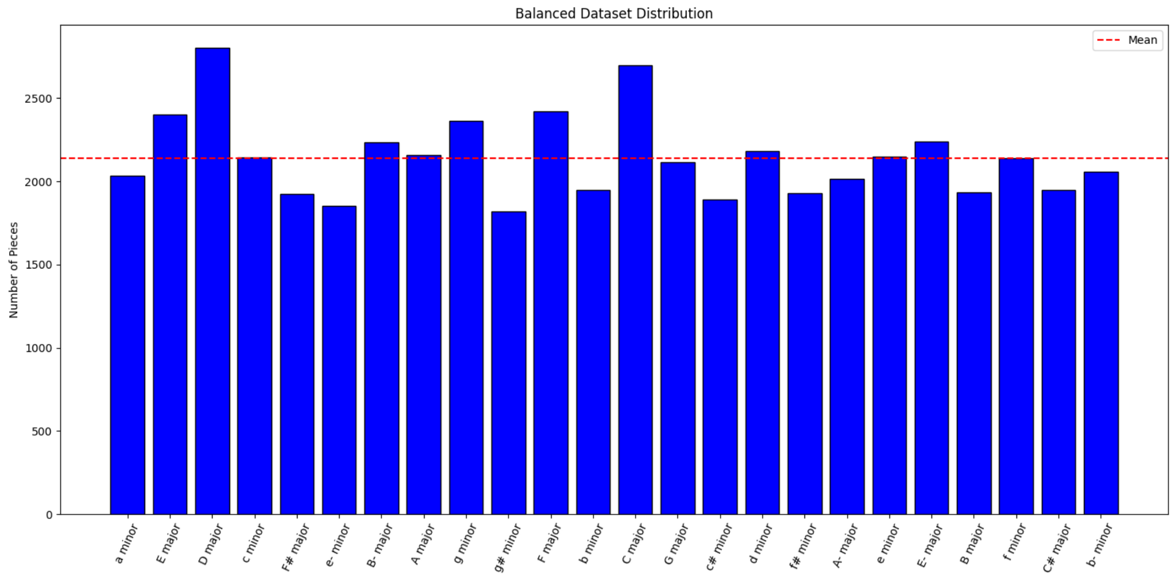


Figure 5.2: Balanced dataset key distribution

5.1.4 Task-Specific Datasets

For the downstream tasks, sub-datasets are created from the original dataset. Specifically, the data collected is already labeled with genres. On the other hand, artists are clearly classified only in Maestro dataset. We implemented the same the of balancing and sampling from the songs in these dataset as well in order to get a more even distribution in the resulting dataset.

5.2 Metrics

Evaluation of music generation is a challenging task and has two aspects to it:

- **Structural Quality**: Structural quality of a musical piece can be simplified to correct playing of notes. Not only notes played should be in the key, but also the harmonic structure needs to be correct as well. Accordingly, the notes that are played should be following the tempo as well. However, since we also use pieces played by humans the musicians cannot follow the tempo with 100% precision.
- **Musical Quality**: Musical quality is a topic that is highly dependent on perspective and it is also resource heavy measurement as human evaluation is required. Often what is done in other works is that the human evaluation predicts whether a given musical piece is generated by an AI or a human. However, as this requires human candidates, in this study we didn't put resources to this quality measurement.

Furthermore, we also evaluate controllability, which refers to the impact of introduction of the control code. We both check whether the generated sequence's key matches target key and the quality of CC-LM in terms of generation quality.

Accordingly we used the following metrics to evaluate our models:

5.2.1 BLEU

Bilingual Evaluation Understudy(BLEU) [Pap+02] is originally a evaluation metric for machine translation. In essence, it works by comparing models predictions and all ground truths of a translation. In our case, we compare the generated song and the original song. The BLEU score is a floating number between 0 and 1, where the higher score means the predictions are more on point.

5.2.2 ROGUE

Recall-Oriented Understudy for Gisting Evaluation (ROGUE) is a rather common metric that is used to evaluate generated text in various tasks. ROGUE mainly focuses on the recall of the model, measuring the effectiveness of generative models by comparing their output to human-generated versions in order to evaluate how much the generated text overlaps with the human-generated text.

Even though, ROGUE is not often used in generation tasks, it is still a valid metric that can be used for our case. As music is a repetitive sequence, through the ROGUE score we can estimate how well the generations follow repetitive patterns in the ground truth. This also shows that, how well the models can pick up the repetitive patterns in the input prompt. However, ROGUE score do does not make sense when the models generate from scratch as the model has no repetitive patterns to follow.

5.2.3 Perplexity

The perplexity serves as an indicator of the language model's ability to predict the succeeding token in a sequence. To compute it, the model's predictions for a specific test set are averaged and transformed into an exponent. A lower perplexity score generally denotes a more proficient language model, as it can make more precise predictions about the following word in a sequence. Conversely, a higher perplexity score suggests a less capable model.

5.2.4 Cross-Entropy

Cross entropy is a measure of the difference between two probability distributions. In a multi-class case this metric compares the output probabilities from the model and the ground truth distribution. We mainly use this metric as a loss on a dataset and compare different models that we trained during this study. Cross-Entropy loss follows the following formula:

$$H(p,q) = -\sum_{i} p(i)\log(q(i)) \tag{5.2}$$

5.2.5 Note Persistency

The objective of Note Persistency is to assess the number of notes produced in the desired key. This metric serves as a reliable evaluation of the task's complexity, as it helps to determine if the model has accurately predicted the notes based on the target key. If the model generates notes that are in the target key, but the resulting key of the generated music is incorrect, it suggests that the model lacks the necessary understanding of the concept of a key based on it's training.

Accordingly, the persistency score is between 0 and 1, where higher score means more notes are in the key. In our test set the average note persistency is 77%, meaning that the usage of borrowed chords and modulations are common. It follows the following equation:

$$Persistency = \frac{N_{inkey}}{N_{total}} \tag{5.3}$$

5.2.6 Accuracy

Accuracy is applied for controllable generation. The aim is to measure how many of generations with target keys the model is able to succeed. As mentioned in Subsection 4.1.3, we use Krumhansl-Schmuckler algorithm to classify the generation with a key and later we compare it to the target key. We later do this comparison for each input and normalize it to get a number between 0 and 1.

5.2.7 Accuracy According to the Classifier

As mentioned in Section 3.4 we train a classifier with the baseline generative network with the aim of understanding how good a classifier from such a LM would perform in classifying keys. Accordingly, we also check the how many of the generated pieces are matching the classifiers prediction. This aim here is to understand what the model thinks the generation is classified as. Similarly to regular accuracy, we apply this over the whole test-set and normalize it to get a number between 0 and 1.

Objective	**Metric**	**Criteria**
Controllability	Accuracy	MAX
	Accuracy According to Classifier	MAX
Structural Quality	Perplexity	MIN
	Cross Entropy	MIN
	BLEU	MAX
	ROGUE	MAX
Musical Quality	Note Persistency	MAX

Table 5.1: The metrics used for evaluations with their objectives. Criteria represents the desired direction of the metrics value. The description of each metric can be found under Section 5.2

5.3 Evaluation

According to the tasks that were described in Chapter 6, we evaluate the baseline music transformer with our dataset, CTRL style music transformer and implementation of CoCon with music transformer separately on the metrics discussed previously. We evaluate their performances on different input sizes, as the input size is a factor that we need to access. For all of the generations we use 1 as temperature value t. Additionally, we evaluate CTRL and CoCon's performances on continuing the input sequences tonality in order to compare their performances with the vanilla music transformer. Furthermore, we provide evaluations of the downstream tasks on their respective tasks.

We present the training outcomes of three models - Music Transformer, CTRL, and CoCon in Table 5.2. From the validation set performances we can see that CoCon achieves similar

Model\Metric	Perplexity	Cross-Entropy
Music Transformer	3.071	1.118
CTRL	3.12	1.148
CoCon	2.941	1.074

Table 5.2: Perplexity and Cross-Entropy loss results on the validation set.

performance with vanilla music transformer, while CTRL performs worse.

As described in Section 4.3 one of the CTG techniques we explored is GEDI. However, as explained before the GEDI generation assumes a CC-LM. Accordingly, as the evaluation results of CTRL and CoCon didn't meet our expectations we did not go into adapting GEDI.

5.3.1 Vanilla Music Transformer

We start by evaluating the vanilla music transformer on our evaluation dataset. We test the performance of the network with different input sizes and let the model generate the rest. From Table 5.3 we conclude the following results:

- The vanilla music transformer is able to continue the note persistency to a high degree with respect to the primer's tonality regardless the input length. This means that the model has a valid understanding on which notes belong to the tonality in question.
- According to both the classifier and the algorithm the generation's tonality is more than half of the time incorrect given a good input size.
- Primer's length affects the quality of the music generated.
- With input size more or equal to 256 tokens, the generation quality is noticeably better. It is expected for the network to have higher ROUGE and BLEU scores due to the references being the snippets of rest of the songs according to the target length.

Input Size\Metric	64	128	256	512
BLEU	0.175	0.273	0.402	0.611
ROUGE	0.455	0.536	0.628	0.762
Note Persistency	0.731	0.733	0.758	0.767
Accuracy	0.258	0.337	0.420	0.531
Accuracy According to Classifier	0.265	0.382	0.465	0.544

Table 5.3: The evaluation results on vanilla Music Transformer on generating sequences continuing a primer. The columns represent the input size given to the model, and the rows represent different metrics for the evaluation.

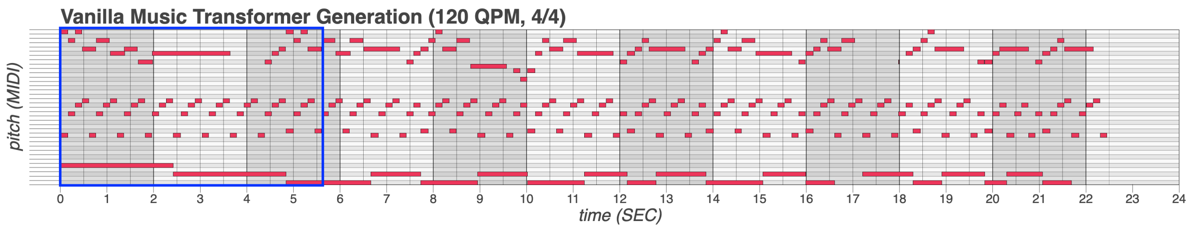


Figure 5.3: Continuation of a Castlevania soundtrack with Vanilla Music Transformer. The part inside the blue square is the primer input and the rest is generated by the network.

5.3.2 CTRL

Firstly for CTRL, we test the models performance on generating sequences with a random closely related key as being the control code, with different input sizes. We kept the maximum input size as half of the target length with the intuition giving more of input sequence makes controlling the generation actually harder. From the Table 5.4 we can extract the following results:

- The model works better when it is also conditioned on a primer input, giving only the control code as the starting point results in outputs that do not follow the control code.
- Fluency of the pieces generated do not heavily depend on the input size, the model is able to generate good pieces both on higher input sizes when continuing a melody or starting completely from scratch.
- Being conditioned on a primer increases the models performance on continuing the piece. Lower input sizes fail heavily when it comes to continuing close to the actual song, however, it also makes the generated song more unique.
- Controllability does not work as we expected. We reach maximum 13% accuracy according to the algorithm we use (See Subsection 4.1.3). However, the note persistency suggests that the introduction of the control code certainly has an effect.
- The classifier trained also agrees with the algorithm when it comes to the true positives, however for false negatives it is completely off. We can definitely see the affect of introducing the control code here.

Secondly, we evaluate the CTRL model on continuing the same key of the input sequence. The reasoning behind this is that we wish to understand the effect of the introduction of the control code to the model. From Table 5.5 we extract the following results:

- The model can continue the primer key as accurately as the vanilla Music Transformer does. Both according to the algorithm and classifier.

Input Size\Metric	1	64	128	256	512
BLEU	0.0003	0.167	0.262	0.394	0.603
ROUGE	0.125	0.416	0.509	0.609	0.743
Note Persistency	0.499	0.564	0.568	0.564	0.557
Accuracy	0.028	0.09	0.09	0.135	0.113
Accuracy According to Classifier	0.028	0.09	0.09	0.145	0.1

Table 5.4: The evaluation results of CTRL on Music Transformer on generating sequences with closely related keys. The columns represent the input size given to the model, and the rows represent different metrics for the evaluation.

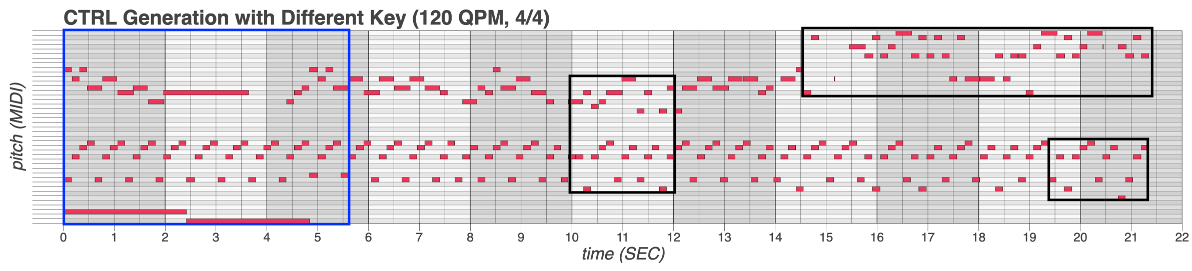


Figure 5.4: Continuation of a Castlevania soundtrack with CTRL. The part inside the blue square is the primer input and the rest is generated by the network. The control key given is a random closely related key of input sequences key. Additionally, we can see notes in the black boxes that originally do not belong to the original key.

- Being conditioned with the same key as the input results in more fluent generations as the model does not try to change the tonality.
- We can also see that the model is able generate structurally more correct pieces when continuing the primer. However, controllability still doesn't work as we expected. The model can continue the primer key even when conditioned on the key in half of the generations
- Note persistency is close to the datasets original mean note persistency when the control code is the same as the primer.

5.3.3 CoCon

Here, we present the evaluation results of the same experiments that were run on CTRL on CoCon. The test dataset as well as the setup is the same. Accordingly, for generating sequences with a closely related key as the condition we get Table 5.6 and infer the following:

- CoCon also works better when conditioned on a primer input. The network fails to generate sequences close to the target key when only control code is given.

Input Size\Metric	64	128	256	512
BLEU	0.169	0.264	0.391	0.611
ROUGE	0.425	0.509	0.594	0.755
Note Persistency	0.731	0.769	0.760	0.767
Accuracy	0.317	0.304	0.396	0.509
Accuracy According to Classifier	0.294	0.331	0.425	0.569

Table 5.5: The evaluation results of CTRL on Music Transformer on generating sequences with the same key as the primer input. The columns represent the input size given to the model, and the rows represent different metrics for the evaluation.

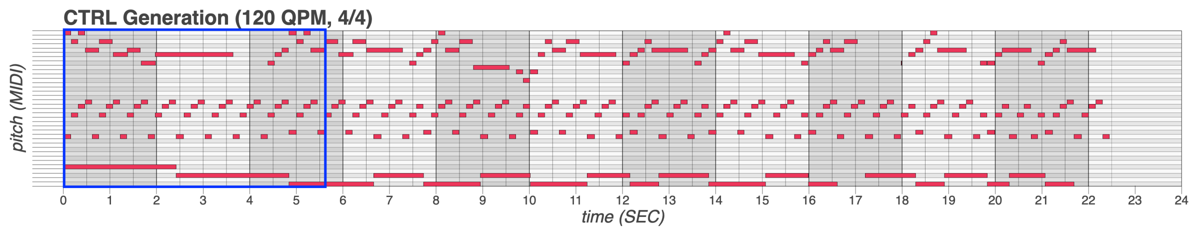


Figure 5.5: Continuation of a Castlevania soundtrack with CTRL with the blue part as primer input. The control key given is the same as the input sequences key. We can see that CTRL's generation was nearly identical to the output of Vanilla Music Transformer.

- Fluency of the pieces generated do not heavily depend on the input size, the model is able to generate good pieces both on higher input sizes when continuing a melody or starting completely from scratch.
- Lower input size results in more unique generations. However, when big enough input is given the model is able to generate pieces that follow the input.
- Controllability doesn't work as expected with CoCon as well. The network is able to put notes from the target key, however fails to modulate.
- CoCon outputs agree more with the classifier than CTRL. It means that the network has more similar understanding of tonality with the classifier than CTRL.

Similar to CTRL we tested CoCon for continuing the input sequences key. From Table 5.7 we infer the following results:

- CoCon can continue the input sequence's key as well as the vanilla music transformer and even better in shorter sequences.
- Note persistency suggests that CoCon can generate sequences that is close to the test datasets note persistency.

Input Size\Metric	1	64	128	256	512
BLEU	0.0016	0.177	0.279	0.406	0.614
ROUGE	0.154	0.454	0.547	0.635	0.766
Note Persistency	0.494	0.554	0.560	0.564	0.572
Accuracy	0.044	0.064	0.102	0.140	0.11
Accuracy According to Classifier	0.076	0.084	0.090	0.122	0.13

Table 5.6: The evaluation results of CoCon on Music Transformer on generating sequences with closely related keys. The columns represent the input size given to the model, and the rows represent different metrics for the evaluation.

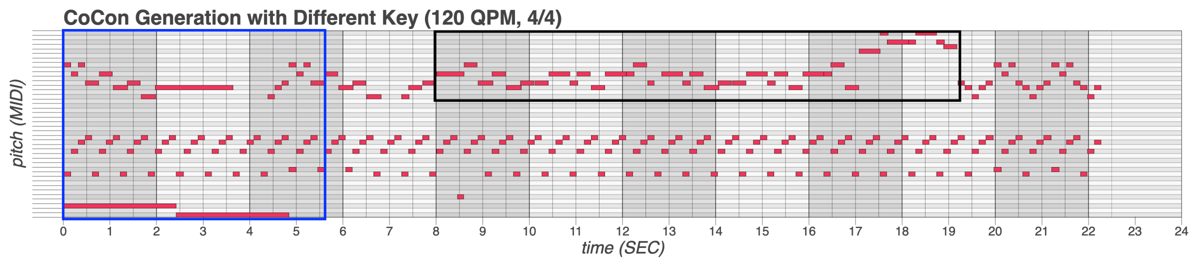


Figure 5.6: Continuation of a Castlevania soundtrack with CoCon. The part inside the blue square is the primer input and the rest is generated by the network. The control key given is a random closely related key of input sequences key. Additionally, we can see notes in the black boxes that originally do not belong to the original key.

- Regardless the control code, the generation performance is similar and slightly better than vanilla music transformer as it is suggested from the training results.
- Similar to CTRL, CoCon also generates fluent results more often when conditioned on continuing the tonality instead of changing it.

Input Size\Metric	64	128	256	512
BLEU	0.175	0.276	0.406	0.613
ROUGE	0.452	0.553	0.625	0.762
Note Persistency	0.733	0.759	0.760	0.763
Accuracy	0.295	0.347	0.456	0.587
Accuracy According to Classifier	0.271	0.408	0.467	0.577

Table 5.7: The evaluation results of CoCon on generating sequences continuing a primer. The columns represent the input size given to the model, and the rows represent different metrics for the evaluation.

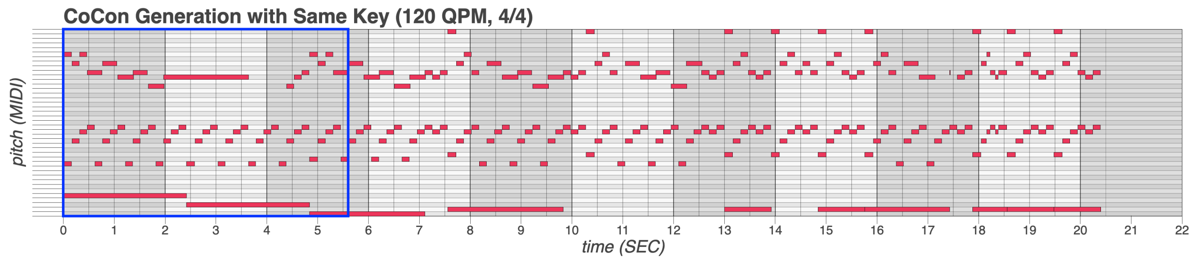


Figure 5.7: Continuation of a Castlevania soundtrack with CoCon. The part inside the blue square is the primer input and the rest is generated by the network. CoCon is also able to follow the primer as good as vanilla music transformer.

5.3.4 Downstream Tasks

Lastly we evaluate the LM's performance on downstream tasks (See Section 3.4 for more explanation on downstream tasks). The results on all of the classification tasks are nothing exceptional to say the least. From the Table 5.8 we conclude the following results:

- Artist and genre classification fails heavily. Due to the imbalance of the dataset and the number of classes in respective datasets, the models are biased towards the artists or genres that appear more often and have a lot of false positives and true negatives. Even though with the sampling we cannot overcome this as we sample *n* amount of sequences from a song depending on it's class. This results in heavily overlapping input data for classes that appear fewer than the dominant classes and pushes the trained network towards memorizing such sequences.
- Key classification perform relatively better than the other classification tasks. While still performing okay at best the model is able to classify them more precisely and accurately.

Task\Metric	f1-Score	Accuracy	Cross-Entropy
Key Classification	0.492	0.582	1.398
Genre Classification	0.164	0.573	1.349
Artist Classification	0.120	0.500	2.091

Table 5.8: Results of classification tasks on the test set, using the vanilla music transformer as the backbone.

Accordingly, when we check the confusion matrices (we didn't share the confusion matrix for artist classification due to number of classes - 43), we can see that the models are able to classify dominant classes much more precisely than others:

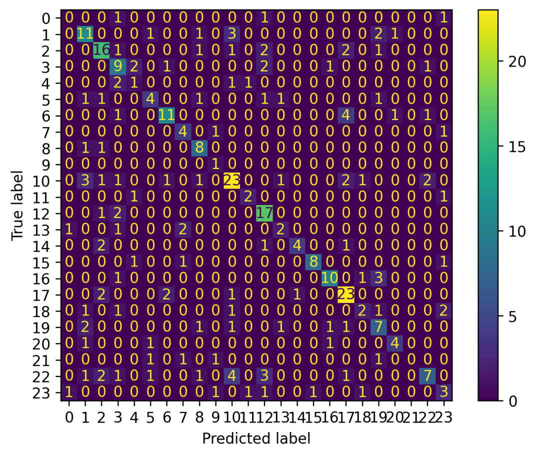


Figure 5.8: Confusion matrix of key classification on the test set.

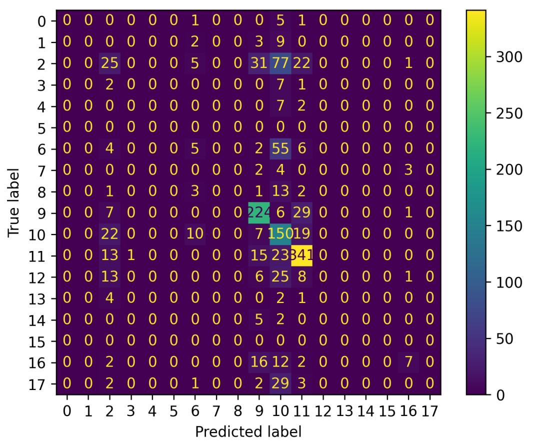


Figure 5.9: Confusion matrix of genre classification on the test set.

Furthermore, we experimented with a limited number of genres in order to further understand the ability of the network to recognise abstract informations and whether it is possible to perform such classification tasks with the current setup. From Figure 5.10 we can infer that, the model can indeed can understand the concepts to some degree with current encoding style and architecture. The 'Unknown' class is actually not unknown but labelled as such and belongs to the other classes, which in the end reduces the performance. However, this still the proves the effect of data imbalance in the genre classification.

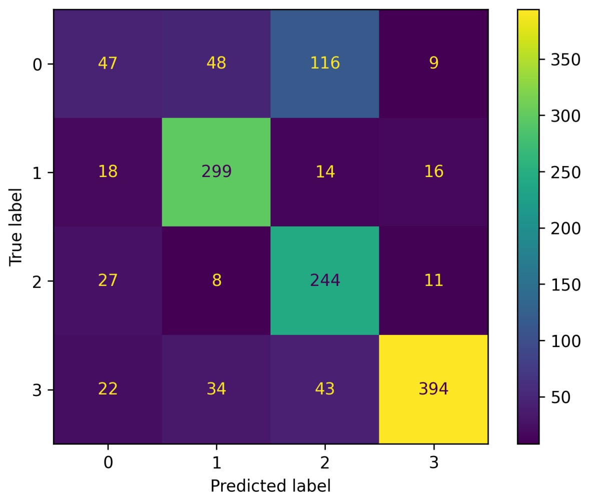


Figure 5.10: Genre classification with 4 classes: 0 - unknown, 1 - classical, 2 - rock, 3 - soundtrack. Achieving 0.66 f1 score in test set.

5.4 Discussion

Overall, when all three models are compared, they perform similarly when it comes to continuing the key of the primer input. While there are small differences between the performances, all three models are able to generally continue the repetitive patterns in the input prompt and add something to the sequence. Specifically, CoCon performs same as the music transformer, while CTRL performs slightly worse when conditioned on the same tonality as the input. From this, we can infer that CoCon as in text generation worked better in music generation as well.

However the introduction of the control code seems to shake the models quite a bit. Both CTRL and CoCon do not work as we expected: With lower input sizes the models should be able to generate sequences belonging the desired tonality, while the performance should drop with the input prompt, especially with longer prompts. However, the results have shown that, the sequences generated from scratch have less accuracy towards the desired tonality. There could be several factors contributing to this: Encoding style, the effectiveness of the techniques or training task itself. It could be that the instead of just conditioning the whole sequence on a single tonality we should hold modulation into account and train the networks to modulate the songs.

Model\Input Length	1	64	128	256	512
Music Transformer	631	944	930	929	945
CTRL	250	940	914	916	932
CoCon	674	914	909	917	945

Table 5.9: Average generation length depending on the prompt length for three models.

One other result we have seen during our evaluation is that all three models do not meet the target length when generating from scratch as it can be seen in Table 5.9. This is in some way expected as we trained the networks on a seq2seq task. Accordingly, this can also indicate the number of shorter sequence samples in the training data is not enough or model focuses too much on finding patterns to repeat from the input prompt. This strongly agrees with the evaluation metrics on the generation quality and could potentially answer why the techniques failed to generate sequences on the desired tonality with no smaller input prompts. In the light of this, we can actually divide the training task in two: Generating symbolic music from scratch, applying modulations to a ongoing primer.

6 Future Work

In order to address the limitations and challenges of our our adaptations and approach and to investigate potential directions for future research, we provide a list of possible future work:

6.1 Future Work in Music Generation Metrics

One possible future work direction is to evaluate music generation in a more quantitative manner and whether the generation resulted in more musically correct and pleasant pieces. Often in music generation evaluation, the musical quality is evaluated by humans as there isn't a reliable quantitative metric for evaluating the musical quality of the generations. Accordingly, the musical pleasantness is a subjective manner as the musical preferences subject to change from person to person.

This would involve exploring the limitations of musical quality evaluation as well as to understand what makes a musical piece generally pleasant without being constrained with musical preferences. Accordingly, one general concept in music theory that defines musical and structural quality of a song is harmony (See Section 2.1). Harmony in simple terms, defines the note-to-note relationships under a tonality. We may be able to improve music generation evaluation by reliably quantifying the harmonic correctness of a musical piece. As an early attempt, we implemented a simple metric in order to evaluate the harmony of the generations (See Subsection 5.2.5).

Furthermore, the quality of a musical piece is not only subject to change with note-to-note relationships. The tempo and velocity of the notes and chords also have an effect on the musical quality of a song, as the tempo of the melody has an heavy affect on the sentiment of a melody. For instance, a slower tempo can make a melody more relaxed, meanwhile when the same set of notes are played with a faster tempo the song will feel more energized and as a result will have a whole different sentiment. Additionally, the rhythmic structure of a melody is a key concept that makes such melody more rememberable. As in text, in music as well there are additional building blocks to a song such as motif [Wik23g] or phrases [Wik23i]. By correctly measuring the usage of such building blocks we can improve the quality of generative models.

6.2 Future Work in Symbolic Music Encoding

One possible direction for future work in symbolic music encoding is that introducing more abstract dimensions into the embeddings such as sentiment or note-to-note relationships. As of now the current implementations of the embeddings are in a more structural setting: [Oor+18b] proposes an absolute-time based embedding style in integer space whereas [HY20] suggests a bar based embedding style in string space. Both of these embedding approaches are centered around MIDI events and prioritize structural accuracy rather than sentimental features. One approach to adding these sentimental values might be similar to a term-document matrix, since the vocabulary size is much more limited than a natural language the computational complexity should not be as much as an issue when compared.

Another possible approach to implementing such features might be a more modern technique such as BERT or Word2Vec. As their performances are much better in natural languages and adaptation of BERT style embeddings to symbolic music generation may be able to yield better results. For instance, MusicBERT [Zen+21] is an early approach to using BERT style embeddings in symbolic music representation, however, one of the main drawbacks of the approach is that it is using high level features as the base of the encodings. By modifying the BERT training to crate embeddings for MIDI events it may be possible to generate embeddings that are more representative in terms of sentiment and tonality while keeping the structural representation intact. Such embeddings would not necessarily be high dimensional as the vocabulary size is not as huge as the vocabulary size of a natural language.

6.3 Future Work in Applications of CTG Techniques on Symbolic Music Generation

We can see that the application of CTG methods to the symbolic music generation has it's effects in terms of quality of the generated outputs. While we focused on applying the CTG techniques to the music generation domain, we could not discover techniques specifically tailored for symbolic music generation. One possibility would be integrating modulations better in the training task. Since we can give an input sequence to the model it might not be completely clear to the model how modulations work, because of the note correlations between keys. Training the model on specifically modulating the key to another key might result in a better controllability in terms of tonality success. Accordingly, further exploration on tailoring CTG techniques would provide valuable insight on application of such techniques on both on music and other language like sequences.

In addition, there are other forms of CTG techniques available for exploration for symbolic music generation besides the ones that involve retraining or refactoring. Techniques that involve reinforcement learning especially have been proven to result in exceptional training outcomes. Other techniques might involve Plug-and-Play or more prompt based techniques.

6.4 General Future Work

One general future work might be constricting such applications on specific genres and tonality. In this study, we worked with multiple genres, however, as different genres include usage of different tonalities, it might have an effect on the generation quality. One other restriction could be controlling specific keys only. This would require playing with the dataset heavily, as some of the keys are under-represented, for instance G# Minor, while others are over-represented, such as C Major. Since the junction between different tonalities is a lot in terms of notes, such limitation on keys might have huge positive effects, as the model doesn't need to care about some of the notes.

Another direction on controlling the tonality is redefining the training task itself. The training tasks we worked on are seq2seq, which generally works better when there are some phrases or motifs in the input prompt as the model tries to find repetitive patterns to follow. However, as our evaluation suggests the networks that we trained generate much lower quality sequences when conditioned on prompts without such patterns. To include better controllability without changing the task completely, the performance could benefit greatly from training the model also on modulations.

7 Conclusion

In conclusion, with this study, our goal was to adapt and apply multiple CTG techniques on music generation and evaluate and compare their performances. We aimed to achieve these objectives by adapting multiple CTG techniques, specifically CTRL and CoCon, on symbolic music generation tasks with a custom dataset on polyphonic piano music on controlling tonality. The replication of the baseline network showed promising results, however, the adaptation of CTG methods did not exactly meet our objectives. We expected the models to be able to continue the sequences with the target tonality given any closely related key. However, while the models are able to introduce notes from target tonalities, they failed to achieve total tonal change in the generation. This being said, their performance aligns with our assumptions when conditioned on the same key with the input.

Accordingly, we evaluated both CTRL and CoCon on different input sizes for continuation of the tonality and changing the tonality to a closely related key. Both are able follow repetitive patterns in the input prompt similar to music transformer. When given a reasonable input sequence, around 128 to 256 tokens, which is enough to include repetitive patterns, both of the models achieve ROUGE scores around 0.5 - 0.6, meaning they can capture the such patterns and align with the original sequence. Additionally, CoCon is able to reach up to 77% note persistency and 43% tonality accuracy when continuing the tonality, while CTRL performs a bit worse. However, when conditioned on a different key the note persistency and tonality accuracy drops down heavily, to around 56% note persistency and 14% tonality accuracy for both of the models. Overall, from our evaluations CoCon slightly outperforms CTRL in the controllable generation task.

Another one of our intuitions was that generation from scratch would work better in terms of controllability as changing the key of an existing melody would be harder as the network has to apply modulation, however, the models performed better when conditioned on a existing melody instead. We believe that the outcomes we achieved are affected by the encoding style, training task itself, and the difficulty and ambiguity of the aspect that we aimed to control.

List of Figures

List of Tables

Bibliography

[Bit+22] R. M. Bittner, J. J. Bosch, D. Rubinstein, G. Meseguer-Brocal, and S. Ewert. *A Lightweight Instrument-Agnostic Model for Polyphonic Note Transcription and Multipitch Estimation*. 2022. arXiv: `2203.09893 [cs.SD]`.

[Cha+20] A. Chan, Y. Ong, B. Pung, A. Zhang, and J. Fu. "CoCon: A Self-Supervised Approach for Controlled Text Generation". In: *CoRR* abs/2006.03535 (2020). arXiv: `2006.03535`. URL: `https://arxiv.org/abs/2006.03535`.

[Dat+19] S. Dathathri, A. Madotto, J. Lan, J. Hung, E. Frank, P. Molino, J. Yosinski, and R. Liu. "Plug and Play Language Models: A Simple Approach to Controlled Text Generation". In: *CoRR* abs/1912.02164 (2019). arXiv: `1912.02164`. URL: `http://arxiv.org/abs/1912.02164`.

[DMP18] C. Donahue, J. J. McAuley, and M. S. Puckette. "Synthesizing Audio with Generative Adversarial Networks". In: *CoRR* abs/1802.04208 (2018). arXiv: `1802.04208`. URL: `http://arxiv.org/abs/1802.04208`.

[GFC21] T. Gao, A. Fisch, and D. Chen. "Making Pre-trained Language Models Better Few-shot Learners". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 3816–3830. DOI: `10.18653/v1/2021.acl-long.295`. URL: `https://aclanthology.org/2021.acl-long.295`.

[Goo+14] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. *Generative Adversarial Networks*. 2014. arXiv: `1406.2661 [stat.ML]`.

[gud20] gudgud96. *Annotated Music Transformer*. Accessed April 5, 2023. 2020.

[HS97] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[Hua+18] C. A. Huang, A. Vaswani, J. Uszkoreit, N. Shazeer, C. Hawthorne, A. M. Dai, M. D. Hoffman, and D. Eck. "An Improved Relative Self-Attention Mechanism for Transformer with Application to Music Generation". In: *CoRR* abs/1809.04281 (2018). arXiv: `1809.04281`. URL: `http://arxiv.org/abs/1809.04281`.

[HY20] Y.-S. Huang and Y.-H. Yang. "Pop Music Transformer: Beat-Based Modeling and Generation of Expressive Pop Piano Compositions". In: MM '20. Seattle, WA, USA: Association for Computing Machinery, 2020, pp. 1180–1188. ISBN: 9781450379885. DOI: `10.1145/3394171.3413671`. URL: `https://doi.org/10.1145/3394171.3413671`.

[Hut21a] R. Hutchinson. *Major Keys*. MusicTheory, openly–licensed online college music theory textbook. Accessed April 5, 2023. Aug. 2021. URL: https://musictheory.pugetsound.edu/mt21c/MajorKeySignatures.html.

[Hut21b] R. Hutchinson. *Minor Keys Keys*. MusicTheory, openly–licensed online college music theory textbook. Accessed April 5, 2023. Aug. 2021. URL: https://musictheory.pugetsound.edu/mt21c/MinorKeySignatures.html.

[JLY20] S. Ji, J. Luo, and X. Yang. "A Comprehensive Survey on Deep Music Generation: Multi-level Representations, Algorithms, Evaluations, and Future Directions". In: *CoRR* abs/2011.06801 (2020). arXiv: 2011.06801. URL: https://arxiv.org/abs/2011.06801.

[Kes+19] N. S. Keskar, B. McCann, L. R. Varshney, C. Xiong, and R. Socher. "CTRL: A Conditional Transformer Language Model for Controllable Generation". In: *CoRR* abs/1909.05858 (2019). arXiv: 1909.05858. URL: http://arxiv.org/abs/1909.05858.

[Kra+20] B. Krause, A. D. Gotmare, B. McCann, N. S. Keskar, S. R. Joty, R. Socher, and N. F. Rajani. "GeDi: Generative Discriminator Guided Sequence Generation". In: *CoRR* abs/2009.06367 (2020). arXiv: 2009.06367. URL: https://arxiv.org/abs/2009.06367.

[LR21] Z. Lin and M. O. Riedl. "Plug-and-Blend: A Framework for Controllable Story Generation with Blended Control Codes". In: *CoRR* abs/2104.04039 (2021). arXiv: 2104.04039. URL: https://arxiv.org/abs/2104.04039.

[LPM15] M. Luong, H. Pham, and C. D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: *CoRR* abs/1508.04025 (2015). arXiv: 1508.04025. URL: http://arxiv.org/abs/1508.04025.

[MSC18] H. H. Mao, T. Shin, and G. W. Cottrell. "DeepJ: Style-Specific Music Generation". In: *CoRR* abs/1801.00887 (2018). arXiv: 1801.00887. URL: http://arxiv.org/abs/1801.00887.

[MBM21] M. Mohaimenuzzaman, C. Bergmeir, and B. Meyer. *Pruning vs XNOR-Net: A Comprehensive Study on Deep Learning for Audio Classification in Microcontrollers*. Aug. 2021.

[Oor+16] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu. "WaveNet: A Generative Model for Raw Audio". In: *CoRR* abs/1609.03499 (2016). arXiv: 1609.03499. URL: http://arxiv.org/abs/1609.03499.

[Oor+18a] S. Oore, I. Simon, S. Dieleman, D. Eck, and K. Simonyan. "This Time with Feeling: Learning Expressive Musical Performance". In: *CoRR* abs/1808.03715 (2018). arXiv: 1808.03715. URL: http://arxiv.org/abs/1808.03715.

[Oor+18b] S. Oore, I. Simon, S. Dieleman, D. Eck, and K. Simonyan. "This Time with Feeling: Learning Expressive Musical Performance". In: *CoRR* abs/1808.03715 (2018). arXiv: 1808.03715. URL: http://arxiv.org/abs/1808.03715.

[Ope21] OpenAI. *GPT-3.5 Language Model*. https://openai.com. Computer software. 2021.

[Pap+02] 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*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://aclanthology.org/P02-1040.

[Pas+21] D. Pascual, B. Egressy, C. Meister, R. Cotterell, and R. Wattenhofer. "A Plug-and-Play Method for Controlled Text Generation". In: *CoRR* abs/2109.09707 (2021). arXiv: 2109.09707. URL: https://arxiv.org/abs/2109.09707.

[Rad+18] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. "Improving language understanding by generative pre-training". In: (2018).

[Raf+19] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *CoRR* abs/1910.10683 (2019). arXiv: 1910.10683. URL: http://arxiv.org/abs/1910.10683.

[SUV18] P. Shaw, J. Uszkoreit, and A. Vaswani. "Self-Attention with Relative Position Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 464–468. DOI: 10.18653/v1/N18-2074. URL: https://aclanthology.org/N18-2074.

[SO17] I. Simon and S. Oore. *Performance RNN: Generating Music with Expressive Timing and Dynamics*. https://magenta.tensorflow.org/performance-rnn. Blog. 2017.

[Sum19] T. A. Summer. *Transformer - The Key Player in NLP*. https://theaisummer.com/transformer/. Accessed April 15, 2023. 2019.

[Tam+19] P. Tambwekar, M. Dhuliawala, L. J. Martin, A. Mehta, B. Harrison, and M. O. Riedl. "Controllable Neural Story Plot Generation via Reward Shaping". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 5982–5988. DOI: 10.24963/ijcai.2019/829. URL: https://doi.org/10.24963/ijcai.2019/829.

[Tay+21] Y. Tay, M. Dehghani, J. Rao, W. Fedus, S. Abnar, H. W. Chung, S. Narang, D. Yogatama, A. Vaswani, and D. Metzler. "Scale Efficiently: Insights from Pre-training and Fine-tuning Transformers". In: *CoRR* abs/2109.10686 (2021). arXiv: 2109.10686. URL: https://arxiv.org/abs/2109.10686.

[Tem99] D. Temperley. "What's Key for Key? The Krumhansl-Schmuckler Key-Finding Algorithm Reconsidered". In: *Music Perception: An Interdisciplinary Journal* 17 (1999), pp. 65–100.

[Tod89] P. M. Todd. "A Connectionist Approach to Algorithmic Composition". In: *Computer Music Journal* 13.4 (1989), pp. 27–43. ISSN: 01489267, 15315169. URL: http://www.jstor.org/stable/3679551 (visited on 04/19/2023).

[TEC03] G. Tzanetakis, A. Ermolinskyi, and P. Cook. "Pitch Histograms in Audio and Symbolic Music Information Retrieval". In: *Journal of New Music Research* 32.2 (2003), pp. 143–152. DOI: 10.1076/jnmr.32.2.143.16743. eprint: https://www.tandfonline.com/doi/pdf/10.1076/jnmr.32.2.143.16743. URL: https://www.tandfonline.com/doi/abs/10.1076/jnmr.32.2.143.16743.

[Vas+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[Wik23a] Wikipedia. *Circle of Fifths*. Wikipedia, The Free Encyclopedia. Accessed January 13, 2023. Mar. 2023. URL: https://en.wikipedia.org/wiki/Circle_of_fifths.

[Wik23b] Wikipedia. *Closely Related Keys*. Wikipedia, The Free Encyclopedia. Accessed January 12, 2023. Mar. 2023. URL: https://en.wikipedia.org/wiki/Closely_related_key.

[Wik23c] Wikipedia. *Harmony*. Wikipedia, The Free Encyclopedia. Accessed April 26, 2023. Apr. 2023. URL: https://en.wikipedia.org/wiki/Harmony.

[Wik23d] Wikipedia. *MIDI*. Wikipedia, The Free Encyclopedia. Accessed November 13, 2022. Mar. 2023. URL: https://en.wikipedia.org/wiki/MIDI.

[Wik23e] Wikipedia. *Modulation in Music Theory*. Wikipedia, The Free Encyclopedia. Accessed January 12, 2023. Mar. 2023. URL: https://en.wikipedia.org/wiki/Modulation_(music).

[Wik23f] Wikipedia. *Monophony*. Wikipedia, The Free Encyclopedia. Accessed April 5, 2023. Mar. 2023. URL: https://en.wikipedia.org/wiki/Monophony.

[Wik23g] Wikipedia. *Motif*. Wikipedia, The Free Encyclopedia. Accessed April 29, 2023. Apr. 2023. URL: https://en.wikipedia.org/wiki/Motif_(music).

[Wik23h] Wikipedia. *Phrase*. Wikipedia, The Free Encyclopedia. Accessed 3 May, 2023. Apr. 2023. URL: https://en.wikipedia.org/wiki/Borrowed_chord.

[Wik23i] Wikipedia. *Phrase*. Wikipedia, The Free Encyclopedia. Accessed April 29, 2023. Apr. 2023. URL: https://en.wikipedia.org/wiki/Phrase_(music)#:~:text=A%5C%20phrase%5C%20is%5C%20a%5C%20substantial,of%5C%20music%5C%20from%5C%20linguistic%5C%20syntax..

[Wik23j] Wikipedia. *Tonality*. Wikipedia, The Free Encyclopedia. Accessed January 12, 2023. Mar. 2023. URL: https://en.wikipedia.org/wiki/Tonality.

[Wik23k] Wikipedia. *Writer's Block*. Wikipedia, The Free Encyclopedia. Accessed April 5, 2023. Mar. 2023. URL: https://en.wikipedia.org/wiki/Writer%27s_block.

[Yu+22] B. Yu, P. Lu, R. Wang, W. Hu, X. Tan, W. Ye, S. Zhang, T. Qin, and T.-Y. Liu. *Museformer: Transformer with Fine- and Coarse-Grained Attention for Music Generation*. 2022. arXiv: `2210.10349 [cs.SD]`.

[Zen+21] M. Zeng, X. Tan, R. Wang, Z. Ju, T. Qin, and T. Liu. "MusicBERT: Symbolic Music Understanding with Large-Scale Pre-Training". In: *CoRR* abs/2106.05630 (2021). arXiv: `2106.05630`. URL: `https://arxiv.org/abs/2106.05630`.

[Zha+22] H. Zhang, H. Song, S. Li, M. Zhou, and D. Song. "A Survey of Controllable Text Generation using Transformer-based Pre-trained Language Models". In: *CoRR* abs/2201.05337 (2022). arXiv: `2201.05337`. URL: `https://arxiv.org/abs/2201.05337`.

[Zha+19] Y. Zhang, S. Sun, M. Galley, Y. Chen, C. Brockett, X. Gao, J. Gao, J. Liu, and B. Dolan. "DialoGPT: Large-Scale Generative Pre-training for Conversational Response Generation". In: *CoRR* abs/1911.00536 (2019). arXiv: `1911.00536`. URL: `http://arxiv.org/abs/1911.00536`.

[Zhe+21] K. Zheng, R. Meng, C. Zheng, X. Li, J. Sang, J. Cai, and J. Wang. "EmotionBox: a music-element-driven emotional music generation system using Recurrent Neural Network". In: *CoRR* abs/2112.08561 (2021). arXiv: `2112.08561`. URL: `https://arxiv.org/abs/2112.08561`.

[Zie+19] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. F. Christiano, and G. Irving. "Fine-Tuning Language Models from Human Preferences". In: *CoRR* abs/1909.08593 (2019). arXiv: `1909.08593`. URL: `http://arxiv.org/abs/1909.08593`.