

Master thesis on Sound and Music Computing
Universitat Pompeu Fabra

Automatic score-to-score music generation

Quốc Dương Nguyễn

Supervisor: Pedro Ramoneda

Co-Supervisor: Carlos Hernández Oliván

August 2023



Contents

1	Introduction	1
1.1	Motivation	1
1.2	Objectives	3
1.3	Structure of the Report	3
2	State of the art	4
2.1	Encodings	4
2.1.1	Note-level representation	4
2.1.2	Notation-level representation	7
2.2	Neural network architectures	10
2.2.1	Variational Autoencoder	11
2.2.2	Generative Adversarial Networks	11
2.2.3	Diffusion models	11
2.2.4	Transformers	12
3	Methods	14
3.1	Dataset	14
3.1.1	Data cleaning with the MuseScore metadata	15
3.1.2	Cleaning with music21	16
3.1.3	Similarity computation	17
3.2	Model	20
3.2.1	Transformers	20
3.2.2	Training	21

3.2.3	Generation	23
3.3	Evaluation	23
3.3.1	Objective evaluation	23
4	Results	25
4.1	Dataset	25
4.1.1	Difficulty classification	25
4.1.2	Dataset for score-to-score music generation	26
4.2	Training	26
4.3	Objective evaluation	27
4.4	Generation	27
5	Discussion	29
5.1	Available score data	29
5.2	Score token representation	29
5.3	Use of the dataset	30
5.4	Conclusion	30
	List of Figures	32
	Bibliography	33
A	Generated score	37

Acknowledgement

I would like to express my sincere gratitude to my supervisors, Pedro and Carlos, for their invaluable guidance and support throughout my master's thesis. Their deep expertise and encouragement helped me to complete this research and write this thesis. They steered me in the right direction whenever I needed it.

In addition, I extend my heartfelt appreciation to my parents and my sister, whose unconditional support has been a cornerstone throughout every phase of my life.

I would also like to extend my gratitude to my SMC master classmates, whose camaraderie and shared journey have significantly enriched my academic experience.

Last but certainly not least, I wish to express my deep appreciation for the wonderful friends I have had the privilege of making over the past year.

Thanks everyone,

Quốc Dương

Abstract

Music generation is the task of generating music using a model or algorithm. There are multiple ways of achieving this task as there are multiple types of data to represent music. Music generation can be audio-based or with symbolic music such as MIDI data. Approaches with symbolic music have been successful, especially using note-level representation such as the MIDI format. However, there is an absence of a baseline dataset tailored specifically for music scores generation using notation-level representations. In this thesis, we first construct a dataset specifically for the training and the evaluation of music generation models, then we build an automatic score-to-score generation model to generate scores. This research not only expands the horizons of music score generation but also establishes a solid foundation for future innovations in the field with a dataset made for score-to-score music generation.

Keywords: symbolic music representation; musical score generation; notation-level

Chapter 1

Introduction

Music is a universal language in human culture and expression that has been essential throughout history. Music composition is the process of creating original music through the succession of many musical elements such as melody, rhythm, timbre. Music generation, on the other hand, is the task of generating music from a model or algorithm. It is a growing research area with a strong emphasis on deep learning in recent years.

1.1 Motivation

Music generation has appeared with the first music generated by a computer in 1957 and its author, Newman Guttman, with a sound synthesis software called Music I developed at Bell Laboratories. During that same year, Lejaren A. Hiller and Leonard M. Isaacson made use of stochastic models, specifically Markov chains, making the first score composed by a computer [1]. Iannis Xenakis explored stochastic composition[2] in the context of algorithmic composition. The idea is to use a computer to quickly calculate various musical possibilities based on probabilities set by the composer. This generates sample pieces of music that can be chosen. Another approach using grammars and rules were used to steer the style of a piece. An example of that approach is Experiments in Musical Intelligence[3] (EMI) by David Cope. These techniques can be grouped in the field of Algorithmic Music Com-

position, a way of composing music using formal methods, such as mathematical instructions. This type of composing follows a controlled procedure and uses different techniques like Markov Models, Generative Grammars, Cellular Automata, etc. These techniques can be combined with Neural Networks to better model music, as seen in DeepBach[4].

Symbolic music is in some aspects very similar to text as they are both sequential and thus can be processed with NLP techniques. There has been a lot of interest in music generation in recent years[5, 6, 7]. However, research in that field is focused on MIDI generation, not score generation. Scores contain various musical symbols such as dynamic markings, slurs, etc. that are not present in the MIDI format. Scores are also made to be human readable.

Although some studies have used symbolic representations at a notation-level, such as in studies[8, 9, 10], there has been limited exploration of the use of complete musical scores containing diverse musical symbols and attributes with sequence models. Thus, it remains unclear whether sequence models can effectively generate comprehensive musical notations, and how musical scores can be represented in a manner that is suitable for feeding such models.

Automatic score-to-score music generation takes existing symbolic music score representations as input and creates new music scores.

The approach of the thesis is to take a more educational point of view, closer to the point of view of a music student or a music professor instead of a music listener. There are multiples possibilities of applications for score-to-score music generation, such as learning to play a score with different levels of difficulty, help artists in terms of inspiration as well as to make music composition more accessible to people.

Most importantly, scores are readable by musicians, people who play instruments can sight-read, learn and play off of music scores.

1.2 Objectives

The main objective of this thesis is to explore symbolic music generation, more specifically music score generation, focusing on the role of deep learning in this task and to do an overview of the most used encoding methods. We will build a dataset of scores that can be used to evaluate score music generation systems as there does not exist any dataset for music generation with a significant amount of data. Relevant representations at the notation-level are reviewed for the task of music score generation.

1.3 Structure of the Report

This thesis will be structured as follows. We will first delve into how symbolic music has been represented through the many encodings that have been created, then talk about the different architectures and techniques that have been developed in the last few years using deep learning. We will then show the methodology for the dataset creation using notation-level representation and the score-to-score generation model that we built.

Chapter 2

State of the art

In this chapter, relevant research works related to this thesis are reviewed. We will present the different representations of music information that we can divide into two categories, note-level representation (MIDI) that corresponds to a symbolic format to convey information mainly about pitch, duration, and notation-level representation that corresponds to a musical score. The different architectures and models used for the generation of symbolic music will be described.

2.1 Encodings

Deep learning models can extract significant features from how music is represented and the way music is encoded is significant as the quality of the generation. Recently, research in symbolic music generation has taken an interest in how important music is represented in different types of encodings[11].

2.1.1 Note-level representation

REMI

REMI, for Revamped MIDI, has been introduced in the Pop Music Transformer [12] paper. It is a tokenization that aimed to overcome the limitations of MIDI (Musical Instrument Digital Interface) such as the lack of support for pitch bends, vibrato,

etc.

The REMI representation uses:

- *Bar* events to indicate the beginning of a bar.
- *Position* events to point specific locations in a *Bar*.
- *Tempo* events for local tempo changes.
- *Chord* events to make harmonic structure more explicit.

This representation tries to represent MIDI data by following the way humans read them. The combination of *Bar* and *Position* events provides an explicit metrical grid to model the music. This allows for models to know that there is a hierarchical structure for a better rhythmic regularity, that is especially relevant in the case of Pop music, a genre that is dealt with in the paper[12].

They introduced the *Tempo* event for a higher level of freedom in rhythm expressiveness. It allows, as an example, the ability to do tempo rubato which means a slight speeding up and slowing down of the tempo of a piece.

Last but not least, the *Chord* events to make the harmonic structure explicit.

REMI+

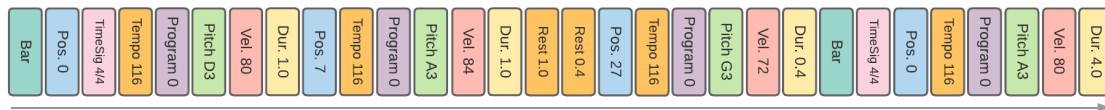
REMI+ is the extended REMI representation for multi-track and multi-signature music, and has been introduced in the FIGARO paper [13].

The REMI+ representation adds or modifies the following elements over REMI:

- Bar tokens
- Time signature
- Instrument information

- Order of events
- Quantization

Figure 1: Visual representation of REMI+ [11]



The *Bar* tokens now include the index of a bar, which is an additional piece of information to give more context to provide to the model.

The time signature is a new token that appears at the beginning of each bar, which is the case in written music.

For each note event, there is now the instrument information that is played as an additional token.

The order of events has been defined and sorted events by (Bar, Position, EventType, Instrument, Pitch) in ascending order. Using that order of events, the modelling task is easier.

Lastly, they use a 12 note onset positions per quarter note instead of 4 in the original REMI presentation. This allows triplet and sixteenth to be quantized accurately and to represent a wider range of music.

MIDI-Like

This tokenization simply converts MIDI events as tokens. It was introduced in [14] and was later used for the Music Transformer [15].

TSD

TSD (Time Shift Duration) is very similar to the MIDI-like representation. The difference is that TSD uses *Duration* tokens to represent the duration of a note instead of using *NoteOff* tokens to represent the end of a note being played.

Structured

The Structured representation was introduced by the Piano Inpainting Application, which is a generative model focused on inpainting piano performances. This representation is similar to TSD but is based on consistent token-type successions. The tokens are always represented in the same pattern of tokens of the following types: Pitch => Velocity => Duration => Timeshift.

CPWord

The CPWord representation [16] is similar to the REMI representation, but reduces the sequence length of the representation by using embedding pooling operations.

Octuple

The Octuple representation also uses embedding pooling but so that each pooled embedding represents one note. It reduces the length of a sequence, but also handles multitracks.

2.1.2 Notation-level representation

There have been various formats developed for symbolic score representations such as tree-structured representation like MusicXML [17] and MEI [18] and text representations such as the ABC notation, LilyPond and Humdrum. They were not designed with the use of machine learning applications in mind, but text representations can be handled as token sequences.

MusicXML

MusicXML is an XML-based format that is designed to be easy to read and write. It allows for a very detailed representation of music notation such as the pitch, rhythm, dynamics of notes, as well as the structure of a musical composition. It is widely used in the music industry and has been adopted by some of the most popular music notation software such as Sibelius or MuseScore.



Figure 3: Notes/pitches in the LilyPond notation

Humdrum (**kern)

Humdrum[20] is a software toolkit that contains programs for music analysis created by David Huron in 1994.

The ***kern* representation is the most popular in Humdrum. It can be used to represent basic or core information for common Western music and focuses on the function information conveyed by the score rather than its appearance. The notation is encoded vertically rather than horizontally.

LilyPond

LilyPond [21] is a free and open-source music engraving software that allows to create music score using text.

The output of the following text is Figure 3

```
\relative { g'1 e1 \bar "||" c2. c'4 \bar "|." }
```

Score Transformers

The Score Transformer[10] uses a score token representation that was designed to address the various visual elements that are present in a score. The proposed representation follows these principles:

- One token per symbol/attribute: A token corresponds to a score symbol or a note attribute
- Compatible with music21[22]: the use of the music21 toolkit is possible to create MusicXML scores from the score token representation

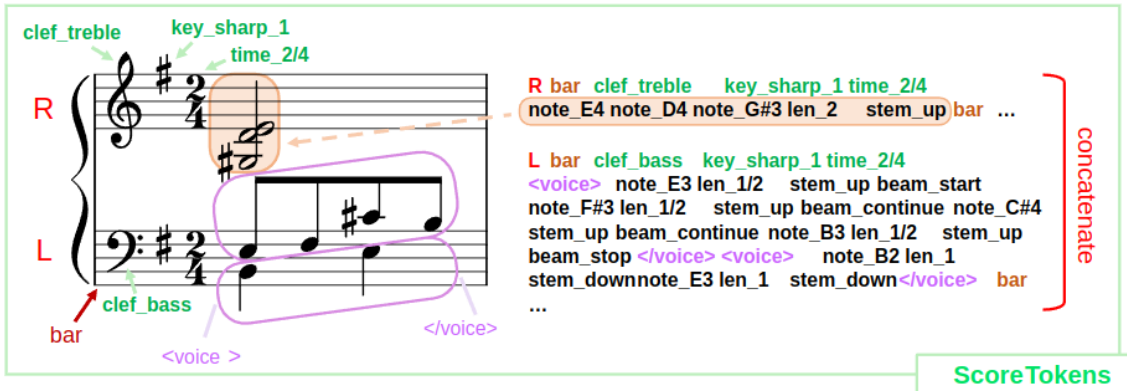


Figure 4: Score token representation [10]

- Concatenated sequences of staves: Sequences of staves are concatenated to form a single sequence which allows a model to refer to generated tokens while making inferences.
- Only essential musical symbols tokenized: it has some limitations, additional expression symbols such as articulations, dynamics, and ornaments are not handled

2.2 Neural network architectures

In this section, we explore the various factors that can influence the choice of deep learning architecture when applied to music generation tasks.

One crucial aspect to consider is the task-specific requirements. For instance, some music generation tasks may require handling polyphony, where multiple simultaneous voices or instruments need to be modeled and generated. Additionally, it may be essential for the chosen architecture to capture long-range dependencies in the music, such as recurring motifs or themes, and to accommodate variable-length sequences that are common in music compositions.

Another factor to consider is the availability and quality of training data. The amount of data available can influence the choice of model architecture. For example, some architectures might perform better with smaller datasets, while others require

big amounts of data to achieve satisfactory results. The choice of representation, such as MIDI, piano rolls, or other notation-level representations, can also have an impact on the compatibility and effectiveness of a particular architecture.

2.2.1 Variational Autoencoder

Variational Autoencoder[23] (VAE) is a type of generative model that has been successfully applied to various domains, including music generation. The VAE architecture consists of two main components: an encoder and a decoder, both of which are typically implemented as neural networks. The encoder learns to map input data (in this case, music) to a lower-dimensional latent space, while the decoder learns to reconstruct the original data from this latent representation. MusicVAE[24] is an implementation of the VAE developed by Google Magenta for music generation.

2.2.2 Generative Adversarial Networks

Generative Adversarial Networks[25] (GANs) are generative models that consist of two neural networks, a generator and a discriminator, that are trained simultaneously in a game-theoretic setting. The generator creates synthetic data, while the discriminator tries to distinguish between real and generated data. Over time, the generator becomes better at creating realistic data, and the discriminator becomes better at identifying the differences. MuseGAN[26] is an implementation of the GAN architecture for generating multi-track polyphonic music.

2.2.3 Diffusion models

Diffusion models[27], also known as denoising score matching or denoising diffusion probabilistic models, are a class of generative models that have recently gained attention. The main idea behind diffusion models is to simulate a continuous-time Markov process, where the data is gradually transformed from a simple noise distribution to the target distribution of interest (e.g., images or music). This is done by iteratively adding noise to the data and then denoising it, which corresponds to following the gradient of the log probability of the target distribution. One use case can

be found in the paper Symbolic Music Generation with Diffusion Models[28]. The approach is based on the idea of generating music by simulating a continuous-time Markov process, where noise is gradually added and removed.

2.2.4 Transformers

The Transformer architecture, introduced in the paper *Attention is All You Need*[29], is a highly expressive and powerful model that has achieved state-of-the-art performance in various natural language processing (NLP) tasks. The key innovation in the Transformer is the self-attention mechanism, which allows the model to weigh and focus on different parts of the input sequence, effectively capturing long-range dependencies and complex patterns. In the context of music generation, the Transformer architecture can be applied to generate music in a variety of formats, such as piano rolls, note sequences, or MIDI events. Due to its ability to handle long sequences and capture intricate patterns, the Transformer is well suited for music generation tasks.

We can differentiate two types of language modeling, such as masked language models and causal language models. Causal language models are frequently used for text generation. It predicts the next token in a sequence of tokens, and the model can only use previous tokens on the left. GPT-2[30] is an example of a causal language model. On the other hand, masked language models can use tokens bidirectionally. It is great for tasks that need good contextual understanding. *BERT*[31] is an example of a masked language model.

The Music Transformer[15] model is an adaptation of the Transformer for the task of music generation, and is a causal language model. It is equipped with relative attention and is very well suited for generative modeling of symbolic music with its ability to capture periodicity in various time scales.

The Multi-Track Music Machine[32] (MMM) is a generative system based on the Transformer architecture that is capable of generating multitrack music. This model creates a time-ordered sequence of events for each track and concatenates them into

one single sequence.

The MusicBERT[33] model is a masked language model. It is a discriminative model used for tasks such as music classification, recommendation, etc.

Chapter 3

Methods

In this thesis, we are using a causal decoder-only transformer model to generate scores. The decoder-only architecture simplifies the model and makes it more efficient for certain tasks, such as language modeling. By removing the encoder, transformers models can process input data more directly and generate output more quickly. At each step, for a given token, the attention layers can only access the words positioned before it in the sentence to predict the next token. These models are often called auto-regressive models.

The methods consists of the following steps:

1. The retrieval, cleaning and processing of the dataset
2. Building, training and evaluating our transformer model to generate scores by difficulty

The code for this thesis can be found on GitHub ¹.

3.1 Dataset

For our experiments, we are using a dataset of piano-only scores that we retrieve from `musescore.com`, the world's largest free sheet music catalog and sheet music-

¹<https://github.com/quoc-duong/score2score-music-generation>

sharing social platform. The dataset contains almost two million scores in the *mscz* format and anyone can upload on the `musescore.com` website. The dataset size contains around 147 GB of scores that have been uploaded up to 2019. As such, an extensive cleaning and filtering of the dataset must be done before being able to use it and to ensure the quality of the data.

3.1.1 Data cleaning with the MuseScore metadata

The dataset came with a metadata file in the *jsonl* format and contains some unstructured information for each music score such as:

- The description: it is a vague field filled by the author that sometimes contains the name of the song and the author along with other information in free form
- The title: Usually the name of the song and the author in free form
- The *instrumentsNames* field that contained a list of instruments
- The duration
- and many other fields that we don't need such as the time created, etc.

As such, we had to retrieve the files that contained only the piano instrument using the *instrumentsNames* field. I obtain over 335 000 files of piano-only files.

Conversion to MusicXML

The original dataset contains files in MuseScore format (**.mscz*). As such, we have to convert the files to the MusicXML format for us to be able to do a more refined search as well as to parse and tokenize the data. Because the MuseScore format is a proprietary format, the MuseScore program on command line must be used to run a batch job to convert the *.mscz* files into a usable *.musicxml* format. It was a tedious process as there is not much control through the command line program. However, the program can crash as soon as there is an internal error or exception. For example, the batch job can stop simply because the current file that is being

```

{
  "authorUserId": "6977801",
  "description": "Super easy arrangement of Coldplay's Viva La Vida in C: transposed (Semitone lower)",
  "instrumentsNames": [
    "Piano"
  ],
  "title": "Viva La Vida - Piano (Super Easy)",
  "url": "https://musescore.com/user/6977801/scores/3278571",
  "partsCount": "1",
  "revisionId": "5323486",
  "pagesCount": "2",
  "summary": "Easy Piano",
  "partsNames": [
    "Piano"
  ],
  "timeUpdated": "2021-03-10 13:13:55 UTC",
  "instrumentsCount": "1",
  "duration": "128",
  "timeCreated": "2017-01-22 16:10:19 UTC",
  "id": "3278571",
  "__error__": [],
  "musicxmlInstruments": []
}

```

Figure 5: Example of metadata for one community-made score

processed is corrupted and does not continue with the rest of the batch job. In order to get around this problem, the program error output was obtained, parsed to retrieve the string containing the files that made the program crash, then another batch job was automatically generated in order to rerun the process with that new batch job created ².

3.1.2 Cleaning with music21

As the given metadata file was not reliable enough and too vague, a deeper dive into the content of the scores themselves was needed to ensure that they contained piano scores. Once again, the dataset contains scores that anyone could upload. The files could contain strange or unexpected content, such as images, empty sheet music, etc.

Music21[22], a widely used Python-based toolkit for computer-aided musicology, had to be used to analyze the MusicXML files we obtained to ensure that they contained strictly 2 staves (Figure 6). I discarded files that were empty or corrupt. Finally, I discarded files that were too short, which means having fewer than ten bars. After

²<https://musescore.org/en/handbook/4/command-line-usage>



Figure 6: Example of a piano with two staves

this process we obtain a dataset of over 225 000 files.

3.1.3 Similarity computation

As (another) reminder, the dataset contains files that anyone can upload on the internet. The dataset contains files that are identical, variations of the same songs musically speaking, files that contain the same song but with more or less the same number of verse/chorus, or slight variations in display and arrangements; they could also be transposed, etc. The list goes on.

Computing the similarity of each possible pair of songs to see if they are similar amounts to $n(n-1)/2$ possible pairs which would amount to billions of comparisons. In order to avoid that complexity and to save a tremendous amount of time, we had to find another way to retrieve similar files.

MinHashing

In order to remove files that are "too similar" or identical, I used MinHash, which is a technique for estimating the similarity between two sets. It was first introduced in information retrieval to evaluate the similarity between documents quickly. The basic idea is to hash the elements of the sets and then take the minimum hash value as a representation of the set. Because the minimum value is used, the technique is called MinHash.

To reduce the computations, the pitches of each sheet music file of the right hand only were retrieved. There is a list of midi pitches for each file in the dataset. When

retrieving the pitch, if a *Chord* object in music21 is encountered, it is flattened, meaning that we get the pitches from highest to lowest and add them to the list. We obtain a list of similar scores for each of the scores which considerably reduces the number of computations required for the next step.

Matching a sequence and similar ones

For each of the scores, we obtain a number of similar scores. One solution to that is an algorithm that finds the longest contiguous matching sub-sequence (LCS) that contains no “junk” elements. It does not yield minimal edit sequences, but does tend to yield matches that "look right". We add some weight to the length difference of each pair of scores so that scores that are closer in length, have a higher similarity score and vice-versa.

Using a fixed threshold of similarity score, I am able to remove files that identical and/or similar in melody. We remove files that are shorter first to speed up the processing. As a result, we obtain around 146 808 piano scores in MusicXML format.

As a last step to further prevent scores that are problematic or too small from appearing in the final dataset as well as to reduce the complexity of the dataset, scores that contain at least 200 notes in the right hand and at most 300 notes on the right hand are kept.

Difficulty classification

The goal of classifying scores by difficulty is to be able to do conditional generation by difficulty (Figure 7). Using a piano difficulty classifier[34], we can obtain the difficulty of each pieces, with a float value ranged from 0 to 9. Intuitively speaking, we can reduce the complexity and the size of our dataset by only using a subset of a low range of difficulty values from the output of the piano difficulty classifier. For that reason, the difficulty of every piece is rounded up to the nearest integer and retrieved files that are ranged in difficulty from 1 to 4.

We have now **4528** piano scores classified in difficulty in four different levels of



Figure 7: An easy and a harder score side by side

difficulty.

Tokenization

Tokenization is way of separating a piece of text into smaller units called tokens. A token is a distinct element, part of a sequence of tokens. In natural language, a token can be a character, a sub-word or a word. A sentence can then be tokenized into a sequence of tokens representing the words and punctuation. For symbolic music, tokens can represent the values of the note attributes (pitch, velocity, duration) or time events. These are the fundamental tokens, that can be compared to the characters in natural language.

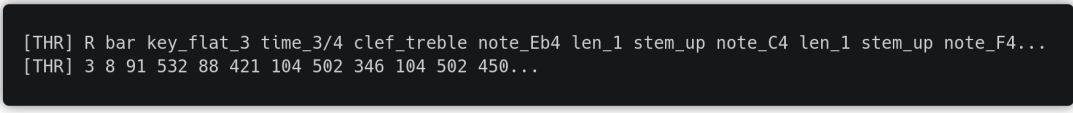
When training the model, there are computational limits in memory. As a consequence, I had to split every piano score into chunks of 4 bars using music21. Each chunk of four bars are now individual and independent scores in the *.musicxml* format.

I am using the Score Transformers[10] intermediate representation to convert the MusicXML files into a format that the model that we are going to use can interpret and train on.

For each of the files that we have, we convert them to the Score Transformers representation. They are stored in text files.

To simplify the training process, we are using index mapping to convert a score token to their associated index.

Knowing that each file have an associated difficulty, we have to create a token for each difficulty level: *[ONE]*, *[TWO]*, *[THR]*, *[FOU]*, respectively for the first, second,



```
[THR] R bar key_flat_3 time_3/4 clef_treble note_Eb4 len_1 stem_up note_C4 len_1 stem_up note_F4...
[THR] 3 8 91 532 88 421 104 502 346 104 502 450...
```

Figure 8: Input sequence of difficulty 3: score token representation and mapping

third and fourth difficulty, from easiest to hardest. We add the difficulty token to the beginning of each text files for each scores. As a result, our model will capture difficulty of piano pieces and the model will be able to conditional generation by steering it towards a certain difficulty given a starting input.

Byte-Pair Encoding (BPE) BPE[35] is a data compression algorithm used in natural language processing and information retrieval. It is primarily used for tokenization and subword encoding of text data. BPE is a statistical algorithm that aims to find the most frequent subword units in a given corpus of text. The basic idea behind BPE is to treat individual characters as the initial vocabulary and iteratively merge the most frequently occurring pairs of symbols to create new, longer subword units. This process continues until a predefined vocabulary size or a desired number of merge operations is reached.

In the case of symbolic music, BPE shows improvements of the performance of Transformers [36].

We use BPE to train a tokenizer using the vocabulary contained in our dataset. The tokenizer will take into account our new difficulty tokens that are treated as special tokens.

3.2 Model

3.2.1 Transformers

A decoder-only Transformer model (GPT-like) is used to train on our dataset and is coded in PyTorch.

The configuration of the decoder layers involve the following aspects:

- The dimension parameter, set to 512, establishes the dimensionality of the hidden states within the model. This dimension impacts the model’s ability to capture intricate patterns and representations within the data.
- The depth, set to 12, dictates the number of decoder layers stacked on top of each other. A greater depth allows the model to capture more complex relationships in the data, but this also demands more computational resources.
- The heads parameter, set to 8, indicates the count of attention heads employed within each decoder layer. Attention heads facilitate the model in focusing on different parts of the input sequence concurrently, enabling the understanding of various inter-dependencies.
- There is some dropout used, which corresponds to a dropout mechanism applied to the input token embeddings. Dropout is a regularization technique utilized during training, involving the random deactivation of certain neural network units to prevent overfitting.

Flash Attention

The model makes use of Flash Attention[37] for memory efficiency but most importantly to speed up training.

This approach involves dividing the attention matrix into smaller sections, allowing for efficient computation of the softmax and exponentiated weighted sums. By recalculating these values in a tiled manner during the backward pass, the memory requirement remains linear regardless of the length of the sequence. As a result, recent models can now incorporate longer context lengths without being hindered by memory limitations.

3.2.2 Training

A preliminary model was trained with music scores that were split in multiple files of four bars of music. The sequence length was small, at around 512. The model was

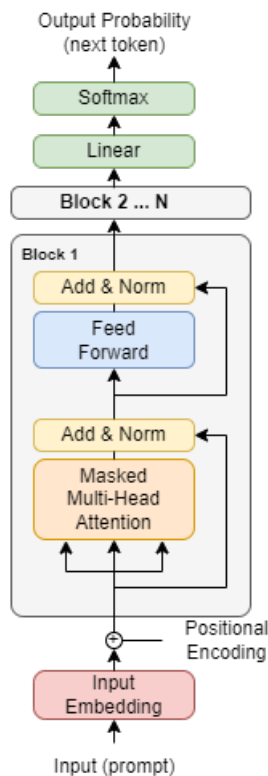


Figure 9: Overview of the decoder-only Transformer model

able to generate a sequence that followed the rules of the score token representation and generate new notes and bars of score. However, we could see very quickly that the notes did not make compared to the input given and did not learn any musical structure, phrasing, rhythm or melody. For that reason, we chose to increase the maximum sequence length.

The actual model is trained on a single NVIDIA Tesla V100 (16 GB). As a result, and with the significant improvement in memory efficiency that comes from flash attention, the maximum sequence length is set at 2048 and the batch size is set at 4. The learning rate is set at $6e-5$ and it is reduced with a learning rate scheduler when it reaches a plateau with a patience of 2 epochs. The data loader is written in such a way that if an input sequence is larger than the maximum sequence length, we are randomly selecting a sub-sequence of the maximum sequence length. As for other input sequences, we zero-pad them on the left.

3.2.3 Generation

In order to obtain better score outputs, we are using sampling, more specifically Top-p sampling [38]. Top-p sampling chooses from the smallest possible set of tokens whose cumulative probability exceeds the probability p . Top-p sampling allows for more control over the diversity of generation. By adjusting the value of "p," you can influence the randomness of the output.

Temperature is another parameter used to control the diversity of the generation. A higher temperature increases the entropy of the distribution, making the probabilities more evenly spread out and leading to more random and diverse outputs. Conversely, a lower temperature narrows down the distribution, making the model more confident and deterministic in its predictions.

Both top-p sampling and temperature can be adjusted based on the desired balance between randomness and control in text generation.

3.3 Evaluation

3.3.1 Objective evaluation

In order to objectively evaluate the performance of our music score generation system, there are two important metrics that come into play: the cross-entropy loss and the perplexity. The cross-entropy loss serves as a guiding objective during the training phase of our Transformer model. It quantifies the dissimilarity between the predicted probability distribution of tokens and the ground truth distribution. Formally, the loss for a single token is computed as the negative sum of the element-wise multiplication between the true and predicted distributions' logarithmic values. This token-level loss is then averaged across all tokens in a sequence and across the entire batch, providing a quantitative measure of the model's training progress and convergence towards accurate token predictions.

$$L_i = - \sum_{j=1}^V y_{i,j} \log(p_{i,j}) \quad (3.1)$$

Where:

L_i = the cross-entropy loss for the i th token

$y_{i,j}$ = the true probability distribution for the j th token in the i th sequence

$p_{i,j}$ = the predicted probability distribution for the j th token in the i th sequence

V = the vocabulary size.

The other metric that we compute is the perplexity, which is a statistical measure of how confidently a language model predicts a text sample. In other words, it quantifies how *surprised* the model is when it sees new data. It is one of the most common metrics for evaluating language models. It is defined as the exponentiated average negative log-likelihood of a sequence, calculated with exponent base e. The lower the perplexity, the better the model predicts the text.

$$PPL = \exp \left(\frac{1}{N \cdot T} \sum_{i=1}^N \sum_{j=1}^T L_i \right) \quad (3.2)$$

Where:

PPL = the perplexity

N = the batch size

T = the sequence length

L_i = the cross-entropy loss for the i th sequence.

Chapter 4

Results

4.1 Dataset

The dataset is composed of 146 808 pieces that are for piano, contains strictly 2 staves, in which we removed similar scores, empty scores and corrupt scores. In order to build a first score-to-score model, we choose to use a smaller subset of the dataset to reduce the complexity and most importantly quicker training.

4.1.1 Difficulty classification

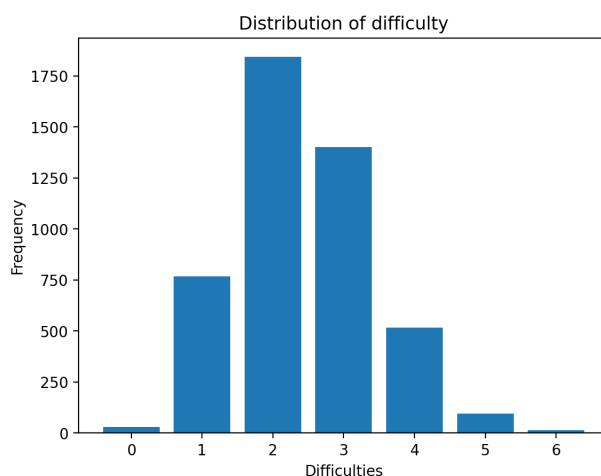


Figure 10: Difficulty distribution

We are limiting the model to do conditional generation on 4 different levels of diffi-

culty (Figure 10). The subset used contains 4528 scores of difficulty 1 to 4, respectively [ONE], [TWO], [THREE], [FOUR] as tokens in the input sequence.

4.1.2 Dataset for score-to-score music generation

After an extensive cleaning of the data as well conversion to MusicXML, score similarity computation with MinHashing and difficulty classification and tokenization to the score token representation, we obtain our final dataset.

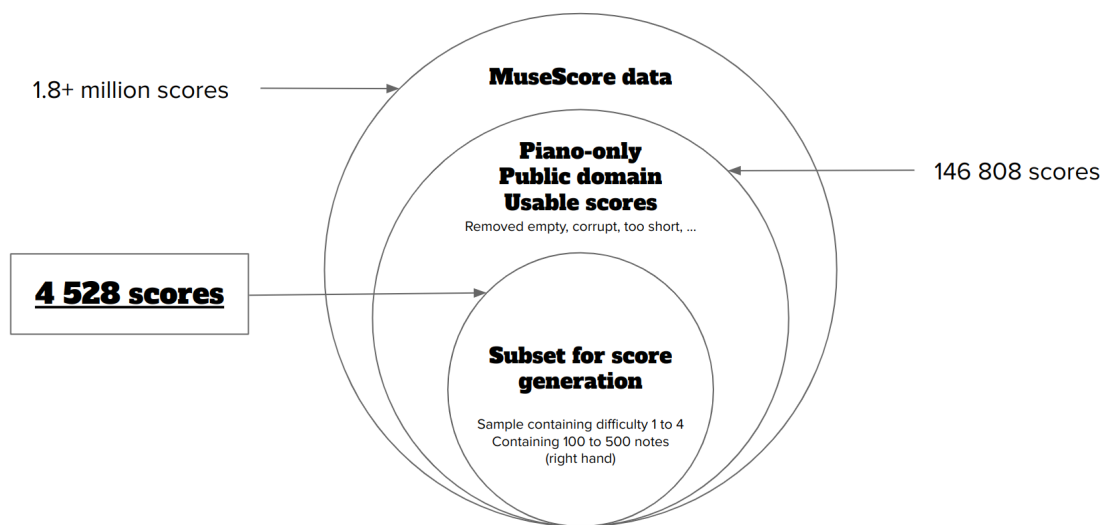


Figure 11: Breakdown of the dataset

Given the significant size of the usable dataset, comprising of 145 808 scores, a subset of it is chosen for our experiments and is large enough to ensure representation across difficulties and musical elements but most importantly to reduce computational complexity and resource requirements during the score generation process. That subset contains 4528 music scores.

4.2 Training

The dataset we used is composed of over 4500 scores. 80% of it was used for training and the rest was split for validation and test. The model was trained for over 34 000 steps.

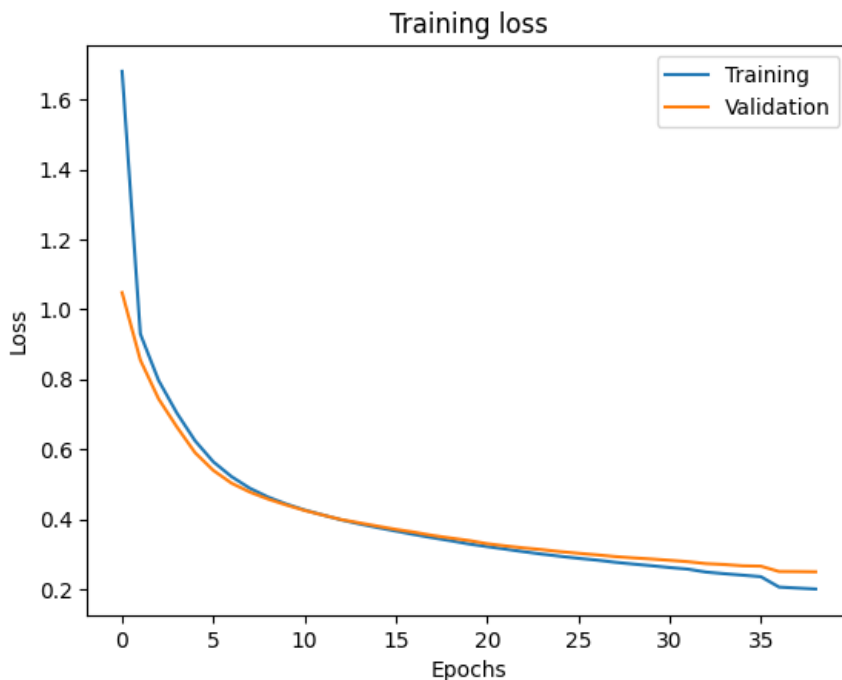


Figure 12: Training progression

4.3 Objective evaluation

On the test set, the model gives the following metrics:

- Perplexity (PPL): 1.2859
- Loss: 0.2515

4.4 Generation

The model successfully generates new bars of music and successfully captures the rules of the score token representation. We can then convert that representation back to the MusicXML format with the de-tokenizer and open it with a score visualizer software to see the results.

The model is able to generate notes for the left hand piano (See appendix A). The generated notes are seen right towards the end, where the right hand is silent. The model was able to understand and capture the left hand rhythm that appears in the

input sequence, as well as the notes that were played. It is able to capture some patterns of the input successfully.

Chapter 5

Discussion

5.1 Available score data

The `musescore.com` website is a place where people can upload any of their creation. As a result, the dataset contains files of varying quality, thus requiring a substantial cleaning of the data.

The dataset contains unstructured data and is not organized by genre, instruments, or artists. It is a limitation that comes from the fact that most of the data are user-created and do not necessarily contain all metadata. The possibilities of score-to-score music generation are hindered as a result. There is an effort to be made to annotate the dataset, or to make available this metadata.

5.2 Score token representation

Score-to-score generation is an area of research that remains mostly unexplored and there are not many ways to represent musical notation effectively. The score token representation [10] is a first iteration to symbolize score elements. However, it is the only representation designed to feed sequence models. Other representations follow a tree-structure such as MusicXML[17] and MEI (Music Encoding Initiative) [18] and other follow a text-like representation such as the ABC notation [19], Humdrum

[20] and Lilypond [21] but are not designed with sequence models in mind. There are multiple ways to improve the representation, such as the

5.3 Use of the dataset

As stated in the thesis, we are using a much smaller subset of the data to reduce the complexity of the processing and to reduce the training time to get satisfactory results. With more time and resources, the dataset can be used in its entirety to capture different patterns in music, rhythms, melodies, etc.

In the field of music score generation, a domain that has thus far received limited attention in research, the absence of a comprehensive and benchmark-worthy dataset has been a glaring limitation. This research endeavor seeks to bridge this gap by introducing a novel and expansive dataset tailored specifically for music score generation tasks. In doing so, we aim to catalyze advancements in the field by providing researchers with a dataset from which they can develop, fine-tune, and evaluate their models. This contribution not only addresses the current scarcity of suitable datasets, but also strives to promote the principles of open science. By sharing this resource, we seek to accelerate research in music score generation and facilitate comparative evaluations, thus fostering a stronger foundation for future research in this field. The dataset is available here ¹ and the smaller version of it ².

5.4 Conclusion

This master's thesis has explored the field of music score generation, specifically focusing on the development and evaluation of score-to-score music generation models employing notation-level representations. It led to the creation of a foundational baseline dataset specially tailored for score music generation utilizing notation-level representation. This dataset serves as an open science resource, inviting researchers to employ it as a valuable asset in their explorations of music score generation.

¹<https://doi.org/10.5281/zenodo.8304135>

²<https://doi.org/10.5281/zenodo.8304162>

Through the construction of this dataset, this work not only addresses a notable gap in the current landscape of music generation research, but also establishes a tangible resource that promotes collaboration, reproducibility, and innovation. Future researchers can confidently assess and compare their advancements in the domain of notation-based music generation.

Furthermore, this thesis demonstrates the feasibility of score token representation in music generation. While the generated scores may not yet achieve perfection, they exhibit the potential to capture significant musical patterns from the input data. This successful incorporation of the score token representation reaffirms its role as a viable avenue for encoding musical nuances and enriching the repertoire of music generation techniques.

This research not only contributes to the academic understanding of music generation but also underscores the importance of open science principles. As the field of music score generation advances, the groundwork laid by this thesis will be used for researchers to build upon, leading to more sophisticated and expressive music generation systems in the future.

List of Figures

1	Visual representation of REMI+ [11]	6
2	Notes/pitches using the ABC notation	8
3	Notes/pitches in the LilyPond notation	9
4	Score token representation [10]	10
5	Example of metadata for one community-made score	16
6	Example of a piano with two staves	17
7	An easy and a harder score side by side	19
8	Input sequence of difficulty 3: score token representation and mapping	20
9	Overview of the decoder-only Transformer model	22
10	Difficulty distribution	25
11	Breakdown of the dataset	26
12	Training progression	27

Bibliography

- [1] Hiller, L. A. & Isaacson, L. M. *Experimental Music; Composition with an electronic computer* (Greenwood Publishing Group Inc., 1979).
- [2] Xenakis, I. *Formalized music: thought and mathematics in composition*. 6 (Pendragon Press, 1992).
- [3] Cope, D. Experiments in musical intelligence (EMI): Non-linear linguistic-based composition. *Journal of New Music Research* **18**, 117–139 (1989).
- [4] Hadjeres, G., Pachet, F. & Nielsen, F. DeepBach: a Steerable Model for Bach Chorales Generation. In Precup, D. & Teh, Y. W. (eds.) *Proceedings of the 34th International Conference on Machine Learning*, vol. 70 of *Proceedings of Machine Learning Research*, 1362–1371 (PMLR, 2017). URL <https://proceedings.mlr.press/v70/hadjeres17a.html>.
- [5] Briot, J.-P., Hadjeres, G. & Pachet, F.-D. Deep Learning Techniques for Music Generation – A Survey (2017).
- [6] Ji, S., Luo, J. & Yang, X. A comprehensive survey on deep music generation: Multi-level representations, algorithms, evaluations, and future directions. *arXiv preprint arXiv:2011.06801* (2020).
- [7] Hernandez-Olivan, C. & Beltran, J. R. Music Composition with Deep Learning: A Review (2021).
- [8] Carvalho, R. G. C. & Smaragdis, P. Towards end-to-end polyphonic music transcription: Transforming music audio directly to a score. In *2017 IEEE*

- Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 151–155 (IEEE, 2017).
- [9] Liu, L., Morfi, V. & Benetos, E. Joint Multi-Pitch Detection and Score Transcription for Polyphonic Piano Music. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 281–285 (2021).
- [10] Suzuki, M. Score Transformer: Generating Musical Score from Note-level Representation. In *ACM Multimedia Asia*, 1–7 (ACM, New York, NY, USA, 2021).
- [11] Fradet, N., Briot, J.-P., Chhel, F., El Fallah-Seghrouchni, A. & Gutowski, N. MidiTok: A Python package for MIDI file tokenization. In *22nd International Society for Music Information Retrieval Conference* (2021).
- [12] Huang, Y.-S. & Yang, Y.-H. Pop Music Transformer: Beat-based Modeling and Generation of Expressive Pop Piano Compositions (2020).
- [13] von Rütte, D., Biggio, L., Kilcher, Y. & Hofmann, T. FIGARO: Generating Symbolic Music with Fine-Grained Artistic Control (2022).
- [14] Oore, S., Simon, I., Dieleman, S., Eck, D. & Simonyan, K. This time with feeling: Learning expressive musical performance. *Neural Computing and Applications* **32**, 955–967 (2020).
- [15] Huang, C.-Z. A. *et al.* Music transformer. *arXiv preprint arXiv:1809.04281* (2018).
- [16] Hsiao, W.-Y., Liu, J.-Y., Yeh, Y.-C. & Yang, Y.-H. Compound word transformer: Learning to compose full-song music over dynamic directed hypergraphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 178–186 (2021).
- [17] Good, M. & Actor, G. Using MusicXML for file interchange. In *Proceedings Third International Conference on WEB Delivering of Music*, 153 (IEEE, 2003).

- [18] Hankinson, A., Roland, P. & Fujinaga, I. The Music Encoding Initiative as a Document-Encoding Framework. In *ISMIR*, 293–298 (2011).
- [19] Walshaw, C. The ABC music standard 2.1 (2011).
- [20] Huron, D. B. *The humdrum toolkit: Reference manual* (Center for Computer Assisted Research in the Humanities, 1994).
- [21] Nienhuys, H.-W. & Nieuwenhuizen, J. LilyPond, a system for automated music engraving. In *Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003)*, vol. 1, 167–171 (Citeseer, 2003).
- [22] Cuthbert, M. S. & Ariza, C. music21: A toolkit for computer-aided musicology and symbolic music data (2010).
- [23] Kingma, D. P. & Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).
- [24] Roberts, A., Engel, J., Raffel, C., Hawthorne, C. & Eck, D. A hierarchical latent vector model for learning long-term structure in music. In *International conference on machine learning*, 4364–4373 (PMLR, 2018).
- [25] Goodfellow, I. *et al.* Generative adversarial networks. *Communications of the ACM* **63**, 139–144 (2020).
- [26] Dong, H.-W., Hsiao, W.-Y., Yang, L.-C. & Yang, Y.-H. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32 (2018).
- [27] Ho, J., Jain, A. & Abbeel, P. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems* **33**, 6840–6851 (2020).
- [28] Mittal, G., Engel, J., Hawthorne, C. & Simon, I. Symbolic music generation with diffusion models. *arXiv preprint arXiv:2103.16091* (2021).
- [29] Vaswani, A. *et al.* Attention is all you need. *Advances in neural information processing systems* **30** (2017).

- [30] Radford, A. *et al.* Language models are unsupervised multitask learners. *OpenAI blog* **1**, 9 (2019).
- [31] Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
- [32] Ens, J. & Pasquier, P. MMM : Exploring Conditional Multi-Track Music Generation with the Transformer (2020).
- [33] Zeng, M. *et al.* Musicbert: Symbolic music understanding with large-scale pre-training. *arXiv preprint arXiv:2106.05630* (2021).
- [34] Ramoneda, P. *et al.* Combining piano performance dimensions for score difficulty classification. *arXiv preprint arXiv:2306.08480* (2023).
- [35] Sennrich, R., Haddow, B. & Birch, A. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909* (2015).
- [36] Fradet, N., Briot, J.-P., Chhel, F., Seghrouchni, A. E. F. & Gutowski, N. Byte Pair Encoding for Symbolic Music. *arXiv preprint arXiv:2301.11975* (2023).
- [37] Dao, T., Fu, D., Ermon, S., Rudra, A. & Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems* **35**, 16344–16359 (2022).
- [38] Holtzman, A., Buys, J., Du, L., Forbes, M. & Choi, Y. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751* (2019).

Appendix A

Generated score

2

0

The first system of music consists of four measures. The treble clef staff begins with a quarter note G4, followed by an eighth note A4, a quarter note B4, and a quarter note C5. The bass clef staff starts with a half note chord of G2 and B2, followed by a half note chord of A2 and C3. In the second measure, the treble clef has a half note G4, and the bass clef has a half note G2. The third measure features a quarter note G4 in the treble and a quarter note G2 in the bass. The fourth measure has a quarter note A4 in the treble and a quarter note A2 in the bass. The final two notes of the bass line in the fourth measure are beamed together.

0

The second system of music consists of two measures. The treble clef staff is empty in both measures. The bass clef staff begins with a half note chord of G2 and B2, followed by a half note chord of A2 and C3. In the second measure, the bass clef has a half note chord of G2 and B2. The final two notes of the bass line in the second measure are beamed together.