INFO8010 - Final Project Report
Generating Music with Transformers

Fanny Bodart,[1] **Julien Brandoit**,[2] and **Tom Clara**[3]
[1] *fanny.bodart@student.uliege.be (s202453)*
[2] *julien.brandoit@student.uliege.be (s200710)*
[3] *tom.clara@student.uliege.be (s200770)*

In this paper, we explore the application of deep learning techniques, specifically transformer-based architectures, for symbolic music generation. We formulate this task as a sequence completion problem. We introduce our methodology, which includes an in-depth discussion of data preprocessing, tokenization strategies, and the design of our transformer models. We compare *state-based* and *event-based* tokenization approaches, highlighting their impacts on model performance and output quality. Our experiments demonstrate that *event-based* tokenization significantly improves the coherence and complexity of generated music. We also address the challenges of evaluating music generation, emphasizing the subjective nature of musical quality and the limitations of current metrics. Furthermore, we discuss the ethical and legal implications of using copyrighted material for training AI models. A series of generated productions publicly accessible on our `GitHub`[a] illustrate our results and confirm that it is possible to generate music using a transformer architecture. To listen easily to some interesting results without delving into the GitHub repository, this web page[b] can be consulted.

I. INTRODUCTION

Music, an art form deeply rooted in human culture, has long been a subject of fascination and study in various domains. With the advancements in artificial intelligence and deep learning, researchers have increasingly explored the intersection of technology and music creation. In this paper, we delve into the realm of symbolic music generation using transformer-based architectures, leveraging the power of AI to create musical compositions.

To make our project memorable, we named it MOzART (Musical Orchestra z Artificial Rhapsody Transformer). MOzART embodies our desire to combine the sophistication of classical music with the innovative capabilities of artificial intelligence. Our aim is to explore the potential of transformer architectures to generate music that is not only coherent but also, hopefully, enjoyable to listen to and creative.

The transformer architecture, introduced by Vaswani et al. [1], has revolutionized natural language processing tasks by effectively capturing long-range dependencies in sequential data through the use of attention mechanisms. Inspired by their success in text generation, we adapt and apply transformer models to the domain of symbolic music generation through a paradigm of successive completion of an initial prompt. Our methodology encompasses a comprehensive exploration of data preprocessing, tokenization strategies, probabilistic models, and network architectures tailored to the unique characteristics of musical data.

Through our experiments, we investigate the efficacy of different tokenization approaches, namely *state-based* and *event-based* tokenization, in capturing the complexity and coherence of musical compositions. Additionally, we address the challenges inherent in evaluating the quality of AI-generated music, emphasizing the subjective nature of musical perception and the limitations of existing metrics.

Furthermore, we delve into the ethical and legal considerations surrounding the use of copyrighted material for training AI models and the implications for the future of music creation. Our project culminates in a series of music productions publicly accessible on our GitHub repository, showcasing the potential of transformer architectures in generating music.

The remainder of this article is organised as follows: section II summarizes the different approaches that already exist to solve similar problems, section III explains thoroughly the processing of the data and the models, section IV shows the main results (including audio demonstrations) and finally in section V, the results are discussed and further improvements are suggested.

This work does not lead to state-of-the-art results, but is meant to highlight the importance of key elements such as the tokenization, the probabilistic model or the transformer architecture itself. Most of the observations and conclusions drawn in this work are already present in the literature. We consider reproducing them from scratch as a pedagogically meaningful task.

II. RELATED WORK

Several approaches to generate music with deep learning exist in the literature. Some of the oldest attempts consist in using recurrent neural networks or long short term memory networks (the latter being an improvement of the former) [2]. Even if they are no longer state-of-the-art techniques, these methods try to take advantage of the sequential structure of music. More recent methods are often based on the transformer architecture. In the sequel, this type of architecture is briefly introduced and some previous attempts of using transformers to generate music are described.

The transformer architecture [1] is meant to handle sequential data and relies extensively on the *attention* mechanism. This mechanism, implemented in *attention layers*, allows to capture the dependencies between elements of a sequence. Attention layers are a powerful tool to handle sequences in which complex and potentially long-range patterns are intertwined. Using these layers in parallel also makes it possible to combine several different semantic learnings, with each layer focusing on a particular semantic concept. The main contribution of [1] is to

[a] `https://github.com/julienbrandoit/INFO8010---MOZART---Generating-Music-with-Transformers.git`
[b] `https://tomclara.com`

use only these powerful attention layers, but extensively. The resulting architecture, the transformer architecture, is free of convolutional networks and recurrent networks. It overcomes all the previous sequence-to-sequence models in terms of accuracy, training time and flexibility.

The use of transformers for symbolic music completion is known to be successful. State-of-the-art transformer models are now able to generate several minutes of consistent music with rich patterns [3]. Some of the generated tracks may even overcome real music in terms of human-assessed quality [4].

One of the significant improvements of the most recent models is the use of a *relative* attention mechanism. As explained by [5], one of the key for the transformer model to yield satisfactory results is the positional embedding. Indeed, the transformer architecture is insensitive to the order of the tokens in the input sequence and it is therefore necessary to embed some positional information to transform set of tokens into an ordered sequence of tokens. A well-known solution is to add to each token its absolute position embedding. A drawback of this approach, highlighted by [5], is the use of absolute positions, which are not completely relevant for most of the applications. Indeed, in most of the sequences fed to transformer networks (text or music, for instance), the relative positions ("token A is 3 positions before token B") are more relevant than the absolute ones ("token A is at position 235, and token B at position 238"). Hence the concept of relative attention has been developed in [5]. It improves significantly the accuracy of the transformer model for text translation tasks. In [3], the relative attention mechanism is shown to be effective for musical sequences. The authors generated several minutes of music using a memory-efficient implementation of relative self-attention.

Beyond model improvements, several articles [4, 6] found in the literature highlight challenges inherent to symbolic music generation. First, there is a lack of meaningful metric to assess the quality of a generated musical sequence. The only way of going beyond the loss values is to ask people to evaluate different tracks, which remains of course highly subjective and time-consuming. Secondly, the symbolic representation of music is not trivial. Some of the subtle musical aspects are extremely difficult to translate into a rigid serialization format such as MIDI. More importantly, the tokenization of music is a crucial step.

A possible approach is to discretize the time in several time steps, and to associate to each time step a token that represents the state of the music at this time. This approach is called *State-based tokenization* in the sequel. Depending on what is encoded in a state, the number of different tokens can vary considerably. In addition, the size of the sequences of tokens crucially relies on the sampling frequency used for the discretization. The authors of [6] report that a sampling period of 10 ms is required (a larger sampling period leads to perceivable distortions). Therefore, generating long tracks of music with strong internal consistency is challenging, because the number of tokens to handle often becomes prohibitive.

Another approach, which is more successful according to some articles [4, 6], is to associate a token to a *change* in state, rather than to a time step. This often results in a smaller vocabulary and more compact sequences, which allows to use longer contexts (*i.e.*, longer input prompts) and to generate longer consistent music tracks. In addition, this approach makes the MIDI-to-tokens translation simpler and more efficient, because it relies strongly on the internal structure of the MIDI files. In the sequel, this approach is called *Event-based tokenization.*

To conclude this section, it is worth mentioning other recent and successful approaches that are not based on the transformer architecture. One possibility consists in using a variational autoencoder [7, 8]. The idea is to build an internal representation of a music sequence and then to reuse it through a decoder to sample new sequences. In [7], a mechanism considers previous notes to create a consistent track, highlighting the need to filter out "wrong" sequences using heuristics.

III. METHODS

This section focuses on the different methods, regarding the processing of the data, the tokenization, the probabilistic model and the network architecture, that have been implemented. Each aspect is treated independently from the others. In Section IV, the different combinations of methods that have been tried are enumerated.

A. Data

The use of MIDI files is convenient, because they do not contain audio information in a waveform format which can be particularly challenging to use and discretize when training deep learning model. Instead, they use a sequence of MIDI events to communicate which of the 128 possible notes should be played, when, how long, how loud, and finally, with which instrument.

They are numerous type of MIDI events [9], but the most relevant ones in this work are the `note_on/off` events. They specify the pitch of the played note (*i.e.*, which note amongst the 128 possible ones is played), and the channel in which it must be played (there are 16 channels, each assignable to a different instrument).

Importantly, in the header of the MIDI file, the resolution (in 'ticks per beat') is indicated. It describes the smallest time interval that can separates two events.

To train and test our models, we initially used a directory of MIDI files containing only classical music, publicly available from the Classical Archives website [10]. Subsequently, to broaden our models to all types of music, we scraped MIDI files from the midiworld website [11]. These files consist of various diverse musical styles.

1. *Classical Music*

This dataset comprises 4572 MIDI files sourced from multiple classical composers (including Mozart, Bach, Chopin, and others). These musical compositions feature a variety of instruments, yet they all share the common characteristic of a classical style.

2. *MIDIWORLD scraping*

This dataset is composed of 5025 MIDI files from diverse musical styles, spanning from pop and rock to Disney themes and national anthems. We are aware that this dataset contains music protected by copyright, and that scraping does not contribute to artists' remuneration. However, we chose to use it, as our goal is purely academic.

3. *The relevance of using two different datasets*

Fig. 1 shows that the channels (instruments) on which the music is played vary significantly from one dataset to another, as do the different pitches played. Utilizing two different datasets allows us to evaluate the generalization ability of our models across various musical styles, and to compare different music generated from the same prompt by models trained on different styles. We also note that the distribution of notes played is quite different between the two datasets, which could indicate a fundamental difference between classical music and any other style.

4. *Data Augmentation*

Data augmentation in the case of music generation is comparable to that of text generation. It is difficult to alter a text without changing the meaning of its sentences. Similarly, music is a complex auditory harmony that is easily disrupted by even the slightest change in a single note or its pitch. Hence, we only perform data augmentation when utilizing the REMI tokenizer from MidiTok [12]. This library provides the ability to intelligently augment the dataset by performing transformations on the velocity of the notes, their duration, or even their pitches. These transformations consist in applying offsets (regarding the velocity, the duration or the pitch) in such a way that the music remains harmonious (for instance, by shifting pitches by octaves).

B. Preprocessing and tokenization

1. *Preprocessing of the MIDI files*

A regular MIDI file has 16 channels, *i.e.*, there are potentially 16 different instruments that play simultaneously. As a first step, these 16 channels are split into 16 independent "songs", because we focus (mainly) on single-instrument music generation. Most of these newly created songs are empty or nearly empty, because many instruments often have a minor role to play in a multi-instrument piece of music. Therefore, the channels in which there is more than 70% of silence are removed from the dataset. Some corrupted MIDI files are also discarded. To tackle multi-instrument music generation, the preprocessing is essentially the same, but the 16 channels are not split and must be tokenized together.

2. *State-based tokenization*

A first possible tokenization consists in assigning a musical *state* to each token. Depending on the definition of a state, the number of existing tokens and the complexity of the problem vary. In the simplest setting, a state is a number between 0 and 127 that indicates which note is played (in case of silence, a special token must be introduced). This means that when several notes are played simultaneously the main one, based on the velocity, should be selected. Another possibility consists in keeping track of all the notes that are played. In this setting, a state must be represented by a boolean vector of size 128 (since each of the 128 notes can either be played or not).

A piece of music can then be converted into a sequence of states, with each state corresponding to a time instant. The discretization period (*i.e.*, the amount of time between two states) must be sufficiently small. The resolution is always a multiple of 24 in our datasets, and a statistical analysis of the two datasets revealed that 75% of midi files have a resolution of 480 ticks per beat or less: to keep the compression ratio at a reasonable level (pre-processed files of a decent size), we decided to set our resolution at 96 tpb. This value is commonly used in MIDI files, plus it is multiple of 24, which ensures good temporal mapping. After verification on our part, the compression effects of the various higher-resolution files was almost always inaudible.

This approach is conceptually simple, but led to the following issues (some of them are highlighted in [6]):

- **Limiting State Information:** The information contained in a state must be limited, otherwise the number of different tokens becomes prohibitive. For instance, if a state at time t is defined as the combination of the two most important notes at time t, there are $128 \cdot 127/2 \geq 8000$ different tokens. Taking into account more notes or other musical aspects (such as the tempo) would make the number of tokens grow up exponentially. Most of the sequence completion models output a discrete distribution over the next token of the sequence. In the next section, this is what we refer to as a *pure classification problem at the token level.* Hence the number of different tokens is also the size of the output of the model. This is the main reason why considering complex states is intractable with this problem formulation. In practice, it is difficult to handle tokens that represent more than one single note. This leads to a drastic simplification of what music is.

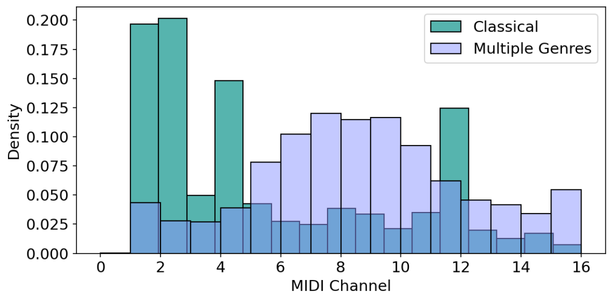
  Therefore, to deal with states that encode all the notes that are played (2^{128} different tokens), the music completion problem must be rephrased. In the next section, it will be shown that strong independence assumptions allow us to handle a very large token space while keeping the output of the model reasonably small. However, this solution suffers from its own limitations as well.

- **Discretization Period:** The discretization period must be small (smaller than 10 ms, as reported by [6]) to avoid distortions. Therefore, even a short context is converted into a long sequence of tokens. Most of the adjacent tokens are identical, because a piece of music usually contains notes that are played much longer than 10 ms. This is very problematic, because even if the context of the transformer is very large, the amount of information contained in the music is very small. This limits drastically the duration of the music that can be generated coherently.

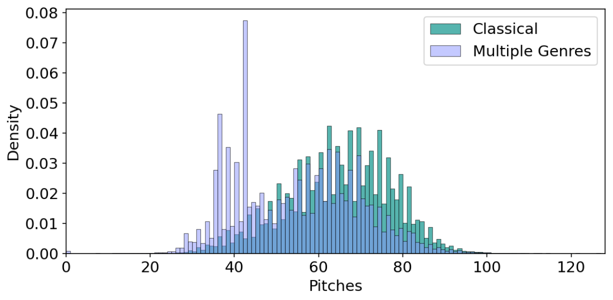
- **Practical Conversion Issues:** From a practical point of view, converting MIDI files to a sequence of state-based tokens is not trivial, because MIDI files contain events (see a previous section for more details), not states.

3. *Event-based tokenization*

Another tokenization approach consists in focusing on events. In this setting, a token represents a change in the music (*e.g.*, the apparition or disappearance of a note, a change in tempo, etc.). This tokenization is a direct translation of the MIDI files, and leads to much shorter



(a) Histogram of MIDI Channels



(b) Histogram of Pitches

FIG. 1: Distribution of MIDI channels and pitches for both datasets. *'Classical' refers to the Classical Music dataset and 'Multiple Genres' to the MIDIWORLD dataset*

sequences of tokens. In addition, capturing complex aspects of music does not require a exponential number of different tokens. For instance, with a succession of "note apparition tokens", one can encode very complex pieces of music in which several notes are played simultaneously. It is not necessary to have one token for each possible subset of notes.

Another advantage of this approach is that there are numerous methods to reduce the size of the vocabulary, that is, the number of tokens. These methods can be based on the a priori knowledge we have about the dataset or can be based on machine learning. There are also many methods, either based on prior knowledge or through learning, to enrich the vocabulary, such as *Byte Pair Encoding* [13].

This tokenization solves all the issues mentioned in the previous subsection, and provides an easy way to deal with multiple instruments (with additional tokens dedicated to specific instruments). In this work, we use the REMI tokenizer [14] of the MidiTok Python package [12]. It is a free-of-use implementation of the event-based tokenization. We use it to take into account the notes, their duration and their loudness (not the tempo, even of the tokenizer supports it). We also investigate multi-instrument music generation by means of this package.

C. Probabilistic model and music generation from sampling

Given a context $x_{1:T}$ (a sequence of T tokens), the deep learning model must output a distribution $p(x_{T+1}|x_{1:T})$ over the next token x_{T+1} of the sequence. This allows to use the model in a non-deterministic way by sampling in this distribution (hence, for a fixed context $x_{1:T}$, the model does not always complete the sequence in the same way). This subsection describes the different parametric distributions $p(x_{T+1}|x_{1:T})$ that were used throughout this work, and the associated loss functions used to train the models.

1. *Pure classification problem at the token level*

In its simplest form, the music completion problem can be cast into a regular classification problem. Denoting by N the number of existing tokens, the underlying probability distribution writes

$$x_{T+1} \mid x_{1:T} \sim \text{Discrete}(\boldsymbol{p}(x_{1:T}))$$

where $\boldsymbol{p}(x_{1:T}) \in \Delta^N$. The mapping from $x_{1:T}$ to $\boldsymbol{p}(x_{1:T})$ is implemented by the neural network. This is the most common approach to solve sequence completion problems. The minimization of the negative log-likelihood leads to minimizing the cross-entropy loss.

In this classification problem at the token level, the number of tokens is the number of classes (and therefore, the size of the output of the network). It must therefore remain reasonable. In other words, the tokenization must be the event-based tokenization, or the state-based tokenization in which a state contains only one note.

2. *Independent classification problems for each note*

In the previous approach, the size of the output vector $\boldsymbol{p}(x_{1:T})$ is equal to N, the number of different tokens. Therefore, this number must remain reasonably low, which prevents us from using a state-based tokenization that deals with more than one note per state. A possible solution is to change the underlying probabilistic model: instead of associating a probability to each token, one could associate a probability of activation to each note. In this setting, the network must output only 128 probabilities. This amounts to solve 128 independent binary classification problems. To generate the next token x_{T+1}, one can simply sample each of the 128 Bernoulli distributions independently from each other. This allows to have an arbitrary number of notes at the same time. This probabilistic model leads to a compact network output, because it makes the (very strong) assumption that the different notes are independent from each other. The consequences of this strong assumption will be analysed later.

To train the model, the loss function is the mean of the loss functions associated to each of the 128 binary classification problems, *i.e.* it is a mean of 128 binary cross-entropies.

D. Architecture

Since from a technical standpoint, music generation is similar to text generation (large language models), the use of a transformer is a natural choice. Our architecture is therefore based on the very first definition of a transformer, described in [1]. However, the encoder-decoder structure is not ideal in our case. We need the decoder-only transformer [15], which is the architecture used in large language models. The decoder-only architecture is trained with self-supervised learning, where the target sequence is the input sequence shifted by one token [16].

1. *Embedding and Positional Encoding*

First trial (v1) : In the first trial, tokens are encoded as integers from 0 to 129. To take advantage of the classic deep learning framework (*i.e.*, optimized tensor operations), we embed the tokens. The role of embedding is to convert the format of the initial tokens (integers) into a vector format better suited to tensorial operations. We proceed in the classic way when it comes to classification: each token is vector-mapped using a look-up table which produces vectors - one per token - whose dimension is a hyperparameter, defined by weights that are model-learnable parameters. As the initial space of tokens is of dimension 130, we arbitrarily decide to compress it to a space of dimensions $n_{embedding} = 65$. The fact that this compression is part of the learning process is a positive point, as it enables us to obtain a dense representation that is more suitable semantically for the generation procedure.

Second trial (v2): For the second trial, the input tokens are already vectors (states of the 128 notes at some time steps). Token-to-vector mapping is thus not required. The tokens are therefore only passed through a linear layer, which reduces their dimensionality by a factor of 2 while enabling the model to learn the semantics of the tokens.

Third trial (v3-v6) : For the third trial, we use event-based tokenization. It results in 281 different tokens. Their semantics differs from the tokens from the first trial, but from a transformer-technical point of view they can be considered as identical. So we proceed in a completely equivalent way for this trial: a learnable vector mapping. We reduce the 281-dimensional space to a 140-dimensional space.

Positional encoding : As self-attention layers are permutation-invariant, they cannot take into account the absolute or relative positions of tokens. However, information about the position of notes in a score is essential for music to be interpreted as a temporally organized sequence. We therefore must inject this information in the form of a ”positional embedding”. In our architecture, we simply add the absolute positions of the tokens after embedding them (to map their integer values, from between 1 and the length of the context, to a vector of size $n_{embedding}$) to each of the previously embedded tokens.

2. *Decoder Stacks*

The Decoder is composed of a stack of N identical blocks, composed of several types of layers.

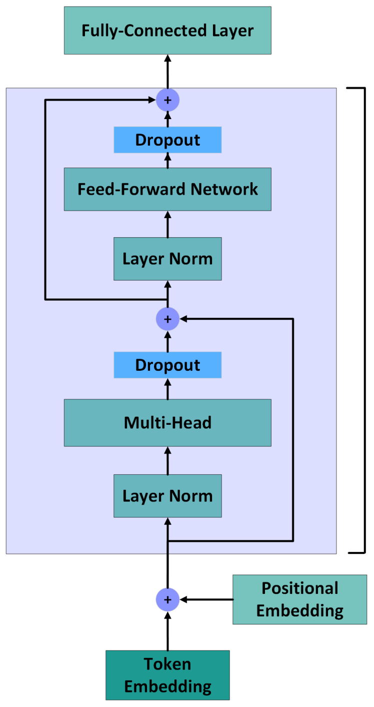


FIG. 2: Our Transformer Architecture

Multi-Head Masked Self-Attention : A Multi-Head is a block composed of several (`n_heads`) Heads, stacked in parallel, each one implementing the Self-Attention Mechanism from [1]. Each Scaled Dot-Product Attention is masked to prevent the model from accessing information from subsequent steps during output generation. This prevents overfitting and allows the model to keep the ability to generalize. It is also consistent with the sampling method used to generate new sequences.

One way in which our multi-head differs from [1] is in how we combine the information coming out of each Head. Indeed, after passing through the Multi-Head, the token keeps the same dimension: we do not concatenate the different outputs of the Heads. Instead, we just stack and sum all outputs coming from each head to obtain a single output, with the same dimension as the initial input.

Layer Normalization and skip connections : We employ layer normalization [17] before the Multi-Head and before the Feed-Forward Network, to stabilize the model and ensure faster convergence. We also add skip connections to allow the gradients to shortcut the layers and pass through the network without vanishing [16]. The initial input is added right after the Multi-Head block (first skip) and the output of the Multi-Head is added directly after the Feed-Forward Network.

Position-wise Feed-Forward Network : The last layer of our main block is a Position-wise Feed-Forward Network that processes each token independently. It is a small MLP composed of two linear transformations and a ReLU activation in between. The hidden layer has $4 \cdot n_{embedding}$ units.

3. *Fully Connected Layer*

The N blocks of the transformer architecture are ultimately followed by a fully-connected layer. The role of this layer is to decompress the information by mapping the $n_{embedding}$ dimensional space to a probability space. The size of this layer depends on the probabilistic model (as does the utilization of the output from the final layer). Generally speaking, we can say that the vector obtained assigns a score to each direction in the token space and that this score can be used, during the generation phase, to generate music or, during the training phase, to calculate the loss associated with the model and the batch.

4. *Training*

This subsection describes the training process, as well as the hyperparameters we use.

Dropout : To add some regularization during training, we apply some dropout ($p_{drop} = 0.3$) after the Multi-Head and the Feed-Forward Network. This prevents the model from overfitting and it improve its robustness.

Optimizer and Learning rate : We use the Adam optimizer [18] with default values for β ($\beta_1 = 0.9, \beta_2 = 0.999$) and ϵ ($\epsilon = 10^{-8}$). The learning rate we use does not vary over the course of training. It is fixed to $lr = 3.10^{-4}$. This choice is purely motivated by previous experiments made in the field of deep learning. When working with Adam, this value is typically small enough to allow stable convergence of the model while being large enough to enable rapid convergence, especially in the early stages of training.

Training all versions, summary : The training of our different versions requires the use of various hyperparameters (see TABLE I). A detailed description (beyond the architecture hyperparameters) of the different trials can be found in TABLE II.

Name	Code	N	n_heads	context	batch_size	#param
First trial	small v1	2	2	576	128	0.155M
First trial	big v1	10	7	576	64	1.285M
Second trial	v2	10	8	288	96	1.393M
Third trial	v3-v6	8	5	1024	64	3.838M

TABLE I: Different training parameters for each versions of our model.

Indeed, depending on the version considered, the number of possible tokens differs greatly: it was therefore appropriate to scale the various components of our architecture so as not to end up with models that were far too voluminous (e.g. for v2, a smaller context was required due to the large number of possible tokens) and we were limited by the hardware. Training two different models for the first trial is motivated by the desire to quantify the influence of the architecture size (*i.e.*, N stacked blocks and `n_heads` parallel heads) on the performance.

IV. RESULTS

In this section, we show our results for different combinations of methods, summarized in TABLE II. The two first trials use state-based tokenization, but with different probabilistic models and different definitions of "state". There are $128 + 2 = 130$ possible tokens in the first trial (since there are 128 notes, and two additional tokens for the silence and the end of a track) and 2^{128} for the second one (since each note can be on or off at a given time). The last trial explores event-based tokenization, based on two different datasets. The aspects related to the architecture are already shown in TABLE I.

For each trial, we provide several audio references that illustrate our observations and analyses. These audio demonstrations can be found in the GitHub repository given in the abstract. We denote by [x.y] the output of version $x \in \{1, 2, 3, 6\}$ after y iterations (*i.e.*, gradient steps). In each of these demonstrations, the prompt comes from the test set.

We do not show results for version 4 and 5, but these versions can be found in the repository. These are attempts to generate multiple-instrument music with event-based tokenization. The results are not completely satisfactory. We think that a longer training time is required, because of the larger number of tokens (456, which is a lot in a pure classification problem at the token level).

A. First trial (v1)

The first model was trained using two sets of hyperparameters in order to quantify the importance of the size of the architecture on the results obtained. In order to quantify the progress of our models, we listened to their different outputs and made the following observations:

1. At the start of the learning phase, the notes played by the two models are almost random. ([1a.500] and [1b.500])

2. In both cases, however, the model seems to improve VERY quickly: after only 2000 gradient descent steps, the outputs are consistent with the prompt: the model has learned to leave moments of silence and to play notes that have a similar tonality to those of the input. However, the results are still not very pleasant to listen to. ([1a.2000] and [1b.2000])

3. After 50,000 batch runs, the melody derived from the prompt is even more consistent with it: the model continues to learn. ([1a.50000], [1b.50000]) However, there is still room for improvement, as can be seen with the small model 1a, which has run up to 99,000 iterations, producing an even more refined sound despite the small size of the model. ([1a.99000])

In both cases, we can see that the loss function decreases greatly with the number of batches run, with the larger version evolving towards a smaller loss than the smaller version, which is expected (II).

The results are not entirely satisfactory, since summarizing a musical state to a single note is highly simplistic, and tokenizing the dataset in this way does not always render something harmonious: by keeping only the most strongly played note at each timestep, we lose the notion of chord and therefore musicality! The prompt itself and the elements on which the model is trained are therefore

Name	Code	Dataset	Tokenization	Nbr. of tokens	Probabilistic model
First trial	v1 (small, big)	Classical	state-based, main note	130	Classification at the token level
Second trial	v2	Classical	state-based, all notes	2^{128}	Independent classification for each note
Third trial	v3 (small, big)	Classical	event-based	281	Classification at the token level
Third trial	v6	Classical + MIDIWORLD	event-based	281	Classification at the token level

TABLE II: Combinations of methods for each of the trials

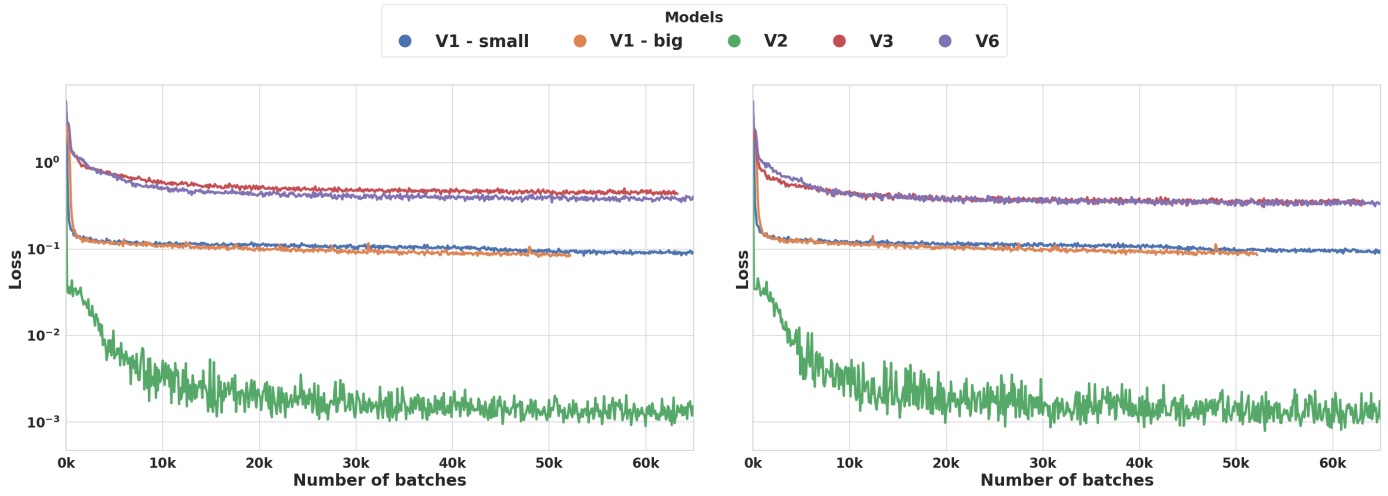


FIG. 3: Evolution of losses for the different models. *The graph on the left represents the training loss. The graph on the right represents test loss. Loss should be compared intra-model but not extra-model.*

not ideally suited, which makes this version oversimplified and therefore dissatisfying.

B. Second trial (v2)

This approach does not yield more satisfactory results, despite the strong decrease in the test loss and the train loss during training (see FIG. 3). Importantly, the loss function is the mean of 128 binary cross entropy functions (as explained earlier in the section dedicated to the probabilistic models), hence the values of the loss should not be compared with losses of other trials. In addition, a decrease in the loss function does not always bring an increase in musical quality, as discussed thoroughly in section V.

The following observations can be made when listening to the output of the model:

1. At the beginning of the training phase (after 1000 iterations, *i.e.*, 1000 gradient descent steps), the generated music is still random ([2.1000]), as expected.

2. Then, the model improves ([2.14000]), but very slowly compared to the other trials (see FIG. 3). One needs 14000 iterations to get an output which can be safely considered as non-random and linked to the prompt (for the first trial, only 2000 iterations are necessary to reach this goal). However, there are still notes that should not be played together. This is probably a consequence of the probabilistic model: all the notes are considered to be independent, which should not be the case in an harmonious piece of music. The use of this inappropriate probabilistic model can probably also explain why the training loss decreases less quickly.

3. Finally, the model seems to converge towards outputting no note at all (*i.e.*, the generated music mainly consists of silence). This is illustrated by [2.44000] and [2.96000]. The reason why the model behaves in this way will be investigated later.

This second trial is obviously a failure. It demonstrates that performing 128 independent classification problems does not work. Furthermore, the use of state-based tokenization leads to very short generated pieces of music.

C. Third trial (v3/v6)

Regarding the evolution of the outputs during the training phase, the observations made for the first trial still hold. In this section, we are more interested in assessing the impact of the training set on the results. We give two outputs for each version of the model (v3 is trained on the classical dataset, v6 on the classical dataset and the MIDIWORLD dataset). These outputs are [3.sea], [3.rivers], [6.sea] and [6.rivers]. Here the outputs are generated when the training phase is completed, hence indicating the number of iterations is not relevant anymore. The names of the files refer to the song from which the prompt is extracted.

These results are not the best ones from a musical point of view, but they demonstrate that:

1. The transition between the prompt and the artificially generated music is smooth, for both versions.

2. With the event-based tokenization, several minutes of music can be generated. In addition, the transformer succeeds in preserving a long-range structure (for instance, in [6.sea] the transformer inserts at the end

of the track a "pattern" seen in the prompt, which is something frequent in real music). This confirms that event-based tokenization greatly outperforms state-based tokenization.

However, sometimes (see for instance the end of [6.rivers], but it is the case in version 3 as well) the model repeats some patterns abusively. This is probably something that could be detected and removed by means of filtering heuristics.

3. Version 6 clearly outperforms version 3 regarding the style of the generated music: version 3 does not succeed in sticking to the style of the prompt (probably because it has been trained on classical music only, hence it tends to deviate towards this style), but version 6 does. It shows that generating consistent music requires training models on huge datasets made up of various musical styles. It makes the models robust and able to complete various prompts.

The best results are clearly obtained by the version 6 of the model. It is the largest model (see TABLE I), uses event-based tokenization, is based on a classification at the token level and is trained on the largest dataset. This combination of choices allows to generate music of reasonable quality (provided that the best outputs are selected).

V. DISCUSSION

Our findings show that it is indeed possible to generate music using a transformer-based architecture. This corroborates the results found in the literature. It is even feasible to achieve productions of human or superhuman quality. Our work not only confirms this potential but also provides deeper insights into the practical implementation and outcomes of using such models. We explore how transformers, with their advanced attention mechanisms and sequence processing capabilities, can produce coherent and musically rich compositions. This section will discuss the implications of our results, the challenges that we faced, and the broader impact on the field of AI-generated music.

We will address several key aspects: the absence of an objective metric for musical quality, the shortcomings of independent note sampling models, ethical and legal considerations, the critical role of tokenization, the influence of different musical styles, and possible improvements for future work. Each subsection will delve into these topics, offering a comprehensive analysis of the strengths and limitations of transformer-based music generation. Through this discussion, we aim to provide a thorough understanding of the current state of AI music generation and the potential directions for further advancements in this exciting intersection of technology and art.

A. Absence of an objective metric for musical quality

As the numerous articles cited in this work, we encountered the absence of an objective metric to quantify the quality of a musical production. While the value of the loss function serves as a metric for the model's progression in the learning phase, it does not reflect the distribution of the errors that are made. However, the impact of these errors on the quality of the generated music is strong. Obviously, establishing a heuristic for determining the quality of a musical production in a completely objective manner is impossible. Human musical tastes – the target audience of the model – are subjective and vary within the population. However, one could design a heuristic based on the mathematical theory of music to penalize predictions that are completely incompatible with music theory.

With this article, numerous productions are available. We decided to create two different dossiers: an unfiltered dossier (in which there is no selection), and a filtered dossier (in which we store the productions that appealed to us the most). Although this biases the results, it remains consistent with how we actually listen to music in our daily lives: often, we focus on artists or music that we appreciate and do not force ourselves to listen to musical productions that we consider to be of poor taste.

B. Failure of the trial based on independent note sampling.

The second model, based on conditionally independent prediction of the state of each of the 128 notes at each time step, failed miserably. In practice, there are several reasons why the results obtained are so unsatisfactory.

One important factor is the state-based tokenization method, which we know to be highly inefficient. However, this factor does not fully explain the model's failure, since the first version, also based on state-based tokenization, still produced qualitative results.

A second factor, and probably the most important, is the fundamental assumption of the model: it does not seem appropriate to assume that note-independent sampling is rational when it comes to producing music. In practice, we can illustrate this problem by imagining that the token x_{t+1} that the model is going to sample and that succeeds the token x_t could be either a chord A or a chord B. It's easy to imagine that, since sampling is carried out independently, when choosing the notes the model could mix chords. An alternative method would have been to generate conditional rather than independent distributions of the different notes, and to perform conditional sampling.

A final factor that is not entirely clear and could be investigated further is the reason for the long silences that the model learns to generate. Our understanding of this phenomenon is that, despite our preprocessing, a large part of the dataset contains completely empty note ranges (due to the multiple instruments that are not all played continuously and that have been flattened). Perhaps we should have removed the completely empty contexts from the batches. This explanation may not be the only reason for the silences.

C. Ethical and legal considerations in AI-generated music

The utilization of artificial intelligence to generate music presents notable ethical and legal challenges, particularly when leveraging datasets encompassing both public domain classical compositions and copyrighted contemporary works. Ethically, concerns arise regarding the originality and creativity ascribed to AI-generated compositions. While AI can adeptly mimic and synthesize existing musical styles, the debate over whether these creations should be considered original works or merely sophisticated derivatives remains contentious. Legally, the incorporation of copyrighted music into training datasets invokes issues of intellectual property rights. Copyright laws safeguard the rights of creators to their works, and the unpermitted use of these works for AI training purposes may constitute infringement. This issue is further complicated by the ambiguity surrounding the extent to which AI-generated outputs are influenced by the copyrighted material on which they were trained.

Given the rapid advancement and increasing capabilities of AI technologies, it is imperative for these ethical and legal questions to be actively deliberated within both public and legal domains. Experts in deep learning and AI are pivotal to these discussions, as their technical insights are crucial for comprehending the capacities and constraints of AI systems. Their expertise can inform policy-making that addresses the complexities of AI-generated content, ensuring that technological innovation progresses in a manner that respects intellectual property rights and adheres to ethical standards. Proactive engagement in these discussions is vital for developing robust frameworks that balance innovation with the protection of creators' rights, thereby fostering a fair and transparent environment for the ongoing evolution of AI in music generation.

D. The importance of tokenization

One important fact that our results highlight is just how important tokenization is. For the same architecture but different tokenization, we obtain results on both extremes of the music quality spectrum. If the total encoded information is more or less the same - in the sense that it is possible to reconstruct the midi file relatively correctly from the sequence of tokens - the use of one tokenization method or another has a considerable impact on the model's performance.

There are several possible explanations for this result. Firstly, a fundamental limitation of transformer architectures is the size of the n context. It is therefore important that the n tokens present in this context provide relevant information, and that the total amount of information in the context is maximized. In this way, the 'style' of the song is better represented.

In state-based tokenization, a large number of identical tokens follow one another, taking up a considerable amount of space in the context, and this phenomenon worsens if we wish to increase the temporal resolution of the sampling. It also quickly becomes intractable to encode complex states, as the size of the vocabulary explodes very quickly. Event-based tokenization doesn't suffer from these problems and gives much better results. This probably explains why event-based tokenization is widely used in the literature, while event-based tokenization is completely absent.

Secondly, event-based tokenization also makes it very easy to enrich the vocabulary by introducing tokens rich in meaning, significantly increasing musical complexity and enabling human or super-human level performances. For example, we could have introduced tempo variations and multi-instrumentation in a very simple way.

E. On the difference in musical styles

One of the most striking features of using the same architecture but two different datasets is the difference in the productions generated from the same prompt.

Although both datasets contain 'music' in the most generic sense, they are not initially identified as belonging to the same 'musical style'. While one might think that this classification is purely subjective and based on "human" criteria, it seems to have a "mathematical" basis. A first look at this basis is the difference in the distribution of pitches in Figure 1b. We could study the distribution of the succession of notes to delve deeper into the mathematical difference between the styles. The very marked difference in the results obtained indicates that the distribution of tokens and token sequences is very different in classical music than in well-mixed musical styles.

This diversity in music style can also be the source of problems in a dataset that is not properly pre-processed. In fact, we didn't sort out the types of instrument transformed into piano during tokenization based on a mono-instrumental principle, with the result that certain drum or wind instrument rhythms, fairly repetitive and without variation, found themselves translated into piano sequences where a single key is repeatedly pressed over a very long period of time. This is a source of problems as the model learns to reproduce this kind of pattern, and we sometimes find musical productions where the same key is played for dozens of seconds.

F. Limitations of the MIDI format

A general limitation of our model is the representation of music in MIDI format. Although this format offers several advantages for compressing music into an easily tokenizable format, it unfortunately comes with certain drawbacks. Listening to the 'MIDI' versions of relatively well-known music often reveals a lack of resemblance to the original versions. This discrepancy arises either because the method used to convert music into MIDI format is inefficient or due to the very synthetic and digital nature of the sounds produced. While the melodies are well-preserved, the timbre of the music is significantly constrained by the choice of MIDI format.

G. About possible improvements

There are numerous improvements that could be made to our architectures to enhance the quality and complexity of the generated music. Many examples from the literature go beyond what we have experimented with in MOzART. If we were to push further, the next steps could involve utilizing a more complex and rich tokenization scheme with the possibility of introducing variations in tempo, instruments, and signature. The utilization of machine learning methods by tokenizers would be a promising avenue to explore, aiming to maintain a limited yet densely informative token count. Larger architectures trained for extended durations could also be explored, as it appears we have not yet reached the limit of model progression. Technical enhancements such as relative attention mechanisms, heuristics based on music theory, or augmenting larger datasets are feasible considerations. For generating productions, we employed a temperature parameter - allowing for the smoothing or sharpening of the probability distribution to sample the next token - with a value of 1; however, investigating the outcome of a higher temperature could yield more original productions. Lastly, fine-tuning procedures could be envisioned to further enhance the model's results. Numerous other possibilities warrant exploration.

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[2] Keunwoo Choi, George Fazekas, and Mark Sandler. Text-based lstm networks for automatic music composition, 2016.

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

[4] Pedro Ferreira, Ricardo Limongi, and Luiz Paulo Fávero. Generating music with data: Application of deep learning models for symbolic music composition. *Applied Sciences*, 13(7), 2023.

[5] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations, 2018.

[6] Sageev Oore, Ian Simon, Sander Dieleman, Douglas Eck, and Karen Simonyan. This time with feeling: Learning expressive musical performance, 2018.

[7] Alexey Tikhonov and Ivan Yamshchikov. Music generation with variational recurrent autoencoder supported by history. *SN Applied Sciences*, 2, 12 2020.

[8] Rui Guo, Ivor Simpson, Thor Magnusson, Chris Kiefer, and Dorien Herremans. A variational autoencoder for music generation controlled by tonal tension, 2020.

[9] Mido MIDI Objects for Python. Message types. `https://mido.readthedocs.io/en/stable/message_types.html`, 2024. Accessed: 2024-05-15.

[10] Bohdan Fedorak. Midi classic music. `https://www.kaggle.com/datasets/blanderbuss/midi-classic-music/data`, `http://www.piano-midi.de/`.

[11] Midi files from midiworld. `https://www.midiworld.com/`. Copyright © 1995-2024 MIDIWORLD All rights reserved.

[12] Nathan Fradet, Jean-Pierre Briot, Fabien Chhel, Amal El Fallah Seghrouchni, and Nicolas Gutowski. MidiTok: A python package for MIDI file tokenization. In *Extended Abstracts for the Late-Breaking Demo Session of the 22nd International Society for Music Information Retrieval Conference*, 2021.

[13] Philip Gage. A new algorithm for data compression. *The C Users Journal*, 12(2):23–38, 1994.

[14] Yu-Siang Huang and Yi-Hsuan Yang. Pop music transformer: Beat-based modeling and generation of expressive pop piano compositions. In *Proceedings of the 28th ACM International Conference on Multimedia*, MM '20, page 1180–1188, New York, NY, USA, 2020. Association for Computing Machinery.

[15] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training, 2018.

[16] Aston Zhang, Zack C. Lipton, Mu Li, and Alex J. Smola. *Dive Into Deep Learning*. Cambridge University Press, 2023.

[17] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. 2016.

[18] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2015.