On Different Symbolic Music Representations for Algorithmic Composition Approaches based on Neural Sequence Models

Felix Schön[0000−0003−0616−3081] and Hans Tompits[0000−0001−5673−2460]

Institute of Logic and Computation E192-03
Technische Universität Wien
Favoritenstraße 9-11, 1040 Vienna, Austria
{schoen,tompits}@kr.tuwien.ac.at

Abstract. Among the different approaches for automated music composition, those based on neural sequence models like the transformer show particular promise. A critical aspect for such approaches is how given music data sets are represented, or *tokenised*, for serving as suitable inputs for such models, as the choice of representation influences the quality of the produced output. In this paper, we introduce seven novel tokenisation techniques for converting MIDI data into numeric sequences. We compare characteristics of our tokenisers based on sets of musical data translated using our approaches. Our results show that some of our techniques greatly outperform the approaches found in the literature with respect to different metrics such as sequence length, information density, or memory requirements. Moreover, to evaluate the influence of our tokenisation approaches on the quality of the output of a model, we trained an ensemble of transformer models on the sets of tokenised musical data and performed a user study to assess the quality of the generated music pieces. The result of the study shows that the quality of pieces produced using our most promising techniques is equal to or outperforms state-of-the-art approaches.

Keywords: Algorithmic Composition · Symbolic Music Generation · Transformer Neural Networks.

1 Introduction

The term *algorithmic composition* (AC) refers to the technique of creating music by means of a formal set of rules or algorithms. While many different automated AC methods have been realised [9, 1, 3, 1, 4, 21], the arguably most successful systems developed in recent years are those based on the *transformer neural network architecture* [27], like the MusicTransformer [16] and MuseNet [23].

As the transformer is a *neural sequence model*, AC methods based on such an architecture treat music as a *language*, where a given digital representation of a musical score is translated into discrete elements, called *tokens*, which are then processed by the network model. These sequence-based methods constitute

symbolic music generation techniques where pieces of music are represented using sequences of symbols from a specific vocabulary representing, e.g., notes, rests, or time signature changes.

Now, the particular representation of the discrete tokens critically influences the network's ability to learn the underlying structure of the input [13]. Furthermore, representation lengths and vocabulary sizes significantly influence memory and computing resource requirements, an often limiting factor for real-world applications.

While research on transformer-based AC methods have for the most part focused on tuning the machine-learning algorithms [6–8, 17], the question of the influence on the token representation has received much less attention. Indeed, notable exceptions in this regard include the works by Huang and Yang [17], who introduced REMI, a representation that explicitly models note durations, CP, due to Hsiao et al. [15], which is based on REMI, and Note Tuple [14].

In this paper, we introduce a suite of different symbolic music representation specifications, also referred to as *tokenisation approaches*. In particular, our main focus lies on reducing the average input sequence length while retaining the same output quality as with representations using longer sequences. More specifically, we present seven tokenisations, classified into so-called MIDI-*like approaches* and *note-like approaches*, respectively. In the former category, representations are based on the way the MIDI protocol [20] represents musical data, whilst tokenisations in the latter category are inspired by the way traditional sheet music is written. For some of our note-like approaches, we make use of a greater vocabulary size. As a result, individual tokens can provide more information than their regular counterparts, e.g., both the pitch of a note and its duration.

Our note-like representations use a similar approach as REMI [15], where note durations are modeled explicitly. However, in contrast to REMI, we do not require these durations to be defined on a per-note basis but rather make use of a *running duration*, allowing for note durations to apply to all successive notes until superseded by the next duration. In the CP [15] method, the model architecture is modified to produce "super tokens" which combine several different REMI tokens. In contrast, our so-called *large-vocabulary representations* similarly represent multiple note attributes using a single token, but can be used without modifying the architecture of a model, making them compatible with more general models and requiring no additional computational effort.

In order to assess the different tokenisations introduced in this paper, we compare them with respect to certain parameters, like sequence length, information density, and memory requirements, applied on a combination of three training sets. Moreover, in order to evaluate the quality of pieces generated with our representations, we trained an ensemble of transformer models on the used data set and compared their output in a user study. The results show that some of our representations significantly reduce the average number of tokens needed to represent a musical sequence—in some cases as much as 40% compared to commonly used approaches such as REMI—without compromising on the output quality.

2 Background

We first provide some basic terminology from music theory (for more information on this subject, cf., e.g., the works of Benward and Saker [2] or Laitz [18]).

In musical scores, *notes* correspond to a tone of a specific pitch. The higher the note is situated on the score, the higher is the corresponding pitch. In general, 88 different notes are used, ranging from A0 (the lowest note) to C8. Notes have a *value*, which is a property that refers to the duration of the tone it represents. This duration is relative with regard to the *bar* it is contained in. A bar is a grouping of notes with a specific overall duration. Commonly, a bar can fit up to four consecutive quarter notes or any combination of (simultaneous) notes that take the same amount of time to play. Using *time signatures*, the capacity of a bar can be specified, usually in terms of how many consecutively played quarter notes it can fit. *Rests* mark pauses in the composition, and, in a similar fashion to notes, the length of these pauses is determined by their value. This, in combination with note values, allows for the construction of *rhythm*.

Relevant for our purposes is also the MIDI file format [20], which is an industry standard for connecting electronic music and audio devices. MIDI files can be used to store musical performance data using sequential blocks of binary data. These blocks can constitute *events*, which consist of a *time-delta* definition, stating how many *ticks* (units of time) passed between the last and the current event, and a *message*, containing musical performance data. Commonly, MIDI uses 24 ticks for the duration of a quarter note.

Important for us are only the so-called "note on" and "note off" messages, indicating the start and end of a note, respectively. These can be used to model, e.g., the pressing and releasing of a piano key. Here, the message contains information about its exact type, the note's pitch, and its *velocity* value. The latter indicates how "loud" or with how much expression a note is played. For the sake of clarity, we represent MIDI messages using a textual representation, e.g., `(note on A4)` for the message indicating the start of A4.

As neural sequence models can only accept numerical input sequences, musical input compositions need to be translated into such a format before they can be used to train the models. This process is referred to as *tokenisation*. Formally, we are interested in transforming a sequence $S_{\text{source}} = (x_1, \ldots, x_n)$, where $x_i \in V_{\text{source}}$, for $1 \leq i \leq n$, into a sequence $S_{\text{target}} = (y_1, \ldots, y_m)$, where $y_i \in V_{\text{target}}$, for $1 \leq i \leq m$. Here, n and m give the lengths of the sequences, respectively, while $V_{\text{source}} = \{x_1, \ldots, x_s\}$ and $V_{\text{target}} = \{y_1, \ldots, y_t\}$ are vocabularies of size s and t, respectively.

For our purposes, V_{source} is the set of all MIDI messages, i.e., $V_{\text{source}} = V_{\text{midi}}$, where $V_{\text{midi}} := \{\ldots, \texttt{(note on A4)}, \texttt{(note off A4)}, \ldots\}$, while V_{target} contains the tokens used by the specific tokenisation approaches discussed in Section 3.

The resulting sequence S_{target} can then be used during the *training* process, where the model is tasked to learn the underlying probability distribution of the corpus of input sequences. In the *inference* step, the model is then used to generate new sequences. This is done by repeatedly tasking it to predict y_i based on $(y_0, \ldots, y_{i-1})$, where $y_1, \ldots, y_{i-1}$ are the predictions made in previous steps.

Finally, the resulting output sequence $(y_1, \ldots, y_m)$ can be translated back into a sequence $(x_1, \ldots, x_n)$, where $x_i \in V_{\text{source}}$, for $1 \leq i \leq n$. Using this approach, neural sequence models can be utilised to compose new musical pieces.

3 Representations

We now introduce our tokenisation approaches for musical sequences. From an abstract point of view, a *tokenisation*, or *representation*, is a function mapping a sequence of elements from the MIDI vocabulary V_{midi} to a sequence of elements of the vocabulary of the tokeniser, which in our setting are sequences of natural numbers.

Our tokenisers can be categorised using the following three characteristics: (i) *temporal representation* (MIDI-like or note-like), (ii) *pitch representation* (absolute, relative, or circle-of-fifths), and (iii) *vocabulary size* (regular or large-vocabulary).

In what follows, we use the following notation: For any set X, $[X]^*$ denotes the set of all finite sequences of elements from X. Then, a *tokenisation* is a function $f : [V_{\text{midi}}]^* \to [V_f]^*$, where the codomain $[V_f]^*$ of f is a set of sequences of natural numbers and the set V_f is referred to as the *target vocabulary* of f.

Based on specific choices of the target vocabulary, we introduce the following tokenisations:

(i) the *relative* MIDI-*like representation*, $\mathcal{T}^{\text{rel}}_{\text{midi}}$, with target vocabulary $V^{\text{rel}}_{\text{midi}} := \{x \in \mathbb{N} \mid 0 \leq x \leq 392\}$);
(ii) the *circle-of-fifths* (CoF) MIDI-*like representation*, $\mathcal{T}^{\text{cof}}_{\text{midi}}$, with target vocabulary $V^{\text{cof}}_{\text{midi}} := \{x \in \mathbb{N} \mid 0 \leq x \leq 84\}$);
(iii) the *regular note-like representation*, $\mathcal{T}^{\text{reg}}_{\text{note}}$, with target vocabulary $V^{\text{reg}}_{\text{note}} := \{x \in \mathbb{N} \mid 0 \leq x \leq 132\}$);
(iv) the *large-vocabulary note-like representation*, $\mathcal{T}^{\text{lvoc}}_{\text{note}}$, with target vocabulary $V^{\text{lvoc}}_{\text{note}} := \{x \in \mathbb{N} \mid 0 \leq x \leq 1450\}$);
(v) the *relative note-like representation*, $\mathcal{T}^{\text{rel}}_{\text{note}}$, with target vocabulary $V^{\text{rel}}_{\text{note}} := \{x \in \mathbb{N} \mid 0 \leq x \leq 219\}$);
(vi) the *circle-of-fifths note-like representation*, $\mathcal{T}^{\text{cof}}_{\text{note}}$, with target vocabulary $V^{\text{cof}}_{\text{note}} := \{x \in \mathbb{N} \mid 0 \leq x \leq 73\}$); and
(vii) the *large-vocabulary circle-of-fifths note-like representation*, $\mathcal{T}^{\text{lcof}}_{\text{note}}$, with target vocabulary $V^{\text{lcof}}_{\text{note}} := \{x \in \mathbb{N} \mid 0 \leq x \leq 3307\}$).

For the sake of clarity, we represent the target vocabularies of our tokenisations using a textual representation rather than their numerical value, e.g., "`(note on A4)`" stands for "77". Furthermore, the shorthand "`(note on _)`" refers to all possible variations of a message, e.g., `(note on A0)` through `(note on C8)`. In Table 1 we give a comparison of textual and numerical representations of tokens for a selection of different representations.

Common to all our representations are the `(start)` and `(stop)` tokens which mark the beginning and end of a composition, respectively, a `(pad)` token that does not represent any musical element and is only used to pad sequences to

Table 1. Textual and numerical representations of the basic tokens used by our tokenisers.

Textual Representation	Numerical Representation
Common to all Tokenisers	
`(pad)`, `(start)`, `(stop)`	0, 1, 2
Relative MIDI-like Tokeniser $\mathcal{T}_{\text{midi}}^{\text{rel}}$	
`(wait 1)` – `(wait 24)`	4 – 27
`(note on -87)` – `(note on +87)`	28 – 202
`(note off -87)` – `(note off +87)`	203 – 376
Regular Note-like Tokeniser $\mathcal{T}_{\text{note}}^{\text{reg}}$	
`(wait)`	4
`(value definition 1)` – `(value definition 24)`	5 – 28
`(note play A0)` – `(note play C8)`	29 – 116
Large-Vocabulary Note-like Tokeniser $\mathcal{T}_{\text{note}}^{\text{lvoc}}$	
`(wait 1)` – `(wait 24)`	4 – 27
`(note value 2 play A0)` – `(note value 2 play C8)`	28 – 115
⋮	⋮
`(note value 96 play A0)` – `(note value 96 play C8)`	1348 – 1435
Circle-of-Fifths Note-like Tokeniser $\mathcal{T}_{\text{note}}^{\text{cof}}$	
`(wait)`	4
`(value definition 1)` – `(value definition 24)`	5 – 28
`(octave shift -8)` – `(octave shift +8)`	29 – 45
`(note play cof -5)` – `(note play cof +6)`	46 – 57

a specific length, and 15 `(time signature _)` tokens used to define the time signature of a bar, determining how much capacity it has. We support time signatures ranging from $\frac{2}{16}$ up to $\frac{16}{16}$. Here, each `(time signature _)` token represents exactly one of the supported time signatures. We do not make use of *velocity* tokens for our representations but argue that including them would be straightforward, e.g., by using a set of `(velocity _)` tokens.

Rather than discuss the individual representations, we will illustrate the differences between the three characteristics mentioned above. This way, the representations are defined implicitly. We refer to Table 1 for more details on the basic token types.

The source code for our tokenisation approaches, the code used to train the models, the weights of the trained models, the generated samples, and the raw survey results can be found at

https://github.com/FelixSchoen/AIxIA-2024.

Temporal Representation. The MIDI-*like temporal representation*, as its name suggests, is based on the way the MIDI protocol [20] represents musical data.

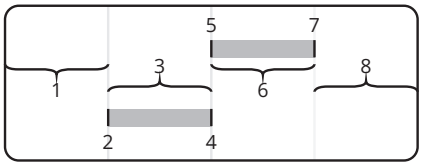


Fig. 1. A graph of the MIDI-like representation for one bar. (1: `(wait 6)`, 2: `(note on A4)`, 3: `(wait 6)`, 4: `(note off A4)`, 5: `(note on C5)`, 6: `(wait 6)`, 7: `(note off C5)`, 8: `(wait 6)`).

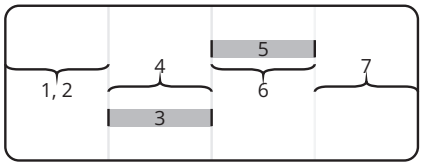


Fig. 2. A graph of the note-like representation for one bar. (1: `(value definition 6)`, 2: `(wait)`, 3: `(note play A4)`, 4: `(wait)`, 5: `(note play C5)`, 6: `(wait)`, 7: `(wait)`).

For our tokenisers, we make use of 88 `(note on _)` tokens to represent the start of a note, and 88 accompanying `(note off _)` tokens to mark the end of it. Furthermore, we make use of 24 `(wait _)` tokens, which correspond to the passing of the respective amount of *ticks*. Using this resolution, we can represent note values as small as thirty-second triplets—which have a duration of 2 ticks—using our representation.

Figure 1 depicts a visual representation of the MIDI-like temporal representation. Here, the beginnings and ends of notes have to be modeled explicitly.

Our *note-like representations* are inspired by the way traditional sheet music works. Instead of using a combination of `(note on _)` and `(note off _)` tokens to represent a single note, we use a `(value definition _)` token to define the length of the successive note, indicated by a `(note play _)` token, e.g., a `(value definition 24)` followed by a `(note play _)` token constitutes a quarter note. This greatly reduces the risk of invalid tokens, e.g., a `(note off _)` for a note that has not been previously opened.

A novel feature of our note-like temporal representation is the notion of a *running value*. Instead of having to define a value for each note, the current value applies to all subsequent tokens until it is replaced. This approach can greatly reduce the number of tokens in a sequence, reducing the computational costs associated with training neural sequence models.

In order to represent rests, we make use of a singular `(wait)` token, which marks an advancement in time in the current sequence. This token also accepts a value definition, or—in the case of a running value—makes use of the current active value.

We allow for consecutive `(value definition _)` tokens. In this case, the total amount of ticks is summed up and applied to the succeeding tokens. This way, notes or rests that last longer than the maximum amount supported by `(value definition _)` tokens can be defined.

Figure 2 depicts a visual representation of a note-like representation. In contrast to the MIDI-like approach, only the start of a note has to be modeled explicitly, its duration is given by previous value definitions.

Pitch Representation. We consider three different pitch representation types: *absolute*, *relative*, and *circle-of-fifths* type.

With the absolute approach, pitches are defined using only their key number on the piano, e.g., 49 for an A4.

With the relative approach, a pitch is defined by the distance to its predecessor. This is modeled using 175 `(note on _)` and the same amount of `(note off _)` tokens, indicating the distance of the current pitch to the previous one within the interval $[-87, +87]$. As the first note in a sequence has no predecessor, we define its representation as the distance from the international standard pitch A4 to it.

With the circle-of-fifths approach, distances between pitches are given by their distance on the *circle of fifths*, which is a categorisation of pitch classes. Any two adjacent classes on this circle, e.g., C and G, or A and E, are exactly 7 semi-tones apart, an interval commonly used in music. We use 12 `(note on _)` and the same amount of `(note off _)` tokens to represent the distance between two pitches on the circle of fifths and 17 `(octave shift _)` messages to be able to model shifts in octaves between two pitches of -8 to $+8$.

Vocabulary Size. In contrast to the regular vocabulary size, with the *large-vocabulary approach*, we replace the use of individual pairs of `(value definition _)` and `(note play _)` tokens by a single `(value _ play _)` token, specifying both value and pitch of a note. This approach requires a large amount of unique tokens in order to model all possible combinations. It works analogously for other pitch representation types, e.g., note value and relative distances can be combined into a single token.

Note that this approach only works for note-like temporal representations as here note values are modeled explicitly.

4 Experiments and Evaluation

To compare between our different tokenisers, we conducted an analysis of the sets of tokenised sequences produced using our approaches. These sets were obtained by tokenising the MIDI files from the dataset discussed in Section 4.1 below. To evaluate the differences in quality between neural sequence models trained using different tokenisation approaches, we conducted a survey on the output of an ensemble of transformer models trained on the same sets of tokenised sequences.

Note that for the sake of comparison, we include in our analysis both the regular MIDI-like tokeniser as found throughout literature [22, 23, 16] and a REMI-like tokeniser similar to the one used by the state-of-the-art Museformer [28] that does not make use of a running value but is otherwise identical to our regular note-like tokeniser.

4.1 Experimental Settings

Model Parameters. We use the original transformer architecture as introduced by Vaswani et al. [27] for our evaluation since we want to isolate the evaluation

of our tokenisation processes and make it as replicable as possible. We used as hyperparameters a model dimensionality of 256, 1024 neurons per feed-forward layer, 4 attention heads, 4 encoder layers, and a dropout rate of 0.15 with a length limit of 1024 for all tokenisers. We utilise the AdamW optimiser [19] with beta values of $0.9, 0.98$, an epsilon value of 1×10^{-9}, and a weight decay value of 0.1. For the learning rate we adapted the original transformer learning rate with 8000 warm-up steps and a multiplicative factor of 2, training for up to 64 epochs with a batch size of 2 over 8 accumulation steps, resulting in a practical batch size of 16. The training was conducted on the CLIP cluster[1] using four NVIDIA Quadro RTX 6000 cards per node.

Dataset. In order to test our tokenisation approaches on a wide variety of musical pieces, we used a combination of three different datasets, namely the *piano-midi.de*, the *ADL Piano MIDI* [10], and the *ASAP* [11] dataset. We utilised our music library S-Coda [25, 26] to preprocess and tokenise the MIDI files. Each piece was first checked for eligibility based on metrics such as number of tracks and empty bars and then split into chunks of 8 consecutive bars. We used a *stride* of 4 bars, i.e., every fourth bar in a piece of the dataset marks the beginning of an 8-bar chunk used for training.

4.2 Analysis of the Sets of Tokenised Sequences

Table 2 shows the numerical results of several analysis approaches of the sets of tokenised musical sequences. As their name suggests, the large-vocabulary approaches exhibit significantly higher vocabulary sizes than their regular counterparts. Note that although larger vocabulary sizes require more computational resources during the embedding step, for transformer models, the main bottleneck stems from the overall length of sequences. The large-vocabulary approaches perform exceptionally well in this regard. They are able to outperform all other approaches in this aspect, improving upon the regular note-like representation by almost 25%, over 40% for the regular MIDI-like representation, and over 50% compared to the REMI-like representation. Both the regular note-like and relative note-like approaches outperform the conventionally used MIDI-like representation techniques and could serve as a good alternative to their large-vocabulary counterparts if smaller vocabulary sizes or the inclusion of information such as, e.g., velocity values is desired.

Interesting to note are the values for the standard deviation. The non-large-dictionary approaches making use of a circle-of-fifths representation exhibit especially high values here, suggesting that the range of sequence lengths is quite high. This has a particularly severe impact, as in a batch, all sequences are padded to the highest sequence length among them, drastically increasing memory requirements.

We compared our approaches also with respect to *entropy*, which is used to measure the information content of a random variable, where a higher entropy

[1] https://clip.science.

Table 2. Results of the analysis on the different sets of tokenised sequences.

Tokeniser	Vocab. Size	Avg. Seq. Len.	Std. Dev. Seq. Len.	Entropy	Gini Coeff.	Final Loss	Memory Req.	Survey Wins
Regular MIDI-like Tokeniser	219	317.68	148.98	6.60	0.63	0.84	7.52 GB	60%
Relative MIDI-like Tokeniser	393	317.68	148.98	6.2	0.80	0.75	6.60 GB	37%
CoF MIDI-like Tokeniser	85	460.90	224.92	4.85	0.72	0.68	6.25 GB	40%
Regular Note-like Tokeniser	133	252.83	109.93	5.05	0.75	0.99	2.35 GB	57%
REMI-like Tokeniser	133	385.77	169.36	4.6	0.81	0.65	8.09 GB	56%
Large Vocabulary Note-like Tokeniser	1451	190.12	84.17	7.30	0.86	1.29	1.95 GB	61%
Relative Note-like Tokeniser	220	252.84	109.93	4.96	0.84	0.88	3.75 GB	53%
CoF Note-like Tokeniser	74	329.58	147.55	4.35	0.77	0.84	12.41 GB	37%
Large Voc. CoF Note-like Tokeniser	3308	190.12	84.17	7.27	0.90	1.2	2.21 GB	48%

indicates a more random distribution and thus a larger amount of information per observation [24]. It is calculated using the formula

$$H(X) := -\sum_{x \in \mathcal{X}} P(x) log_2 P(x),$$

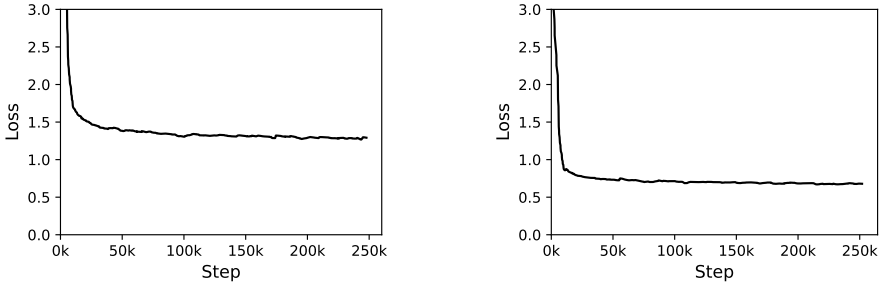
where X is a random variable with domain $\mathcal{X}$ and $P(x)$ is the probability of X having value $x \in \mathcal{X}$. Lower values of entropy indicate a higher predictability of next words while higher values imply larger vocabulary sizes and greater information gain per word.

Both large-vocabulary approaches exhibit significantly higher entropy values than their regular-vocabulary counterparts. This is in accordance with the decreased sequence lengths as each word carries a larger amount of information for these approaches. On the other hand, the regular circle-of-fifths approaches carry the least amount of information per word.

Another parameter we considered is the *Gini coefficient* [12, 5], which can be used to measure statistical inequality in the distribution of values across a set of classes. It is given by

$$G = \frac{1}{2n^2\overline{x}} \sum_{i=1}^{n} \sum_{j=1}^{n} |x_i - x_j| ,$$

where n is the number of classes, x_i is the number of values that belong to class i, and $\overline{x}$ is the average of values per class. Alternatively, if $x_i \leq x_{i+1}$ holds, we



(a) Learning curve of the model trained using the large vocabulary note-like tokenisation approach.

(b) Learning curve of the model trained using the circle-of-fifths MIDI-like tokenisation approach.

Fig. 3. Learning rates of two models trained on the same dataset using two different tokenisation approaches.

can calculate the Gini coefficient as follows:

$$G = \frac{1}{n} \sum_{i=1}^{n} (2i - n - 1) x_i \cdot \left(\sum_{i=1}^{n} x_i \right)^{-1} .$$

The Gini coefficient provides a measure of how equally observations are distributed over a number of classes. $G = 0$ indicates a perfectly equivalent distribution while $G = 1$ indicates a maximally imbalanced distribution, e.g., all samples belonging to a single class.

The Gini coefficient is relatively high for all our representation approaches, suggesting that a few single tokens are heavily prioritised over others. We argue that this most likely stems from the repeated usage of the `(wait)` token which is essential for the construction of rhythm and temporal resolution. Although in practice this did not pose problems, future variations of our representations could try to tackle this shortcoming. We note that, for the calculation of the Gini coefficient and the entropy value, we only considered tokens of the vocabulary that were used in the dataset in order not to dilute the results due to a large number of unused tokens.

We trained an ensemble of nine transformer models on the dataset, for each of the representation techniques, respectively. For the sake of reproducability, we report the final loss values achieved for each of the models after the last epoch. Note that due to the differences in vocabulary size between the approaches, the loss values are not directly comparable. This is reflected in Figure 3, showing learning curves for two of the models. Although the learning curve in Figure 3b seems to outperform the one in Figure 3a, in practice the former model exhibits significantly higher output quality compared to the latter.

Lastly, we include the highest reported memory usage during the training of the models using the respective representation (cf. Section 4.1 for the exact hyperparameters used). In contrast to the loss values, these values are directly comparable and provide important insights. Here, our large-vocabulary approaches perform exceptionally well, reducing the memory requirements by approximately 70% compared to the regular MIDI-like- and REMI-like representation. Despite the large difference in average sequence length, the memory requirement of the regular note-like representation is comparable to our large-vocabulary approaches. We argue that this is due to the overhead induced by the significantly larger vocabulary size. Increasing the number of layers would likely result in the regular note-like approach consuming drastically more memory as the space requirements are dependant on the square of the sequence lengths.

4.3 Evaluation of the Quality of the Generated Music

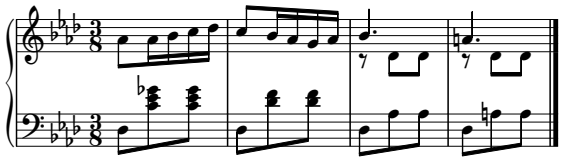
In order to evaluate qualitative differences between pieces generated by the models trained on the different sets of tokenised sequences, we performed a survey involving 11 participants. For each of the nine models, we generated 64 compositions. Here, the models were only primed with the `(start)` token and tasked to predict new tokens until the `(stop)` token or a length of 1024 is reached. Note that we did not remove any failure samples in this process as this could potentially skew the results of the survey.

The participants were provided with eight pairs of two compositions, randomly sampled from the output of one of the trained models. Note that the origin in a pair of samples was mutually exclusive. We then asked the participants to select which of the pieces they believed to be more musically sophisticated, interesting, or less computer-generated. Table 2 shows the percentage of match-ups won for each of the tokenisers. As the pieces were randomly drawn for each instance of the survey, we allowed for multiple submissions. We asked the participants to mark their entries as repeats in this case. In total, we received 17 submissions comparing 272 compositions generated by our models.

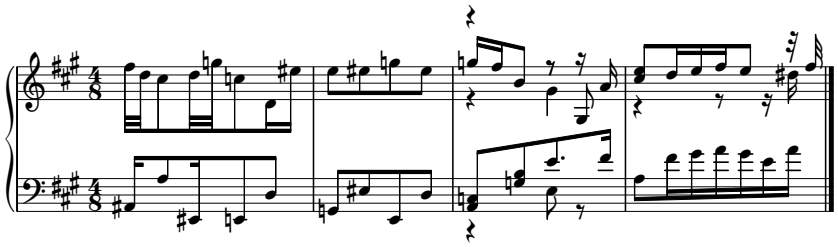
The survey shows that with 60.6% of 33 match-ups, our large-vocabulary note-like tokeniser performed best. This is an encouraging result as it suggests that the quality of pieces produced using this technique is on par or better compared to pieces produced using more traditional approaches, even though the sequence length is greatly reduced.

All of the regular MIDI-like tokenisers, the regular note-like tokeniser, and the REMI-like tokeniser showed good promise as well, winning 60% of 30, 57.1% of 28, and 56.3% of 32 match-ups, respectively. Contrary to our assumptions, the relative and circle-of-fifths approaches did not perform as well, often exhibiting frantic jumps between notes and disharmonic melodies.

Figure 4 depicts score representations for two samples generated by two of our models. More specifically, Figure 4a shows a piece generated by our large-vocabulary note-like model which exhibits typical rhythmic and melodic structure. On the other hand, in Figure 4b, a piece generated by the circle-of-fifths



(a) Score representation of a piece generated using the large vocabulary note-like tokenisation approach.



(b) Score representation of a piece generated using the circle-of-fifths MIDI-like tokenisation approach.

Fig. 4. Score representation of two pieces generated by models trained on the same dataset using two different tokenisation approaches.

MIDI-like model is given. Here, erratic jumps in pitch and inconsistent rhythm can be observed.

5 Conclusion

In this paper, we introduced seven novel tokenisation techniques for symbolic music generation. These techniques can be categorised into two main categories: (i) MIDI-like tokenisers and (ii) note-like tokenisers. Our large-vocabulary note-like tokeniser shows particular promise. It makes use of a note-like temporal representation style while using a large vocabulary of tokens to indicate pitch and value of a note using a single token. This way, we are able to reduce the average length of a sequence needed to represent a musical composition by more than 40% compared to the most commonly used and the REMI-like approach.

Our user study shows that the output quality of models trained using our large-vocabulary note-like tokeniser is not negatively impacted by the reduction of sequence lengths.

Concerning future work, we plan on adapting our tokenisers to a multi-track setting, potentially supporting a variety of instruments. Here, the large-vocabulary approaches could be extended by additional sets of tokens representing a specific instrument or track. Furthermore, for the note-like approaches, an idea analogous to the running value could be implemented, specifying the current instrument for all subsequent messages until replaced.

References

1. Bell, C.: Algorithmic music composition using dynamic Markov chains and genetic algorithms. Journal of Computing Sciences in Colleges **27**(2), 99–107 (2011)
2. Benward, B., Saker, M.: Music in Theory and Practice: Volume 1. McGraw-Hill, New York, NY, USA, eighth edn. (2009)
3. Biles, J.A.: Autonomous GenJam: Eliminating the fitness bottleneck by eliminating fitness. In: Poon, J., Maher, M.L. (eds.) Proceedings of the GECCO 2001 Workshop on Non-routine Design with Evolutionary Systems (2001)
4. Boenn, G., Brain, M., De Vos, M., Fitch, J.P.: Automatic music composition using answer set programming. Theory and Practice of Logic Programming **11**(2-3), 397–427 (2011). https://doi.org/10.1017/S1471068410000530
5. Breiman, L., Friedman, J., Olshen, R., Stone, C.J.: Classification and Regression Trees. Chapman and Hall, New York (1984)
6. Child, R., Gray, S., Radford, A., Sutskever, I.: Generating long sequences with sparse transformers. CoRR **abs/1904.10509** (2019)
7. Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q., Salakhutdinov, R.: Transformer-XL: Attentive language models beyond a fixed-length context. In: Korhonen, A., Traum, D.R., Màrquez, L. (eds.) Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL 2019). pp. 2978–2988. University of Chicago Press (2019)
8. Donahue, C., Mao, H.H., Li, Y.E., Cottrell, G.W., McAuley, J.: Lakhnes: Improving multi-instrumental music generation with cross-domain pre-training. In: Flexer, A., Peeters, G., Urbano, J., Volk, A. (eds.) Proceedings of the 20th International Society for Music Information Retrieval Conference (ISMIR 2019). pp. 685–692 (2019)
9. Eigenfeldt, A., Pasquier, P.: Realtime generation of harmonic progressions using constrained Markov selection. In: Ventura, D., Pease, A., y Pérez, R.P., Ritchie, G., Veale, T. (eds.) Proceedings of the 1st International Conference on Computational Creativity (ICCC 2010). pp. 16–25. computationalcreativity.net (2010)
10. Ferreira, L.N., Lelis, L.H.S., Whitehead, J.: Computer-generated music for tabletop role-playing games. In: Lelis, L., Thue, D. (eds.) Proceedings of the 16th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2020). pp. 59–65. AAAI Press (2020)
11. Foscarin, F., McLeod, A., Rigaux, P., Jacquemard, F., Sakai, M.: ASAP: A dataset of aligned scores and performances for piano transcription. In: Cumming, J., Lee, J.H., McFee, B., Schedl, M., Devaney, J., McKay, C., Zangerle, E., de Reuse, T. (eds.) Proceedings of the 21th International Society for Music Information Retrieval Conference (ISMIR 2020). pp. 534–541 (2020)
12. Gini, C.: On the measure of concentration with special reference to income and statistics. Colorado College Publication, General Series **208**, 73–79 (1936)
13. Goodfellow, I.J., Bengio, Y., Courville, A.C.: Deep Learning. Adaptive computation and machine learning, MIT Press (2016)
14. Hawthorne, C., Huang, C.A., Eck, D.I.D.: Transformer-NADE for piano performances. In: Elliott, L., Dieleman, S., Fiebrink, R., Roberts, A., Engel, J., White, T. (eds.) Proceedings of the NeurIPS Workshop on Machine Learning for Creativity and Design (2018)
15. Hsiao, W., Liu, J., Yeh, Y., Yang, Y.: Compound Word Transformer: Learning to compose full-song music over dynamic directed hypergraphs. In: Guerzhoy,

M., Torrey, L. (eds.) Proceedings of the 11th Symposium on Educational Advances in Artificial Intelligence (EAAI 2021). pp. 178–186. AAAI Press (2021). https://doi.org/10.1609/AAAI.V35I1.16091
16. Huang, C.A., Vaswani, A., Uszkoreit, J., Simon, I., Hawthorne, C., Shazeer, N., Dai, A.M., Hoffman, M.D., Dinculescu, M., Eck, D.: Music Transformer: Generating music with long-term structure. In: Levine, S., Livescu, K., Mohamed, S. (eds.) Proceedings of the 7th International Conference on Learning Representations (ICLR 2019). OpenReview.net (2019)
17. Huang, Y., Yang, Y.: Pop Music Transformer: Beat-based modeling and generation of expressive pop piano compositions. In: Chen, C.W., Cucchiara, R., Hua, X., Qi, G., Ricci, E., Zhang, Z., Zimmermann, R. (eds.) Proceedings of the 28th International Conference on Multimedia (ACM 2020). pp. 1180–1188. ACM (2020)
18. Laitz, S.G.: The Complete Musician: An Integrated Approach to Tonal Theory, Analysis, and Listening. Oxford University Press, Oxford, England, 3rd edn. (2012)
19. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: Levine, S., Livescu, K., Mohamed, S. (eds.) Proceedings of the 7th International Conference on Learning Representations (ICLR 2019). OpenReview.net (2019)
20. MIDI Manufacturers Association: The Complete MIDI 1.0 Detailed Specification. https://midi.org/ (1996)
21. Opolka, S., Obermeier, P., Schaub, T.: Automatic genre-dependent composition using answer set programming. In: Schiphorst, T., Pasquier, P. (eds.) Proceedings of the 21st International Symposium on Electronic Art (ISEA 2015). pp. 627–632. ISEA International, Brighton, UK (2015)
22. Payne, C.: Clara: A neural net music generator. https://christinemcleavey.com/clara-a-neural-net-music-generator, accessed 2024-06-04
23. Payne, C.: MuseNet. https://openai.com/research/musenet (2019), accessed: 2023-06-20
24. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Pearson, 4th edn. (2020)
25. Schön, F.: PAUL-2: A Transformer-Based Algorithmic Composer of Two-Track Piano Pieces. Diploma Thesis, Technische Universität Wien, Institute of Logic and Computation, E192-03 (2023)
26. Schön, F., Tompits, H.: PAUL-2: An upgraded Transformer-based redesign of the algorithmic composer PAUL. In: Basili, R., Lembo, D., Limongelli, C., Orlandini, A. (eds.) Proceedings of the 22nd International Conference of the Italian Association for Artificial Intelligence (AIxIA 2023). Lecture Notes in Computer Science, vol. 14318, pp. 278–291. Springer (2023). https://doi.org/10.1007/978-3-031-47546-7
27. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (eds.) Proceedings of the 30th Annual Conference on Neural Information Processing Systems (NIPS 2017). pp. 5998–6008 (2017)
28. Yu, B., Lu, P., Wang, R., Hu, W., Tan, X., Ye, W., Zhang, S., Qin, T., Liu, T.: Museformer: Transformer with fine- and coarse-grained attention for music generation. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (eds.) Proceedings of the 36th Annual Conference on Neural Information Processing Systems (NeurIPS 2022) (2022)