Compound Word Transformer: Learning to Compose Full-Song Music over Dynamic Directed Hypergraphs

Wen-Yi Hsiao,[1] Jen-Yu Liu,[1], Yin-Cheng Yeh,[1] Yi-Hsuan Yang[2]

[1]Yating Team, Taiwan AI Labs, Taiwan

[2]Academia Sinica, Taiwan

{wayne391, jyliu, yyeh, yhyang}@ailabs.tw

Abstract

To apply neural sequence models such as the Transformers to music generation tasks, one has to represent a piece of music by a sequence of tokens drawn from a finite set of pre-defined vocabulary. Such a vocabulary usually involves tokens of various *types*. For example, to describe a musical note, one needs separate tokens to indicate the note's pitch, duration, velocity (dynamics), and placement (onset time) along the time grid. While different types of tokens may possess different properties, existing models usually treat them equally, in the same way as modeling words in natural languages. In this paper, we present a conceptually different approach that explicitly takes into account the type of the tokens, such as *note* types and *metric* types. And, we propose a new Transformer decoder architecture that uses different feed-forward heads to model tokens of different types. With an expansion-compression trick, we convert a piece of music to a sequence of *compound words* by grouping neighboring tokens, greatly reducing the length of the token sequences. We show that the resulting model can be viewed as a learner over dynamic directed hypergraphs. And, we employ it to learn to compose expressive Pop piano music of full-song length (involving up to 10K individual tokens per song), both conditionally and unconditionally. Our experiment shows that, compared to state-of-the-art models, the proposed model converges 5 to 10 times faster at training (i.e., within a day on a single GPU with 11 GB memory), and with comparable quality in the generated music.

Introduction

To apply neural sequence models such as recurrent neural networks (RNNs) or Transformers (Vaswani et al. 2017) to automatic music composition (a.k.a., symbolic-domain music generation), one has to represent a piece of music as a sequence of tokens drawn from a pre-defined *vocabulary* (Oore et al. 2018). Unlike the case in text, such a vocabulary usually involves tokens of various *types*. For example, to represent a musical score, we may need tokens that describe the content of the musical notes (e.g., pitch and duration), their placement along time, the instrument that plays each note, as well as indicators of metrical events such as the beginning of a new beat, bar (measure), or musical phrase (Wu and Yang 2020). We need such a diverse set of tokens as music is multifaceted; a type alone captures only a certain

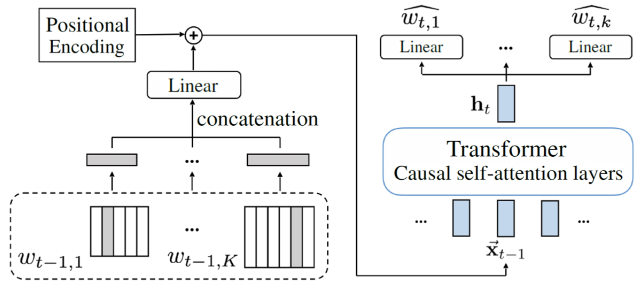


Figure 1: Illustration of the main ideas of the proposed compound word Transformer: (left) *compound word modeling* that combines the embeddings (colored gray) of multiple tokens $\{w_{t-1,k}\}_{k=1}^{K}$, one for each token type k, at each time step $t-1$ to form the input $\vec{\mathbf{x}}_{t-1}$ to the self-attention layers, and (right) *toke type-specific feed-forward heads* that predict the list of tokens for the next time step t at once at the output.

aspect of music (e.g., melody, harmony, rhythm, timbre) and cannot faithfully represent a music piece.

As different types of (musical) tokens may have different properties, modeling the dependency of these tokens might not be the same as modeling words in text. However, to our best knowledge, little work has been done to explicitly account for the heterogeneity of tokens in music. The tokens are mostly treated equally, in the same way as words in text (Huang et al. 2019; Payne 2019; Huang and Yang 2020).

We are therefore motivated to study in this paper whether we can improve sequence modeling of music by highlighting the role of token types. Our first proposal is to *customize the prediction heads for tokens of different types*. Specifically, using the Transformer as the main architecture of the underlying sequence model, we approach this by using different feed-forward heads for tokens of different types.

Our second proposal is to group consecutive and related tokens in a token sequence into "compound words," and then *perform sequence modeling over* the resulting *sequence of compound words*. This is to capture the co-occurrence relationship of tokens—e.g., to generate a new musical note, we may need at least two consecutive tokens to indicate its pitch and duration; to change the tempo in the middle of a piece of music, we need a token to indicate the target tempo value, and an co-occurring time-related token to indicate the

	Representation	Model	window	Voc. size	Data type
Music Transformer (Huang et al. 2019)	MIDI-like	Transformer	2,048	388	Classical performance
MuseNet (Payne 2019)	MIDI-like*	Transformer	4,096	N/A	Multi-track MIDI
LakhNES (Donahue et al. 2019)	MIDI-like*	Transformer-XL	512	630	Multi-track MIDI
TR autoencoder (Choi et al. 2020)	MIDI-like	Transformer	2,048	388	Classical performance
Pop Music TR (Huang and Yang 2020)	REMI	Transformer-XL	512	332	Pop piano performance
Transformer VAE (Jiang et al. 2020)	MIDI-like	Transformer	128	47	Pop lead sheets
Guitar Transformer (Chen et al. 2020)	REMI*	Transformer-XL	512	221	Guitar tabs
Jazz Transformer (Wu and Yang 2020)	REMI*	Transformer-XL	512	451	Jazz lead sheets
MMM (Ens and Pasquier 2020)	MIDI-like*	Transformer	2,048	>442	Multi-track MIDI
This work	CP	linear Transformer	5,120	350	Pop piano performance

Table 1: A comparison of existing Transformer-based models and the proposed one for automatic music composition. The representations marked with * are extensions of either MIDI-like (Oore et al. 2018) or REMI (Huang and Yang 2020).

time of the tempo change. Under the proposed compound-word modeling, the individual tokens (e.g., pitch and duration) are still predicted separately with different heads. Yet, instead of predicting them at different time steps, we predict multiple tokens of various types *at once* in a single time step. The token embeddings of the tokens predicted at the current step are then combined and fed as the input for the next time step. Namely, the self-attention is computed over combined embeddings of individual tokens of a compound word.

From a theoretical point of view, the proposed model can be interpreted as a learner over discrete-time *dynamic directed hypergraphs* (Kazemi et al. 2020). Here, a graph consists of nodes that each corresponds to a token in our vocabulary. A sequence of tokens can then be viewed as a sequence of edges (each connecting two nodes), or a *walk*, over this graph. A sequence of compound words, in contrast, can be viewed as a sequence of *hyperedges* (each connecting multiple nodes) (Feng et al. 2019), over the same graph. We discuss this at greater length later in the paper.

We refer to the proposed representation as the *compound word representation*, or CP for short. CP can be considered as an extension of existing representations, with the following additional merits. First, it allows for fine-grained, type-specific control over the prediction heads. For example, we can now use different loss functions, sampling policies, and token embedding sizes for different token types.

Second, as a compound word represents multiple tokens at once, it requires much less time steps to generate a music piece using compound words. Namely, the sequence length of the same music piece is much shorter in CP than in existing representations. As the computational complexity of a Transformer is related to the sequence length (Vaswani et al. 2017), this makes training and inference faster, and may facilitate learning the long-range dependency in music.[1]

Finally, the sequence length in CP is determined by the number of compound words in a sequence, not by the number of individual tokens per compound word. Therefore, it is possible to add new token types (by adding the corresponding feed-forward head) to increase the expressivity of the representation, without increasing the sequence length. This makes it easy to extend to underlying representation, though we do not explore this potential in this work.

For performance study, we consider generating expressive Pop piano music at full-song scale in both the unconditional setting (i.e., from scratch) and conditional setting (i.e., generating the piano arrangement given the lead sheet). This involves modeling fairly long music sequences for up to 10K individual tokens each. We show that, with CP, we are able to train a linear Transformer decoder (Katharopoulos et al. 2020) with music quality similar to that of strong baselines, with faster training and inference time. We provide audio examples and open source the project at a GitHub repo.[2]

Related Work

Both language and music have principles governing the organization of discrete structural elements (e.g., words or musical notes) into sequences (Patel 2003). As such, the Transformers, which have been firstly shown to work well for text generation (Child et al. 2019; Keskar et al. 2019), have been increasingly applied to music generation in recent years, by treating music pieces as sequences of discrete tokens akin to text words. We list some related papers in Table 1.

Table 1 shows that most existing work adopt a music representation derived from either MIDI-like (Oore et al. 2018) or REMI (Huang and Yang 2020), with possible addition of track- or structure-related tokens. MIDI-like and REMI differ mainly in how the advance of time is represented: the former uses [time_shift] tokens to mark the time interval (in absolute time) between note-related tokens, whereas the latter assumes symbolic timing and uses [bar] and [position] tokens to place tokens on a metrical grid that uniformly divides a bar into a certain number of positions. Neither MIDI-like nor REMI groups the tokens by token types.[3]

Existing work also differs in the length of the *attention window* (see the methodology section for definition) and vocabulary size (which is data- and task-dependent). To our knowledge, our work represents the first one to consider Pop music modeling at full-song scale (involving 10k tokens per song), and to use the recently-proposed linear Transformer (Katharopoulos et al. 2020) as the model backbone.

[1] For example, we can study whether the proposed model creates music with better "structureness," or long-term repetitions (Wu and Yang 2020; Jhamtani and Berg-Kirkpatrick 2019) in the future.

[2] https://github.com/YatingMusic/compound-word-transformer

[3] Upon paper completion, we noticed an early but preliminary attempt of grouping tokens by (Hawthorne et al. 2018b).

Methodology

Background

For sequence modeling, we need a conversion function $g(\cdot)$ that converts a music piece $\mathcal{X}$ to a time-ordered sequence of symbolic elements $\mathcal{S} = g(\mathcal{X}) = \{w_1, w_2, \dots, w_T\}$, where T denotes the resulting sequence length. Given a number of such sequences, we train a neural sequence model with an architecture such as the Transformer decoder to learn to generate new sequences $\mathcal{S}'$. We then use a deterministic inverse function $g^{-1}(\cdot)$ to get a new music piece from such a generated sequence, namely $\mathcal{X}' = g^{-1}(\mathcal{S}')$. There can be different algorithms to implement the conversion function and its inverse, leading to numerous possible sequence representations of the same music piece, e.g., $\mathcal{S}_{\text{MIDI-like}} = g_{\text{MIDI-like}}(\mathcal{X})$ and $\mathcal{S}_{\text{REMI}} = g_{\text{REMI}}(\mathcal{X})$. Different conversion functions (or sequence representations) assume different vocabulary sizes M, so $\mathcal{S}_{\text{MIDI-like}}$ and $\mathcal{S}_{\text{REMI}}$ differ in both T and M.

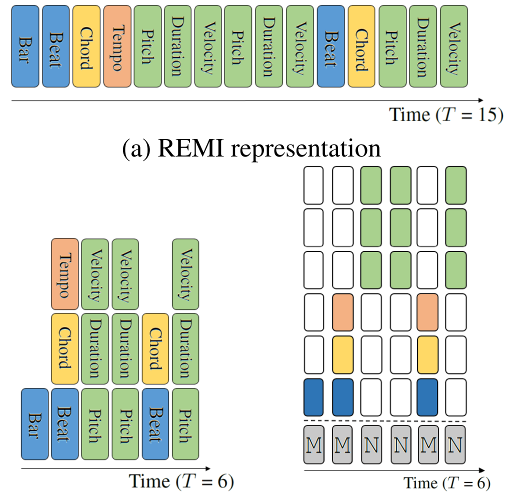
A Transformer decoder comprises a stack of *self-attention* layers and a stack of *feed-forward* layers. The self-attention layers operate on a fixed-length sub-sequence of $\mathcal{S}$ to learn the dependency among the elements. The length of such a sub-sequence, a.k.a., the *attention window*, denoted as N, is usually much smaller than T, as N directly affects the space complexity of the model. For the vanilla Transformer (Vaswani et al. 2017) and its faster variant Transformer-XL (Dai et al. 2019), it is $\mathcal{O}(N^2M)$; for the linear Transformer (Katharopoulos et al. 2020), it is $\mathcal{O}(NM)$.

Individual Tokens vs Compound Words

In this paper, we refer to the elements in either $\mathcal{S}_{\text{MIDI-like}}$ or $\mathcal{S}_{\text{REMI}}$ as the *individual tokens*. They are drawn from a predefined vocabulary $\mathcal{V} = \{1, \dots, M\}$. As mentioned in the introduction, each token is associated with a *type* defined in the type set, $\mathcal{K} = \{1, \dots, K\}$. We can partition $\mathcal{V}$ into K subsets by token group, i.e., $\{\mathcal{V}_k\}_{k=1}^{K}$.

We propose to convert a sequence of tokens (e.g., $\mathcal{S}_{\text{REMI}}$) into a sequence of compound words $\mathcal{S}_{\text{CP}}$ with the following procedure. First, *neighboring tokens that define a musical event together are grouped into a super token*, i.e., placed on the same time step, as illustrated in Figures 2(a)–(b). A musical event here can be a *note* related one, i.e., to create a new musical note, or a *metrical* related one, e.g., to mark the beginning of a new beat, or a new bar. For example, in REMI, a note is created by consecutive tokens of [pitch], [duration], and [velocity], which are grouped in CP. And, a tempo or chord change in REMI takes place only at beat times, so we also group [beat], [chord] and [tempo]. Accordingly, the model has to make multiple predictions (i.e., generate multiple tokens) at each time step.

Second, we *fill the missing token types per time step with "[ignore]" tokens*, so that at each step there are consistently K tokens to be predicted, as illustrated in Figure 2(c). This is to make computational modeling feasible, as otherwise the shape and meaning of the target output at each time step would be uncertain. In other words, a *compound word* is composed of a list of K tokens, each drawn from the corresponding subset $\mathcal{V}_k \cup$ [ignore], that are placed on the same time step t. Formally, $\mathcal{S}_{\text{CP}} = g_{\text{CP}}(\mathcal{X}) = \{cp_t\}_{t=1}^{T_{\text{cp}}}$, in which



(a) REMI representation

(b) Tokens grouped

(c) Compound words

Figure 2: An example illustrating the conversion from a sequence of REMI tokens (Huang and Yang 2020) into a (shorter) sequence of compound words. A compound word comprises a number of grouped tokens and the [ignore] tokens, which are colored white in (c), as well as a family token (`N`: note-related or `M`: metric-related). Best seen in color.

$cp_t = \{w_{t,1}, \cdots, w_{t,K}\}$. We view this conversion function $g_{\text{CP}}(\cdot)$ as performing an *expansion-compression trick*, as the original sequence is firstly expanded to a sequence of KT_{CP} individual tokens, and then compressed to a sequence of T_{CP} compound words; in general $T_{\text{CP}} < T_{\text{REMI}} < KT_{\text{CP}}$.

To facilitate modeling the CP, we further partition the type set $\mathcal{K}$ into F *families*. For example, if $\mathcal{K}$ can be partitioned into two families, the *note* family $\mathcal{K}_{\text{N}}$ and *metric* family $\mathcal{K}_{\text{M}}$ (marked as '`N`' and '`M`' in Figure 2(c)), we would have $\mathcal{K} = \mathcal{K}_{\text{N}} \cup \mathcal{K}_{\text{M}}$, and $\mathcal{K}_{\text{N}} \cap \mathcal{K}_{\text{M}} = \emptyset$. Each compound word cp_t is associated with a *family token* f_t. For a metric-related cp_t, we would have $w_{t,k} =$ [ignore], for $k \in \mathcal{K}_{\text{N}}$. Similarly, for a note-related cp_t, $w_{t,k} =$ [ignore], for $k \in \mathcal{K}_{\text{M}}$.

Combining Token Embeddings of Adaptive Sizes

As input to Transformers, an element in a sequence is represented by an *embedding* vector, $\mathbf{x}_t \in \mathcal{R}^d$, and then added with a positional embedding vector (Ke, He, and Liu 2020). In CP, we propose to form an embedding vector for a compound word cp_t by combining the embedding vectors $\mathbf{p}_{t,k}$ of the composing tokens $w_{t,k}$, as well as an embedding vector $\mathbf{q}_t$ associated with the family token f_t. Specifically, we combine the vectors by firstly concatenating them, and then linearly projecting the resulting long vector to a d-dimensional vector with a projection matrix $\mathbf{W}_{\text{in}}$. Namely,

$$
\begin{aligned}
\mathbf{p}_{t,k} &= \text{Embedding}_k(w_{t,k})\,,\; k = 1, ..., K\,, \\
\mathbf{q}_t &= \text{Embedding}_{\mathcal{F}}(f_t)\,, \\
\mathbf{x}_t &= \mathbf{W}_{\text{in}}\,[\mathbf{p}_{t,1} \oplus ... \oplus \mathbf{p}_{t,K} \oplus \mathbf{q}_t]\,, \\
\vec{\mathbf{x}}_t &= \text{Positional Encoding}(\mathbf{x}_t)\,,
\end{aligned}
\tag{1}
$$

where $\oplus$ denotes vector concatenation, and $\text{Embedding}_k(\cdot)$ and $\text{Embedding}_{\mathcal{F}}(\cdot)$ involve the use of lookup tables.

In essence, $\mathbf{x}_t$ can be considered as a *compressive* representation of the composing tokens $w_{t,k}$ and family token f_t. We note the action of compressing the embeddings is reminiscent of the main idea of the Compressive Transformer (Rae et al. 2020), which proposes to compresses past memories beyond the attention window for long-range sequence learning. Unlike it, we compress the memories *within* the attention window defined over the individual tokens.

A main merit of CP is that we can customize the settings for different token types. Being inspired by the *adaptive word representation* (Baevski and Auli 2018), we use different embedding sizes d_k for tokens of different types, i.e., $\mathbf{p}_{t,k} \in \mathcal{R}^{d_k}$. We basically use larger d_k for token types with larger vocabulary size $|\mathcal{V}_k|$. See Table 3 for details.

Multi-head Output Module

A main proposal of our work is to use different feed-forward heads for tokens of different types in a Transformer. Specifically, we have $(K+1)$ heads in total, one for each token type $\mathcal{V}_k$ and an additional one for the token family $\mathcal{F}$.

Instead of working on the $K+1$ heads at the same time, we devise a *two-stage* setting that predicts the family token first, and then the remaining tokens given the family token. Specifically, at the t-th time step, the feed-forward procedure can be summarized as:

$$\begin{aligned}
\mathbf{h}_t &= \text{Self-attn}\left(\vec{\mathbf{x}}_{t-1}\right) , \\
\widehat{f_t} &= \text{Sample}_{\mathcal{F}}\left(\text{softmax}(\mathbf{W}_{\mathcal{F}}\mathbf{h}_t)\right) , \\
\mathbf{h}_t^{\text{out}} &= \mathbf{W}_{\text{out}}\left[\mathbf{h}_t \oplus \text{Embedding}_{\mathcal{F}}(\widehat{f_t})\right] , \\
\widehat{w_{t,k}} &= \text{Sample}_k\left(\text{softmax}(\mathbf{W}_k\mathbf{h}_t^{\text{out}})\right) ,\ k = 1, ..., K ,
\end{aligned} \tag{2}$$

where $\mathbf{W}_{\mathcal{F}}$ and $\{\mathbf{W}_k\}_{k=1}^{K}$ are the $K+1$ feed-forward heads, Self-attn$(\cdot)$ the causal self-attention layers, and Sample$(\cdot)$ a sampling function. We empirically find that this two-stage setting makes it easier for the model to predict $w_{t,k} =$ [ignore], for k not in the target family $\mathcal{K}_{\widehat{f_t}}$.

Figure 1 illustrates Eqs. (1)–(2) in work, omitting the first-stage part at the output for $\widehat{f_t}$ due to space limit.

Adaptive Sampling Policy

At inference time, we use stochastic temperature-controlled sampling (Holtzman et al. 2020) to avoid degeneration and to increase diversity. With CP, we employ different sampling policies $\text{Sample}_k(\cdot)$ for different token types; see Table 3.

Graph Interpretation

We discuss the proposed model from a graph-theoretical point of view below. Given a vocabulary of tokens, we can construct a fully-connected *static graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ (Kivelä et al. 2014) comprising nodes $\mathcal{V} = \{1, \ldots, M\}$ and edges $\mathcal{E} = \mathcal{V} \times \mathcal{V}$. Each node corresponds to an individual token in our vocabulary. This way, a token sequence $\mathcal{S}_{\text{MIDI-like}}$ or $\mathcal{S}_{\text{REMI}}$ can be viewed as a sequence of edges (each connecting two nodes), or a *walk*, over this graph.

In CP, the vocabulary (and accordingly the graph) is augmented with a set of *special tokens*, denoted as $\mathcal{V}^*$, that includes for example type-specific [ignore] tokens and family tokens. And, a compound word consists of $K+1$ nodes, one from each of the K types and an additional one from the set of family tokens. A sequence of compound words, namely $\mathcal{S}_{\text{CP}}$, therefore, involves transitions from $K+1$ nodes to another $K+1$ nodes per time step. Such a transition can be viewed as a directed *hyperedge* (Feng et al. 2019; Jiang et al. 2019), that connects at once $K+1$ source nodes (e.g., cp_{t-1}) to $K+1$ target nodes (cp_t). It is directed because the order of the nodes matters (i.e., from $t-1$ to t).

A sequence of compound words also forms a *dynamic directed hypergraph* (Kazemi et al. 2020): $\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_T\}$, where $\mathcal{G}_t = (\mathcal{V}, \mathcal{E}_t)$. Starting from an empty graph with no edges, at each time step $t > 1$ we add a new directed hyperedge, labeled with the time step t, connecting in total $2K+2$ nodes. In practice, we have a [BOS] token (beginning of sequence) and [EOS] token (end of sequence), so the hyperedge at $t = 1$ and $t = T$ connects to only $K+2$ nodes.

A neural model for graphs, or a *graph neural network* (GNN), can be regarded as an encoder-decoder pair (Kazemi et al. 2020; Rossi et al. 2020), where an *encoder* is a function that maps from a graph $\mathcal{G}$ to node embeddings $\mathbf{z}_i, i = 1 \ldots M$, and a *decoder* takes as input one ore more node embeddings and makes a prediction based on these, e.g., node classification or edge prediction. The proposed CP Transformer can therefore be regarded as a learner over dynamic directed hypergraphs, as at each time step t it manages to predict the next hyperedge to be added (i.e., $\widehat{w_{t,k}}$ and $\widehat{f_t}$) based on the node embeddings updated from $\mathcal{G}_{<t} = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_{t-1}\}$, or the collection of input embeddings $\mathbf{x}_{<t} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{t-1}\}$ marked with positional embeddings (i.e., edge labels on the directed hyperedges).

We note that, while we introduce the proposed methods in the context of music modeling, the idea of compound words is generic and may be applicable to sequences seen in other data domains, when multiple tokens (i.e., a hyperedge) are needed to represent a single event, entity, or object.

Implementation

To test the effectiveness of the proposed methods, we implement a CP Transformer that learns to generate Pop piano music with human performance characteristics such as expressive variations in velocity (i.e., the force with which a note is played, which is related to loudness) and tempo (Oore et al. 2018; Lerch et al. 2019). We consider Pop piano for its richness and expressivity, and for offering a direct performance comparison with the Pop Music Transformer (Huang and Yang 2020) (see Table 1).

Specifically, we consider both the **conditional** and **unconditional** generation tasks. In the former, a *lead sheet* (i.e., a melody line and an accompanying sequence of chord labels) is given, and the model has to generate a piano performance according to that. In the latter, the model generates a piano performance of full-song length from scratch freely.

We intend to compare CP with REMI in our evaluation. We provide the implementation details below.

Task	Repre.	#words (T) mean ($\pm$ std)	max
Conditional	REMI	6,432 ($\pm$ 1,689)	10,240
	CP	3,142 ($\pm$ 821)	5,120
Unconditional	REMI	4,873 ($\pm$ 1,311)	7,680
	CP	2,053 ($\pm$ 580)	3,584

Table 2: Statistics of the number (#) of words (i.e., tokens in REMI; compound words in CP) per song in the training set.

Dataset

We collect the audio files of 1,748 pieces of Pop piano from the Internet. The average length of the songs is about 4 minutes, and we have about 108 hours in total. All the songs are in 4/4 time signature (four beats per bar). We convert each song (an audio) into a symbolic sequence as follows.

- **Transcription**: We use the state-of-the-art RNN model for automatic piano transcription, "Onset and Frames" (Hawthorne et al. 2018a), to estimate the pitch, onset and offset time, and velocity of the musical notes from audio.
- **Synchronization**: To get symbolic timing from the original wall clock time, we use the RNN-based model available in the Python package `madmom` (Böck et al. 2016) to estimate the downbeat and the beat positions, which represent the state-of-the-art for the task. Then, we interpolate 480 ticks between two adjacent beats, and map the absolute time into its according tick. By doing so, we can keep tiny offset. Lastly, we infer the tempo changes from the time interval between adjacent beats.
- **Quantization**: We quantize the tempo, velocity, duration and the beat positions to reduce the size of the vocabulary. For example, we set the 16-th note as our basic time unit. See Table 3 for the number of tokens per type.
- **Analysis**: For the conditional generation task, we estimate the melody notes and chord symbols from the transcription result to form the lead sheets. Specifically, we develop an in-house rule-based chord recognition algorithm[4] to recognize 12 roots and 7 chord qualities. We use the "Skyline algorithm" (Uitdenbogerd and Zobel 1999) to extract the melodies. And, as a lead sheet is usually of coarser time resolution, we quantize the chord symbols and melody notes to the 4-th notes (i.e., beat times).

We randomly hold out 50 songs for testing, and use the remaining for training the Transformers.

Vocabulary

To represent the content of a piano performance, the basic setting employs tokens of six types: three note-related types [pitch], [duration], [velocity], and three metric-related types [position/bar], [tempo], [chord]. The specific vocabulary is task-dependent and is introduced below.

Conditional generation—We additionally use [track] tokens to mark whether it is the *lead sheet* track (i.e., the condition) or the *piano* track (the track to be generated). While the piano track (i.e., the sub-sequence after the [track=piano] token) involves all the six types of tokens mentioned above, the lead sheet track only involves the use of composition-related tokens [position/bar], [chord], [pitch], [duration], not performance-related tokens [velocity], [tempo]. In CP, we have three family tokens, [family=track], [family=note], [family=metric]. Moreover, we have type-specific [ignore] tokens and an additional [conti] token for the beat positions having no tempo or chord changes.

Unconditional generation—This task only concerns with the piano track so we do not need the [track] tokens. But, as it concerns with full-song generation, we add an [EOS] token to signify the end of a sequence. We view it as a family token, so there are three possible family tokens here: [family=EOS], [family=note], [family=metric].

Details of the adopted representations are shown in Tables 2 and 3. Table 2 compares the sequence length T of REMI and CP. We can see that $\mathcal{S}_{\text{CP}}$ is much shorter than $\mathcal{S}_{\text{REMI}}$, especially under the conditional task.[5] Table 3 displays the size of each vocabulary subset $\mathcal{V}_k$. We see that CP and REMI have similar total vocabulary size M. REMI does not use the family tokens (except for [EOS]) and special tokens.

Repre.	Token type	Voc. size $\lvert\mathcal{V}_k\rvert$	Embed. size (d_k)	Sample$_k(\cdot)$ τ	ρ
CP	[track]	2 (+1)	3	1.0	0.90
	[tempo]	58 (+2)	128	1.2	0.90
	[position/bar]	17 (+1)	64	1.2	1.00
	[chord]	133 (+2)	256	1.0	0.99
	[pitch]	86 (+1)	512	1.0	0.90
	[duration]	17 (+1)	128	2.0	0.90
	[velocity]	24 (+1)	128	5.0	1.00
	[family]	4	32	1.0	0.90
	total	341 (+9)	—	—	—
REMI	total	338	512	1.2	0.90

Table 3: Details of the CP representation in our implementation, including that of the sampling policy (τ-tempered top-ρ sampling). For the vocabulary size, the values in the parentheses denote the number of special tokens such as [ignore].

Model Settings

For the backbone architecture of our model, we employ the linear Transformer (Katharopoulos et al. 2020),[6] as its complexity is a linear function of the length of the attention window N. Moreover, we set N equal to the sequence length T for our model. That is, *no segmentation* over the training sequences is done, and thereby *all the tokens in a sequence can be accessed* by our model under causal masking, without using tricks such as memory caching (Dai et al. 2019) or memory compression (Rae et al. 2020). We refer to our model as **CP+linear** in what follows.

For the **baselines**, we employ the Pop Music Transformer

[4] https://github.com/joshuachang2311/chorder

[5] We set an upper limit of the number of elements per sequence (e.g., 10,240 tokens in REMI) and remove overly long songs, which amounts to removing 25–88 songs from the training set depending on the task and the adopted representation.

[6] https://github.com/idiap/fast-transformers

Task	Representation + model@loss	Training time	GPU memory	Inference (/song) time (sec)	tokens (#)	Matchness melody	chord
Conditional	Training data	—	—	—	—	0.755	0.838
	Training data (randomized)	—	—	—	—	0.049	0.239
	REMI + XL@0.44	3 days	4 GB	88.4	4,782	0.872	0.785
	REMI + XL@0.27	7 days	4 GB	91.5	4,890	0.866	0.800
	REMI + linear@0.50	3 days	17 GB	48.9	4,327	0.779	0.709
	CP + linear@0.27	0.6 days	10 GB	29.2	18,200	0.829	0.733
Unconditional	REMI + XL@0.50	3 days	4 GB	139.9	7,680	—	—
	CP + linear@0.25	1.3 days	9.5 GB	19.8	9,546	—	—

Table 4: Quantitative evaluation result of different models. REMI+XL represents a re-implementation of the state-of-the-art Pop Music Transformer (Huang and Yang 2020), while CP+linear stands for the proposed CP Transformer.

(Huang and Yang 2020), which is open-source and stands for a state-of-the-art for unconditional music composition.[7] This **REMI+XL** model adopts the REMI representation and uses Transformer-XL (Dai et al. 2019) as the model backbone. As its complexity grows quadratically with N, we set $N = 512$, following (Huang and Yang 2020).

Moreover, we consider one more baseline that replaces Transformer-XL by linear Transformer, using also $N = T$, to offer a sensible performance comparison between CP and REMI. We refer to this variant as **REMI+linear**.

We use 12 self-attention layers each with 8 attention heads for all the models for fair comparison. The model hidden size and inner layer of the feed-forward part are set to 512 and 2,048, respectively. For the token embedding size d, we fix it to 512 for REMI, following (Huang and Yang 2020). For CP, we set it adaptively based on the vocabulary size of each token type, as shown in Table 3. For sampling, we employ the "nucleus sampling" (Holtzman et al. 2020), a stochastic method that samples from the smallest subset of tokens whose cumulative probability mass exceeds a threshold $\rho \in [0, 1]$. Before sampling, we reshape the probability distribution of the tokens (e.g., softmax($\mathbf{W}_k \mathbf{h}_t^{\text{out}}$)) through "temperature" (Ackley, Hinton, and Sejnowski 1985), with the temperature parameter $\tau > 0$. As Table 3 also shows, we use different ρ and τ for different token types. For example, we use a large τ to encourage diverse velocity values.

The conditional generation task can be approached with a sequence-to-sequence model, since we have paired data of lead sheets and piano performances (i.e., the former is extracted automatically from the latter). Instead of adding a Transformer encoder (as done in (Choi et al. 2020)) to realize this, we use the encoder-free "Prefix LM" method of the Google's "T5" model (Raffel et al. 2020), and run a single Transformer over an *interleaved* sequence of lead sheets and piano performances. Specifically, a sequence of lead sheet and the corresponding target sequence of piano performance are integrated into one sequence bar after bar. That is, the integrated sequence would have the form of $\{\ldots,$ [bar], [track=leadsheet], (content of the lead sheet for a bar), [track=piano], (content of the piano for the same bar), [bar], (content of the two tracks of the next bar) $\ldots\}$. This makes it easy to learn the dependency of the two tracks, and to impose the pre-given lead sheet at inference time.

[7]https://github.com/YatingMusic/remi

Quantitative Evaluation

The experiments hereafter are conducted in the interest of a resource-constrained scenario, assuming that we only have a single GPU with 11 GB memory and are only willing to train a model for 3 days. We conjecture that this makes sense for most middle-size academic labs worldwide. Yet, to have an idea of the model performance when more resources are available, we include to the evaluation of the conditional task two settings exceeding such a specification.

We firstly compare the efficiency of the models in terms of training time, inference time, and GPU memory usage, under the conditional setting. The average result over the 50 held-out test songs is shown in Table 4.

GPU memory usage. Table 4 shows that both CP+linear and REMI+XL require <11 GB GPU memory for training. Accordingly, in our implementation, we train them (separately) on an NVIDIA RTX 2080 Ti GPU (with 11GB memory). In contrast, REMI+linear requires 17 GB GPU memory, so we train it on a TITAN GPU with 24 GB memory.

Training time. We see that REMI-based models require much longer clock time to reach a low training loss. While it takes nearly 7 days for REMI+XL to reduce the negative log-likelihood (NLL) of the training data to 0.27, it takes only 0.6 days for CP+linear to reach the same NLL. Such a training efficiency is desirable (especially given that it is on a single 2080 Ti GPU), as it makes further extensions and modifications of the model easy and affordable.

Inference time. CP+linear is remarkably fast, taking on average <30 seconds to complete the conditional generation of a song. As a song in our dataset is about 4 minutes, this is much faster than real time. In contrast, REMI+XL and REMI+linear are about 3x and 1.7x slower, respectively. CP+linear is fast for it generates in total 8 individual tokens (of different types) at once each time step.

Table 4 also compares the efficiency of REMI+XL and CP+linear under the unconditional setting, for which we generate also 50 songs (from scratch) and report the average inference time. We see that CP+linear is even faster here, requiring only <20 seconds to create a new song at full-song length. In contrast, REMI+XL is on average 7x slower.

Next, we compare the performance of the models in terms of two objective metrics, also under the conditional setting. As the goal is to generate a song given a lead sheet, we can measure whether the generated song has a melody line and

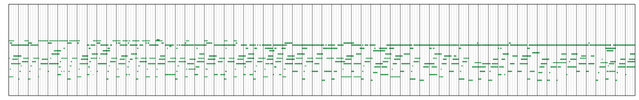
Repre. + model@loss	F	R	H	C	O
REMI + XL@0.44	4.05	3.12	3.38	3.55	3.31
REMI + XL@0.27	**4.29**	**3.14**	**3.70**	**3.64**	**3.35**
REMI + linear@0.50	4.03	3.09	3.48	3.46	3.29
CP + linear@0.27	4.09	3.13	3.50	3.31	3.08

(a) Conditional generation

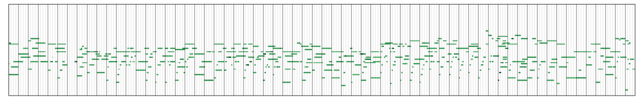
Repre. + model@loss	R	H	S	O
REMI + XL@0.50	3.11	3.46	2.91	3.03
CP + linear@0.22	**3.33**	**3.68**	**3.11**	**3.34**

(b) Unconditional generation

Table 5: Result of subjective evaluation (Fidelity, Richness, Humanness, Correctness, Structureness, Overall).



(a) REMI+XL



(b) CP+linear

Figure 3: Piano-rolls of middle 64 bars of random generated pieces of two models in the unconditional setting. We see richer and diverse content in the result of CP+linear.

chord progression similar to that in the given condition, and take that as a figure of merit. (In contrast, proper objective evaluation of unconditional generation models remains an open issue (Yang and Lerch 2020; Dong et al. 2020; Wu and Yang 2020).) Specifically, we consider:

- **Melody matchness**. We represent the lead sheet and the correspondingly generated piano both in the REMI format and compute the bar-wise *longest common sub-sequence* (LCS) of the two resulting sequences $\mathcal{S}^{\text{LS}}_{\text{REMI}}$ and $\widehat{\mathcal{S}}^{\text{piano}}_{\text{REMI}}$. When two notes (each from the two sequences) have the same pitch and close onset time (within the 8-th note), we consider that as a match. We divide the length of the LCS by the number of [pitch] tokens in $\mathcal{S}^{\text{LS}}_{\text{REMI}}$ (i.e., the number of target melody notes) of that bar, and take the average value of such a ratio across all the bars of a song as a simple measure of melody matchness.
- **Chord matchness**. The *chroma vector* (Fujishima 1999) represents a short-time fragment of music by the distribution of energy across the 12 pitch classes (`C`, `C#`, etc) and offers a simple way to evaluate the harmonic similarity between two fragments. We calculate the segment-wise cosine similarity between the chroma vector representing each chord label of a lead sheet (which would be binary-valued) and the chroma vector of the correspondingly generated piano segment (normalized by the maximum value so it is $\in [0,1]^{12}$), and treat the average value across time as a measure of chord matchenss.

Table 4 shows that the evaluated models all have matchness close to that of the training set, and much higher than that of the random baseline (i.e., the average matchness between a lead sheet and a random song from the test set). This suggests, while CP+linear is easier and faster to train than REMI+XL, they may generate music of similar quality. We further investigate this through a user study, which directly assesses the perceptual quality of the generated music.

Qualitative Evaluation

We devise an online questionnaire that solicits anonymous response to the music generated by different models for both the conditional and unconditional settings. For the former, we present excerpts of 32 bars taking from one-third location of the music. For the latter, we present the full songs (i.e., when an [EOS] token is generated).[8] Our intention is to investigate whether CP+linear and REMI+XL indeed generate music of similar perceptual qualities.

The generated music is rendered into audio with a piano synthesizer using a free, non-professional grade sound font. Each batch comprises the result of the evaluated models in random order. A subject has to rate the music for three random batches for each setting separately, in terms of the following aspects on a five-point Likert scale. 1) **Fidelity**: is the conditionally generated piece similar to the reference, from which the condition lead sheet was taken from? 2) **Richness**: diversity and interestingness. 3) **Humanness**: does the piece sound like expressive human performances? 4) **Correctness**: perceived absence of composing or playing mistakes. 5) **Structureness**: whether there are structural patterns such as repeating themes or development of musical ideas. 6) **Overall**. As the music can be long, the questionnaire may take around 30 mins to complete.

Table 5 shows the average result from 18 subjects. We see that REMI+XL performs the best in the conditional setting, yet with only moderate performance gap between the models.[9] In contrast, CP+linear performs (slightly) better consistently across the four metrics in the unconditional setting, suggesting it a powerful alternative to REMI+XL.

Conclusion

In this paper, we have presented a new variant of the Transformer that processes multiple consecutive tokens at once at a time step. Each individual token is associated with a token type, which is exploited by the model to customize its input and output modules. The proposed model achieves sequence compression by integrating the embeddings of the tokens, which can be seen as forming a hyperedge over a dynamic graph. We show that the new Transformer works remarkably well for modeling music, creating full-song piano of comparable perceived quality with a competing Transformer-XL based model in much shorter training and inference time.

[8] It turns out that the REMI+XL model seldom generates [EOS] tokens even when the music is already quite long (e.g., 8 minutes), so we stop it each time when it has generated 7,680 tokens.

[9] In the conditional setting, the global structure of the song to be generated is fairly outlined in the given condition (i.e., the melody). Thus, it seems sufficient for models to learn from short segments.

Acknowledgements

We are grateful to our interns at the Taiwan AI Labs, Joshua Chang for developing the symbolic-domain chord recognition algorithm, and Yu-Hua Chen and Hsiao-Tzu Hung for helping organize the PyTorch code. We also thank the anonymous reviewers for their valuable comments.

Ethics Statement

Research on automatic music generation may infringe copyright laws and may raise concerns regarding the role of human musicians in the future. Cares have to be given regarding the fair use of existing musical material for model training, and the potential concern of "deepfaking" an existing artist's style in computer-generated music.

References

Ackley, D. H.; Hinton, G. E.; and Sejnowski, T. J. 1985. A learning algorithm for Boltzmann machines. *Cognitive Science* 9(1): 147–169.

Baevski, A.; and Auli, M. 2018. Adaptive input representations for neural language modeling. *arXiv preprint arXiv:1809.10853* .

Böck, S.; Korzeniowski, F.; Schlüter, J.; Krebs, F.; and Widmer, G. 2016. Madmom: A new Python audio and music signal processing library. In *Proc. ACM Multimedia*, 1174–1178.

Chen, Y.-H.; Huang, Y.-S.; Hsiao, W.-Y.; and Yang, Y.-H. 2020. Automatic composition of guitar tabs by Transformers and groove modeling. In *Proc. Int. Soc. Music Information Retrieval Conf.*

Child, R.; Gray, S.; Radford, A.; and Sutskever, I. 2019. Generating long sequences with sparse Transformers. *arXiv preprint arXiv:1904.10509* .

Choi, K.; Hawthorne, C.; Simon, I.; Dinculescu, M.; and Engel, J. 2020. Encoding musical style with transformer autoencoders. In *Proc. Int. Conf. Machine Learning*.

Dai, Z.; Yang, Z.; Yang, Y.; Carbonell, J.; Le, Q.; and Salakhutdinov, R. 2019. Transformer-XL: Attentive language models beyond a fixed-Length context. In *Proc. Annual Meeting of the Association for Computational Linguistics*, 2978–2988.

Donahue, C.; Mao, H. H.; Li, Y. E.; Cottrell, G. W.; and McAuley, J. 2019. LakhNES: Improving multi-instrumental music generation with cross-domain pre-training. In *Proc. Int. Soc. Music Information Retrieval Conf.*, 685–692.

Dong, H.-W.; Chen, K.; McAuley, J.; and Berg-Kirkpatrick, T. 2020. MusPy: A toolkit for symbolic music generation. In *Proc. Int. Soc. Music Information Retrieval Conf.*

Ens, J.; and Pasquier, P. 2020. MMM- Exploring conditional multi-track music generation with the Transformer. *arXiv preprint arXiv:2008.06048* .

Feng, Y.; You, H.; Zhang, Z.; Ji, R.; and Gao, Y. 2019. Hypergraph neural networks. In *Proc. AAAI*, 3558–3565.

Fujishima, T. 1999. Realtime chord recognition of musical sound: A system using common Lisp. In *Proc. International Computer Music Conf.*, 464–467.

Hawthorne, C.; Elsen, E.; Song, J.; Roberts, A.; Simon, I.; Raffel, C.; Engel, J.; Oore, S.; and Eck, D. 2018a. Onsets and Frames: Dual-objective piano transcription. In *Proc. Int. Soc. Music Information Retrieval Conf.*, 50–57.

Hawthorne, C.; Huang, A.; Ippolito, D.; and Eck, D. 2018b. Transformer-NADE for piano performances. In *Proc. Machine Learning for Creativity and Design Workshop*.

Holtzman, A.; Buys, J.; Du, L.; Forbes, M.; and Choi, Y. 2020. The curious case of neural text degeneration. In *Proc. Int. Conf. Learning Representations*.

Huang, C.-Z. A.; Vaswani, A.; Uszkoreit, J.; Simon, I.; Hawthorne, C.; Shazeer, N.; Dai, A. M.; Hoffman, M. D.; Dinculescu, M.; and Eck, D. 2019. Music Transformer: Generating music with long-term structure. In *Proc. Int. Conf. Learning Representations*.

Huang, Y.-S.; and Yang, Y.-H. 2020. Pop Music Transformer: Beat-based modeling and generation of expressive Pop piano compositions. In *Proc. ACM Multimedia*.

Jhamtani, H.; and Berg-Kirkpatrick, T. 2019. Modeling Self-Repetition in Music Generation using Generative Adversarial Networks. In *Proc. Machine Learning for Music Discovery Workshop*.

Jiang, J.; Wei, Y.; Feng, Y.; Cao, J.; and Gao, Y. 2019. Dynamic hypergraph neural networks. In *Proc. IJCAI*, 2635–2641.

Jiang, J.; Xia, G. G.; Carlton, D. B.; Anderson, C. N.; and Miyakawa, R. H. 2020. Transformer VAE: A hierarchical model for structure-aware and interpretable music representation learning. In *Proc. Int. Conf. Acoustics, Speech and Signal Processing*, 516–520.

Katharopoulos, A.; Vyas, A.; Pappas, N.; and Fleuret, F. 2020. Transformers are RNNs: Fast autoregressive Transformers with linear attention. In *Proc. Int. Conf. Machine Learning*.

Kazemi, S. M.; Goel, R.; Jain, K.; Kobyzev, I.; Sethi, A.; Forsyth, P.; and Poupart, P. 2020. Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research* 21(70): 1–73.

Ke, G.; He, D.; and Liu, T.-Y. 2020. Rethinking positional encoding in language pre-training. *arXiv preprint arXiv:2006.15595* .

Keskar, N. S.; McCann, B.; Varshney, L. R.; Xiong, C.; and Socher, R. 2019. CTRL: A conditional Transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858* .

Kivelä, M.; Arenas, A.; Barthelemy, M.; Gleeson, J. P.; Moreno, Y.; and Porter, M. A. 2014. Multilayer networks. *Journal of Complex Networks* 2(3): 203–271.

Lerch, A.; Arthur, C.; Pati, A.; and Gururani, S. 2019. Music performance analysis: A survey. In *Proc. Int. Soc. Music Information Retrieval Conf.*

Oore, S.; Simon, I.; Dieleman, S.; Eck, D.; and Simonyan, K. 2018. This time with feeling: Learning expressive musical performance. *Neural Computing and Applications* .

Patel, A. D. 2003. Language, music, syntax and the brain. *Nature Neuroscience* 6: 674–681.

Payne, C. M. 2019. MuseNet Accessed: 2021-03-01.

Rae, J. W.; Potapenko, A.; Jayakumar, S. M.; Hillier, C.; and Lillicrap, T. P. 2020. Compressive Transformers for long-range sequence modelling. In *Proc. Int. Conf. Learning Representations*.

Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; and Liu, P. J. 2020. Exploring the limits of transfer learning with a unified text-to-text Transformer. *Journal of Machine Learning Research* 21(140): 1–67.

Rossi, E.; Chamberlain, B.; Frasca, F.; Eynard, D.; Monti, F.; and Bronstein, M. 2020. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637* .

Uitdenbogerd, A.; and Zobel, J. 1999. Melodic matching techniques for large music databases. In *Proc. ACM Multimedia*, 57–66.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Proc. Advances in Neural Information Processing Systems*, 5998–6008.

Wu, S.-L.; and Yang, Y.-H. 2020. The Jazz Transformer on the front line: Exploring the shortcomings of AI-composed music through quantitative measures. In *Proc. Int. Soc. Music Information Retrieval Conf.*

Yang, L.-C.; and Lerch, A. 2020. On the evaluation of generative models in music. *Neural Computing and Applications* 32: 4773–4784.