electronics

Article

Generating Music Transition by Using a Transformer-Based Model

Jia-Lien Hsu * and Shuh-Jiun Chang

Department of Computer Science and Information Engineering, Fu Jen Catholic University, New Taipei City 242062, Taiwan; 0417csj@gmail.com
* Correspondence: alien@csie.fju.edu.tw

Abstract: With the prevalence of online video-sharing platforms increasing in recent years, many people have started to create their own videos and upload them onto the Internet. In filmmaking, background music is also one of the major elements besides the footage. With matching background music, a video can not only convey information, but also immerse the viewers in the setting of a story. There is often not only one piece of background music, but several, which is why audio editing and music production software are required. However, music editing is a professional expertise, and it can be hard for amateur creators to compose ideal pieces for the video. At the same time, there are some online audio libraries and music archives for sharing audio/music samples. For beginners, one possible way to compose background music for a video is "arranging and integrating samples", rather than making music from scratch. As a result, this leads to a problem. There might be some gaps between samples, in which we have to generate transitions to fill the gaps. In our research, we build a transformer-based model for generating a music transition to bridge two prepared music clips. We design and perform experiments to demonstrate that our results are promising. The results are also analysed by using a questionnaire to reveal a positive response from listeners, supporting that our generated transitions conform to background music.

Keywords: music transition; transformer; deep learning

check for updates

Citation: Hsu, J.-L.; Chang, S.-J. Generating Music Transition by Using a Transformer-Based Model. *Electronics* **2021**, *10*, 2276. https://doi.org/10.3390/electronics10182276

Academic Editor: George Angelos Papadopoulos

Received: 22 August 2021
Accepted: 13 September 2021
Published: 16 September 2021

1. Introduction

In the past, people used cameras and camcorders to take pictures of things in our lives. Nowadays, along with the advances in technology, mobile devices have become more prevalent, and many people use smartphones instead of cameras and camcorders to keep records of details in lives. These recorded images can not only be put on Facebook, Instagram, and other social media platforms, but can also be made into videos and uploaded to YouTube and other video sharing sites for people to share their daily lives with others.

In the face of the current trend of photography and video creation, much video editing software is available on the market for editing video materials, such as PowerDirector, iMovie, and Quik, which allows users to produce a video in a short time through simple operations. During production, not only do we need the filmed materials, we also need suitable background music. Many Internet platforms offer free soundtracks, but most of them are monotonous and repetitive. If we want the music to match the changing plots of a film, we need to select multiple pieces of music and edit them ourselves. However, music editing requires domain knowledge, which is a challenge for amateurs and the general public to produce materials that meet professional levels.

With the rise of Artificial Intelligence (AI) in recent years, people have easy access to professional knowledge and technology in music creation. Some companies and scholars have developed AI-based algorithms for automatic composition in which they are able to customize their music according to the user-specified musical styles, lengths, and beats. Examples can be found on platforms such as Jukedeck and AIVA. Some others leverage a small piece of user-given music to enable AI-based algorithms to generate subsequent parts, such as Music Transformer and MuseNet. Our project (sponsored by the Ministry of Science and Technology, Taiwan, under Contract No. MOST-108-2221-E-030-013-MY2) makes use of transformer-based algorithms to build a score generation system for short videos that allow the public to easily create their own video background music.

1.1. Research Background

In our MOST Project (MOST-108-2221-E-030-013-MY2), we design a soundtrack generation system for a short video. The proposed system consists of two main parts: video analysis and soundtrack generation.

In the video analysis, the video provided by the user undergoes several image analyses, and the features are extracted for subsequent soundtrack generation as shown in Figure 1.

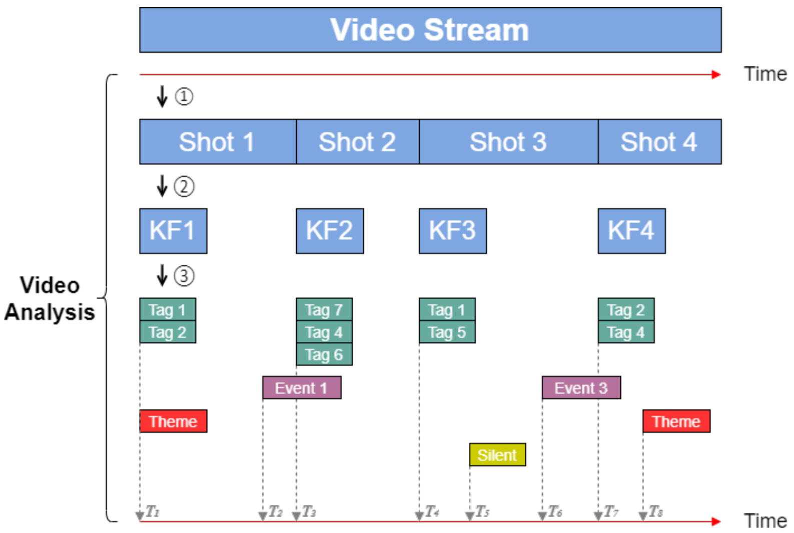


Figure 1. Video analysis framework.

At the beginning of the process (① in Figure 1), the system divides the user-given video into multiple shots by shot detection approaches, then finds the key frame (② in Figure 1) from the segmented shots, such as KF1, KF2, and KF3. From these key frames, the system finds the corresponding tags (③ in Figure 1), which consist of descriptive words, such as "style", "tone", "emotion", "feeling", etc. (e.g., Tag1 and Tag2 in Figure 1). Image analysis technology is also used to formulate the pattern for editing, which makes the event tags (e.g., Event 1 and Event 3 in Figure 1). Users can also add other tags during the process as they wish. Examples are tags for "playing the music of a specific theme at the time point of T_1" (e.g., 'Theme' in Figure 1) or tags for "maintaining silence at the time point of T_5" (e.g., 'Silent' in Figure 1) After video analysis, the system will retrieve all the tags and organize them as a sequence of tags associated with time stamps, where the element of each sequence is composed of time stamps and tag sets $(T_n, \{tag, \dots\})$. The following is an example of a sequence of tags:

$$Video \rightarrow \{(T_1, \{Tag1, Tag2, Theme\}), (T_2, \{Event1\}), \dots, (T_8, \{Theme\})\}$$

In soundtrack generation, the matching music clips are selected from the pool, and these clips are combined into a complete soundtrack to go with a short video for the user. The process is shown in Figure 2.

In this system, we build a music pool, and each music segment in the pool will be analysed beforehand to associate with a series of related tags. The corresponding tags and music clips will be maintained in the pool for later use in soundtrack generation.

At the beginning of soundtrack generation (④ in Figure 2), we perform the "match process" from the pool based on the tag sequences recorded by the video analysis and find candidates of music segments to be arranged on the timeline according to the time stamps in the sequences. Next, music fusion (⑤ in Figure 2) is conducted using these candidate segments. The fusion steps include "resolving conflicts between candidate segments" (e.g., MS 2 v.s. MS 4 and MS 5 v.s. MS 3 in Figure 2) and "creating music transition" (e.g., MS 1 vs. MS 2 and MS 3 vs. MS 7 in Figure 2). Lastly, we add the fused single-track score with chord and orchestration, coupled with arrangement and audio mixing (⑥ in Figure 2) to make a multi-track score that completes the whole short-video soundtrack.

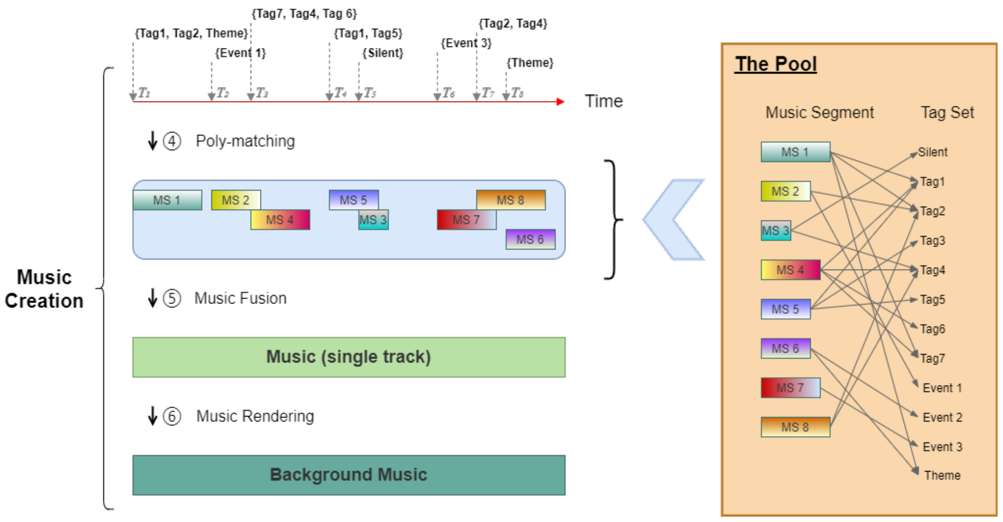


Figure 2. Music generation framework.

1.2. Research Objectives

In this paper, we focus on the "create music transition" process in the framework through the use of AI algorithms. Given two music segments, say MS 1 and MS 2, in the symbolic sequences, we design a transformer-based model to generate a *music transition sequence* to bridge MS 1 and MS 2.

2. Related Work

In this section, we introduce the related research topics, including MIDI communications protocol, automatic composition algorithms, and the transformer deep learning model.

2.1. MIDI

The Musical Instrument Digital Interface (MIDI) is a technical standard for electronic communications protocol that allows electronic instruments and computers to coordinate the playing of music through codes. MIDI files are composed of a header chunk followed by one or more track chunks. The header chunk records the basic data of the entire file, while the track chunk records the data of the music played through three types of events, including the MIDI event, sysex event, and meta event. In the MIDI event, there are seven functions to control the sound playback to correspond to the movement of the instrument as shown in Table 1.

Table 1. MIDI event functions.

Type	Function
Note on	triggering note playback
Note off	stopping note playback
Polyphonic key pressure	controlling pressure of each played note
Control change	changing controller settings
Program change	switching tones used in the channel
Channel aftertouch	controlling pressure of each played note in specified channels
Pitch wheel change	changing pitch wheel

2.2. Automatic Composition

In Fernández's paper [1], algorithms for automatic music composition are categorized into four main groups, Symbolic AI (grammars and rule-based systems), Machine Learning (Markov Chains and Artificial Neural Networks), Optimization Techniques (Evolutionary Algorithms), and Complex Systems (self-similarity and cellular automaton), as shown in Figure 3.

Symbolic AI applies music syntax to generate music. In earlier days, music syntax was obtained by humans from music theory or existing scores, and after the 1980s, some proposed computational approaches to extract music syntax. After obtaining the rules of syntax, it is possible to automatically generate music through algorithms such as the L-system [2–5] or Evolutionary Algorithms [6–9], and also to build a rule-based system [10] for generating music. Machine learning uses extensive existing music data to learn music with algorithms, such as Markov Chains [11–13] or Artificial Neural Networks [14–16], which has been popular in recent years. Evolutionary Algorithms are often used in Optimization Techniques where the fitness function selects the best candidate among many after several generations of cycles. The fitness function in Evolutionary Algorithms can conduct automatic determination using rule-based methods, artificial neural networks [17,18] or interactive filtering based on feedback from the test subjects [19,20]. In Complex Systems, it was found that the sound signal composed of Pink noise ($1/f$ noise) sounded better as in musical syntax. Researchers used the self-similarity to generate music [15], and some others created music through Cellular Automaton (CA) [21,22].

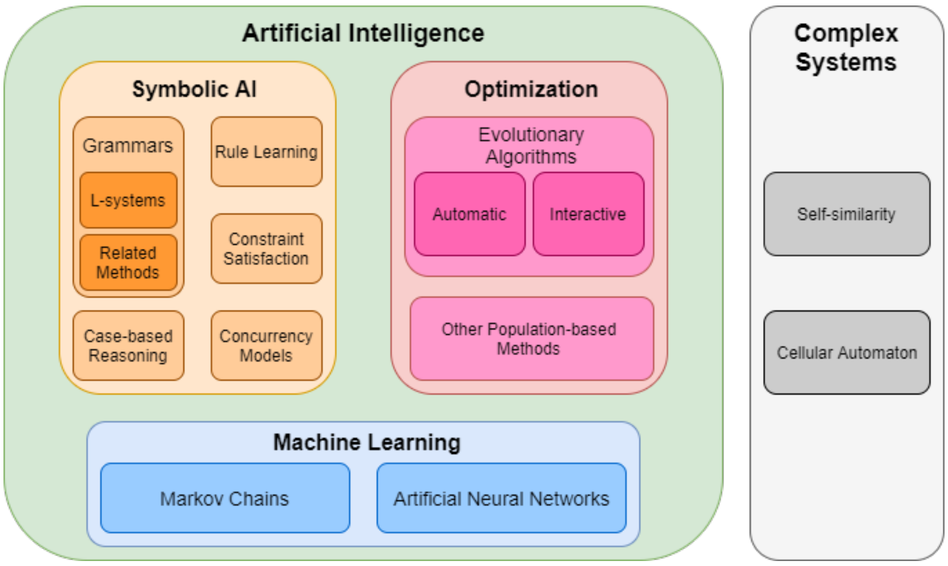


Figure 3. Categories of algorithms for automatic composition.

2.3. Transformer

Transformer is a deep learning algorithm proposed in Vaswani's paper [23] for text translation, and has been widely used in natural language processing (NLP) in recent years.

The Transformer is formed by an encoder–decoder architecture and uses the self-attention mechanism in the model. Figure 4 illustrates the Transformer model. When the input sequence is first entered into the Transformer, it goes through an embedding layer, which transforms each one-hot token in the input sequence into vectors of other dimensions. The converted input sequence is then processed with position encoding before entering the Encoder or Decoder. In this way, the position difference among tokens can be identified in the input sequence when self-attention is carried out. After this, the input sequence enters the Encoder or Decoder for computation.

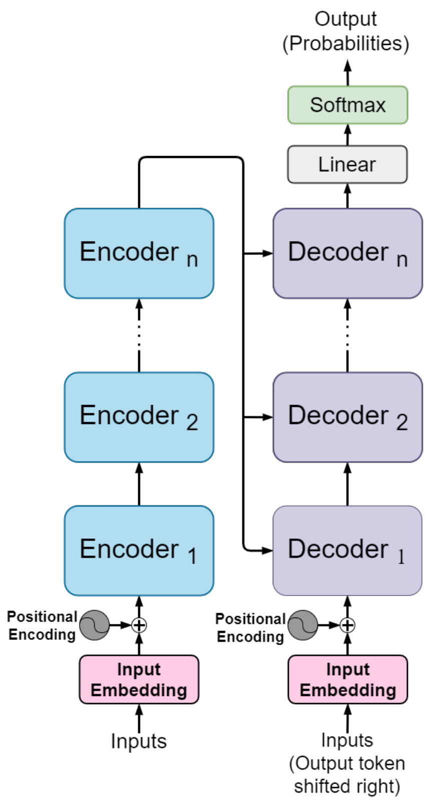


Figure 4. An illustration of transformer model.

The Encoder (as shown in Figure 5) consists of two sub-layers, the multi-head attention layer and the position-wise feed-forward layer, where the multi-head attention is the main part in which the Transformer carries out self-attention computing. Upon entering the multi-head attention layer, the input sequence is linearly transformed into multiple sets of queries (Q_h for Query, $Q_h = W_h^Q X$), keys (K_h for Key, $K_h = W_h^K X$), and values (V_h for Value, $V_h = W_h^V X$) based on the number of heads. Each group of Q_h, K_h, and V_h computes the attention between each token via scaled dot-product attention. The equation for the scaled dot-product attention computing is as follows:

$$A_h = \text{Softmax}(\frac{Q_h K_h^{\mathsf{T}}}{\sqrt{d}})V_h \tag{1}$$

The A_h is the result from the scaled dot-product attention computing, and d is the dimension size of Q_h, K_h, and V_h (to avoid large variance generated by Q_h and K_h dot-product computing, d is used for scaling the result of dot-product computing). After computing, multiple sets of results, $A_1, A_2, \ldots, A_h$, will be concatenated, and the final output from the multi-head attention layer will be calculated via a linear transformation. The Decoder (as shown in Figure 5), consists of three sub-layers, the masked multi-head attention layer, multi-head attention layer, and position-wise feed-forward layer. The masked multi-head attention layer, which does not exist in the Encoder, features masking that prevents the Decoder from viewing the tokens in advance during training. Therefore, the token after the predicted target is masked, as shown in Mask (Opt.) in Figure 6.

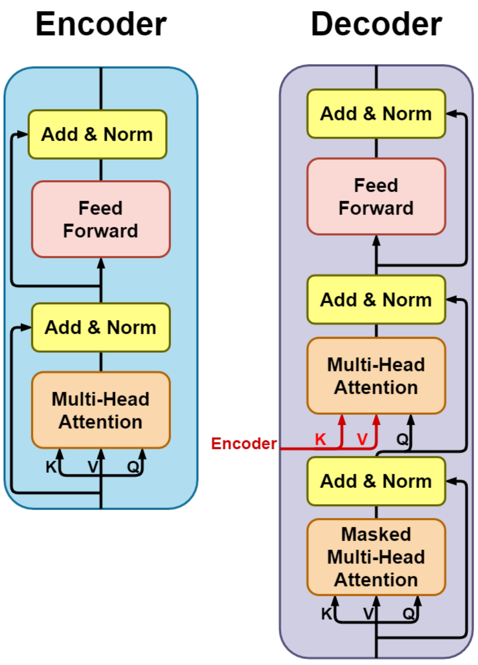


Figure 5. Encoder and Decoder in the Transformer.

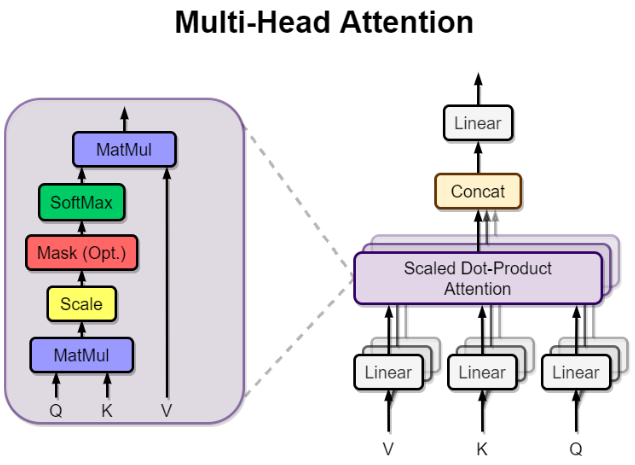


Figure 6. Multi-head Attention [24].

In the Encoder and Decoder, the position-wise feed forward layer is composed of two feed-forward layers, which are connected after the multi-head attention layer. The sub-layers of the Encoder and the Decoder are connected using layer normalization and residual connection. The experimental analysis of Vaswani [23] found that the training speed of the Transformer is faster than that of recurrent layer and convolution layer, and the hidden layer in the Transformer can retain more information with the help of self-attention computing, which makes the training result from the Transformer much better than that from the RNN.

In addition to the Transformer [23], this model has been applied in various fields. In music, for instance, Huang [25] proposed a Music Transformer, which uses a specified music sequence to generate subsequent music sequences. In text generation, Dai [26] proposed Transformer-XL, which can break the length limit of the Transformer [23] to generate longer texts. In terms of image generation, Child [27] proposed Sparse Transformers, which allow Transformers to process 2D data in addition to 1D data and adjust the attention mechanism to reduce the volume of model parameters for computation.

In reference to Figure 3, there are some possible approaches to developing specific methods of generating music transitions, such as a rule-based system, evolutionary algorithms, and machine-learning-based algorithms. When applying a rule-based system or evolutionary algorithms, we need domain expertise of music theory to construct rules or fitness functions. However, machine-learning-based algorithms would be data-driven, in which limited domain expertise is involved. Meanwhile, the transformer is a promising method for sequence generation. We believe that the machine-learning-based methods, including our approach, could be more friendly to beginners and be easily applied for rapid development in the early stage. Certainly, in the following stages, domain expertise should be always appreciated and involved to polish and refine our method to achieve more pleasant/delightful/attractive music transition sequences.

3. Method

In this section, we introduce the data processing and model construction used in the study. Figure 7 illustrates our framework of proposed methodology. The upper part of the framework is designed for the training and validation process to build the Transformer-based model. After finishing model construction, in the test phase, the input data are two music segments, namely preceding data and following data; the output data is music transition sequence (MTS) to bridge the two user-given segments.

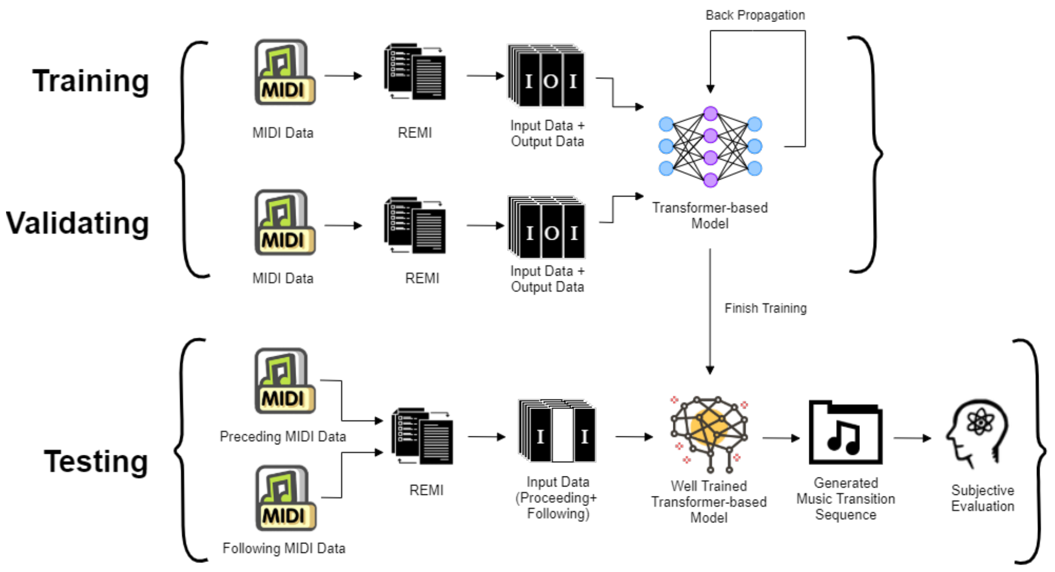


Figure 7. The proposed framework.

3.1. Data Pre-Processing

This section explains the music data representation method which converts music MIDI files into the specific data formats, and then properly organizes them as a training dataset for model construction.

3.1.1. REMI

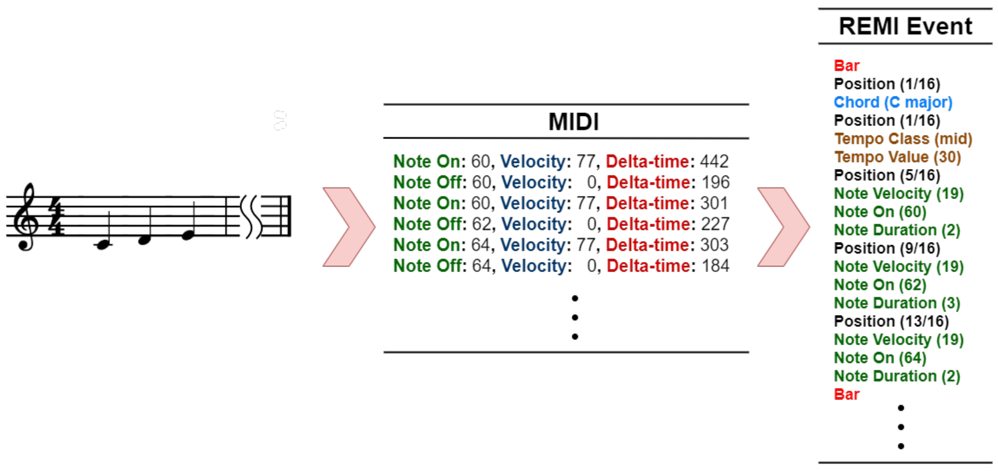
In this paper, the REMI representation proposed by Huang [28] is used to convert music MIDI files into the REMI-defined data format. Before the REMI representation was made public, most of the recent studies used the MIDI-like event representation proposed by Oore [29], which converts musical MIDI files into a data format compatible with the MIDI-like event representation to train models. The MIDI-like event representation converts the MIDI events in the MIDI file into 4 corresponding token events, which are, respectively, Note-On (trigger note play), Note-Off (end note play), Note Velocity (note play force), and Time-Shift (time difference between token events).

Huang's study [28] found several features of MIDI-like event representations; for example: MIDI-like event representations convert music MIDI files in a way that features faithful representation of keyboard-style music (e.g., piano music). In this way, the converted data lack the high-level musical information such as Downbeat, Chord, and Tempo presented in the score. Huang [28] also mentioned in their paper that "We note that when humans compose music, we tend to organize regularly recurring patterns and accents over a metrical structure defined in terms of sub-beats, beats, and bars." [28] (p. 2), therefore they proposed REMI to resolve the issues of MIDI-like event representations.

In reference to Table 2, REMI representation retains the original MIDI-like event representation's Note-On and Note Velocity, and adds token events that include Note Duration, Position & Bar, Tempo, and Chord. The newly-added Note Duration replaces Note-off in the MIDI-like event representation. In the MIDI-like event representation, the playing of a note is based on its Note-on, Note-off, and the Time-Shift accumulated in between, which often contains many other token events between Note-On and Note-Off (in Huang's practice [28], there is an average of 21.7 ± 15.3 token events in between). REMI representation only needs to use Note-On and the adjacent Note Duration to determine a note's beginning and ending for a model in training to easily learn this feature. REMI representation replaces Time Shift in the MIDI-like event representation with the newly added Position & Bar. Time-Shift in MIDI-like event representation marks the time difference between token events. Huang's study found that training models could not use Time-Shift to generate music with a steady beat, and Huang attributed this to the lack of metrical structure in MIDI-like event representation. Therefore, in REMI representation, Position & Bar is used to represent token events' absolute positions on the score. In subsequent experiments, they also found that the models could easily learn the dependency of note events on the same Position through Position & Bar. Lastly, REMI's newly added Tempo and Chord supplement higher-level musical data which MIDI-like event representation lacks. In reference to Figure 8, we illustrate an example of REMI representation.

Table 2. Comparison between MIDI-like and REMI representations.

Token Event	**MIDI-like [29]**	**REMI [28]**
Note onset	Note-On (0–127)	Note-On (0–127)
Note offset	Note-Off (0–127)	Note Duration (32th note multiples; 1–64)
Time grid	Time-Shift (10–1000 ms)	Position (16 bins; 1–16) & Bar
Tempo changes	N/A	Tempo (30–209 BPM, beats per measure)
Chord	N/A	Chord (60 types)



Note On: 60, Velocity: 77, Delta-time: 442
Note Off: 60, Velocity: 0, Delta-time: 196
Note On: 60, Velocity: 77, Delta-time: 301
Note Off: 62, Velocity: 0, Delta-time: 227
Note On: 64, Velocity: 77, Delta-time: 303
Note Off: 64, Velocity: 0, Delta-time: 184
⋮

Bar
Position (1/16)
Chord (C major)
Position (1/16)
Tempo Class (mid)
Tempo Value (30)
Position (5/16)
Note Velocity (19)
Note On (60)
Note Duration (2)
Position (9/16)
Note Velocity (19)
Note On (62)
Note Duration (3)
Position (13/16)
Note Velocity (19)
Note On (64)
Note Duration (2)
Bar
⋮

Figure 8. An example of REMI representation.

3.1.2. Data Processing

The goal of this study is to generate a musical transition sequence (MTS), in which the generated MTS must refer to its preceding music sequence and the following one. The preceding music sequence and the following music sequence are the input for the training models, with their output as MTSs. In data processing, the music MIDI files are converted into REMI format and segmented into units every n number of tokens. In this paper, the value of n is set to 256 (based on NVIDIA GeForce RTX 2080, the equipment used in this experiment). In the experiment, every three consecutive units form one sample of training input and the corresponding output data. In reference to Figure 9, the sliding window is designed to cover three units, and the window moves one unit forward at a time to generate the next sample for the training.

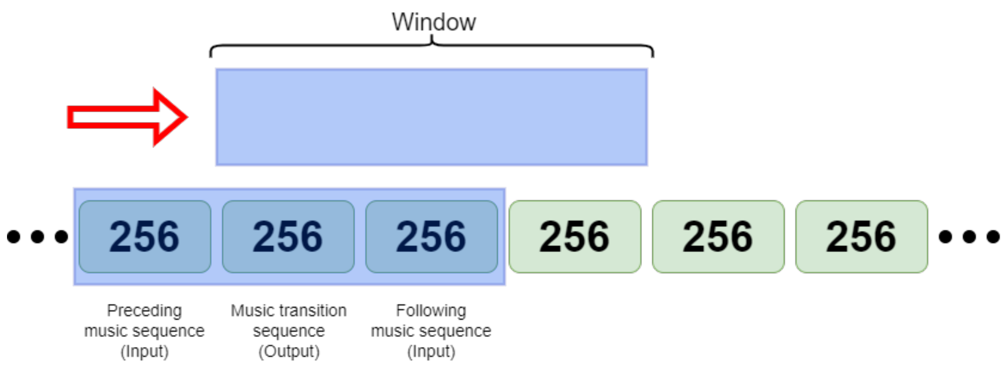

Window
256
256
256
256
256
256
Preceding music sequence (Input)
Music transition sequence (Output)
Following music sequence (Input)

Figure 9. Data processing of training samples for model construction.

3.2. *Deep Learning Framework*

In this paper, we use a transformer-based model to generate music transition sequences (MTS). A preceding music sequence and a following music sequence are used as the input to generate an MTS that bridges the two musical sequences. In recent years, Vaswani [23] proposed the transformer deep learning model, which not only received good results in NLP, but also is widely applied in other fields. In the music field, Huang [25] also successfully used music transformers to generate subsequent music sequences with a small segment of sequence. In this study, we also applied a transformer-based model to solve the challenges when it comes to MTS.

The structure of the model referred to the Encoder–Decoder structure used by Vaswani [23] in the transformer learning model. Since the goal of this paper is to generate an MTS between two music segments of sequences, our framework, consisting of two encoders and one decoder, is applied to build the transformer-based model, as shown in Figure 10.

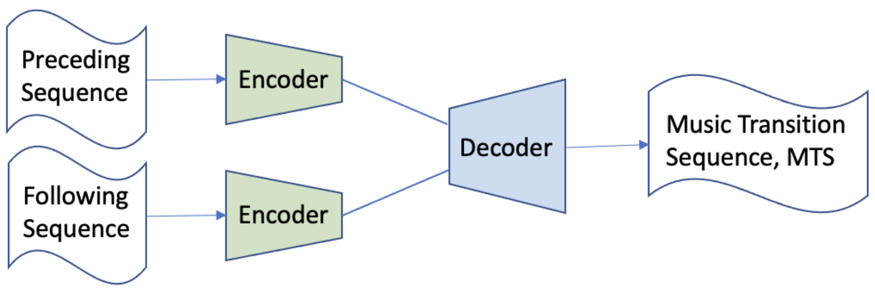


Figure 10. Encoder–Decoder framework.

By using two encoders to capture key information from the preceding music sequence and the following one, the decoder might generate the suitable/appropriate MTS with reference to the features from the two music sequences.

In our model, we also make use of the positional encoding used in Transformer-XL [26], which is different from the one used in the Transformer [23]. In the Transformer, positional encoding is applied to enhance the positional information between tokens, so that the input sequence can recognize the positional difference between tokens when performing self-attention computation. Sinusoid values of different frequencies are applied to the input sequence, which can be seen as an implant of absolute positional information into the input sequence. After Shaw's [30] paper was published, subsequent studies of transformers [25,26] started to employ relative positional encoding in the models. In the study [25,26], the training error of using relative positional encoding is smaller than that of using absolute positional encoding. As a result, in this paper, we also employ relative positional encoding in the transformer-based model used in the experiment.

In the transformer-based model in this paper, the encoder is the same as that used in Transformer [23], which consists of a multi-head attention layer and position-wise feed-forward layer, and these layers are connected through layer normalization and residual connection. The decoder, on the other hand, adopts the one from the Transformer for adjustment. As shown in the Decoder part in Figure 11, the sub-layer at the bottom adopts the masked multi-head attention layer from the Transformer [23], and after it, two consecutive multi-head attention layers are used. One of them is responsible for receiving the encoder information from the preceding music sequence of the MTS, and the other for the one from the following music sequence. Lastly, the results from the two multi-head attention layers are concatenated to go through a feed-forward layer. In the Decoder part, (masked) multi-head attention layers are connected to each other using layer normalization and residual connection. Only the concatenated feed-forward layer in the end of the process is connected to the masked multi-head attention layer in the beginning using the residual connection. Figure 12 shows our framework of transformer-based model.

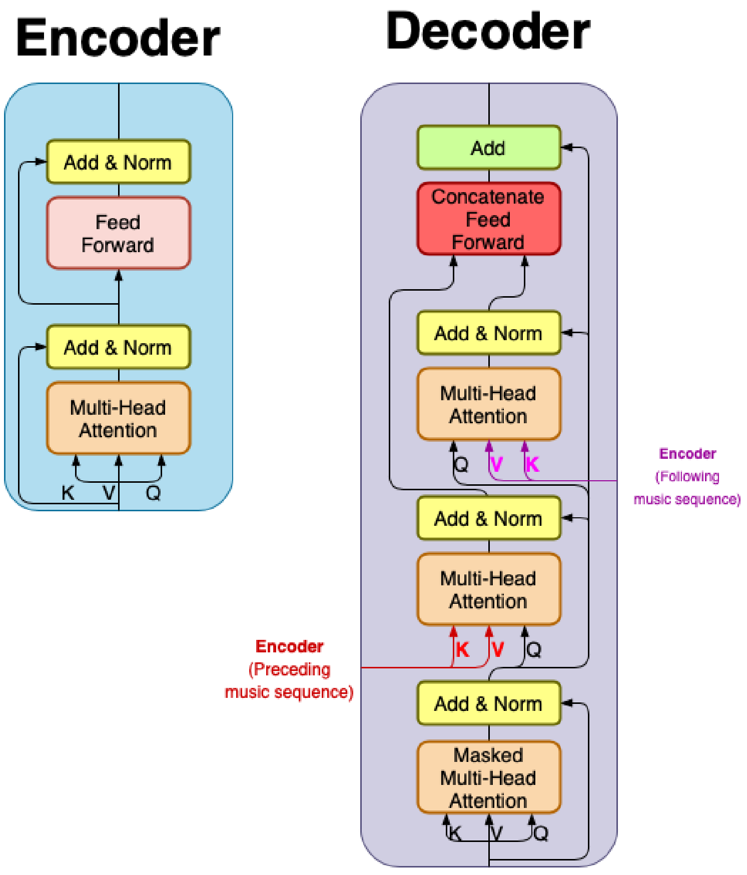


Figure 11. Encoder and decoder.

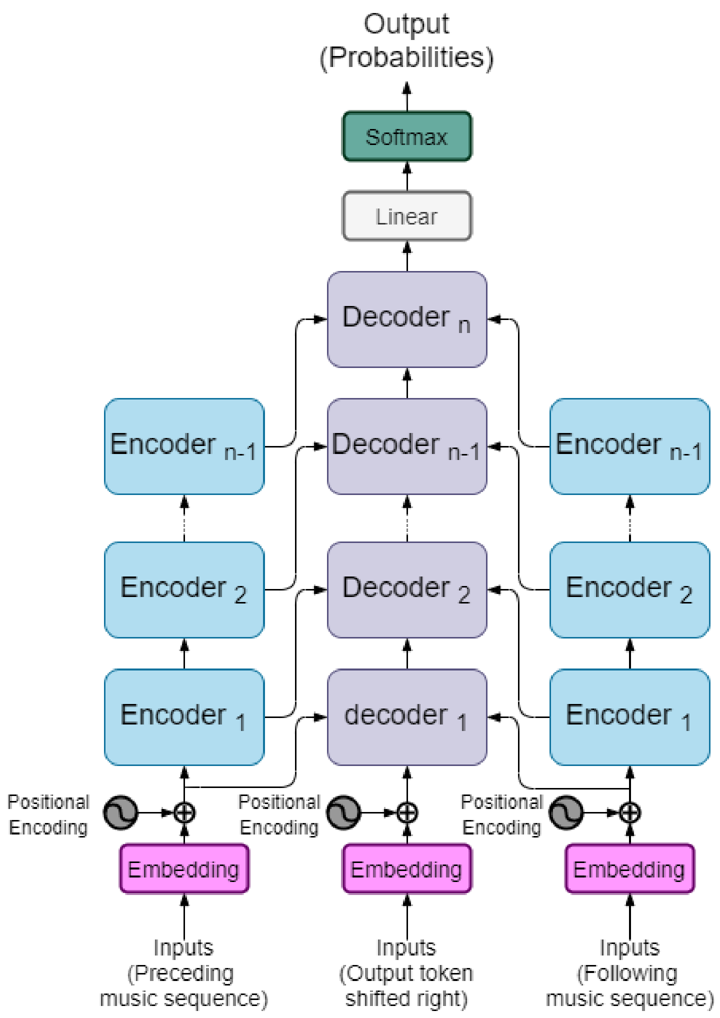
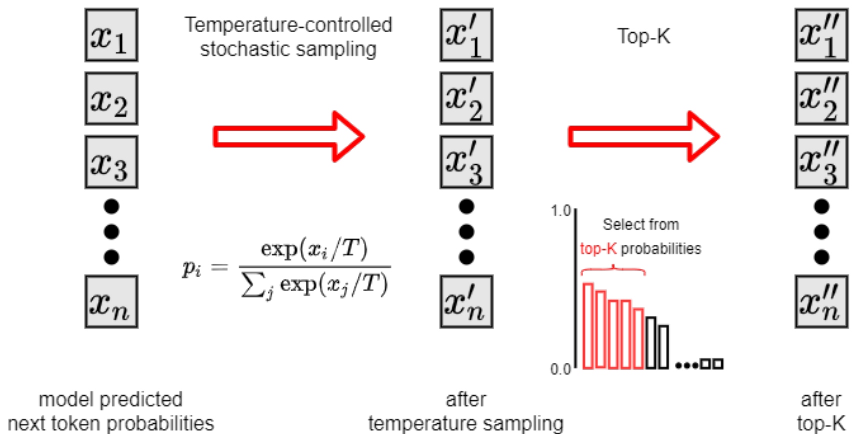


Figure 12. Transformer-based model.

3.3. *Sampling*

In the framework, the trained transformer-based model can be used to only predict the probability of the next token. To generate a sequence consisting of tokens, we apply the temperature-controlled stochastic sampling method with top-*k* [31] to determine consecutive tokens. As shown in Figure 13, the token probabilities predicted by the model undergo temperature sampling, before top-*k* is used for the selection of the next token.



Temperature-controlled stochastic sampling
Top-K
x_1
x_2
x_3
x_n
x'_1
x'_2
x'_3
x'_n
x''_1
x''_2
x''_3
x''_n
$p_i = \frac{\exp(x_i/T)}{\sum_j \exp(x_j/T)}$
1.0
0.0
Select from top-K probabilities
model predicted next token probabilities
after temperature sampling
after top-K

Figure 13. Temperature-controlled stochastic sampling method with top-*k*.

3.4. Loss Function

In our framework, KL divergence defined in Equation (2) is used as the loss function. When processing data, the prediction target is converted into one-hot encoding. Therefore, when training the model, the output is the probability of each class. Thus, this paper uses KL divergence to measure the degree of difference between two independent probability distributions with the following equation:

$$D_{KL}(T||P) = -E_{x\sim T}[\ln P(x) - \ln T(x)] = -\sum_{x} T(x) \ln \frac{P(x)}{T(x)} \tag{2}$$

$T(x)$ is the probability distribution of the predicted target, and $P(x)$ is that of the model output. By calculating the KL divergence, we can obtain the decreased expected value that has changed from the probability distribution of the predicted target to that of the model output, which can be used to adjust the model parameters and improve the accuracy of the training model.

3.5. Optimizer

This section describes the optimizer used in the model training process.

3.5.1. Adam Optimizer

The training process uses the Adaptive Moment Estimation (Adam) as the optimizer in the experiment. Adam is one of the commonly used optimizers in deep-learning models, combining the advantages of momentum and RMSprop, as shown in the following equation:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{3a}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{3b}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{3c}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{3d}$$

$$x^{t+1} = x^t - \frac{\gamma}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t \tag{3e}$$

In the equation, m_t serves similar purposes as momentum, which is used to adjust the amount of corrections made to the model. The v_t , on the other hand, works as RMSprop, which is used to adjust the learning rate in a dynamic approach in the optimizer according to the gradient of loss in the model. β_1 and β_2 in the equation are the degrees of decline of m_t and v_t, and g_t is the gradient of loss in the training model. Equation (3c) and (3d) are used to keep m_t and v_t from leaning toward 0 in the early stages of model training, which would lead to excessive model correction and scattered training results. ε in Equation (3e) is a parameter that keeps the denominator from being 0, while γ is the learning rate set by Adam.

In this paper, the Adam parameters are set as $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\gamma = 0$, $\varepsilon = 10^{-9}$.

3.5.2. Warmup

Vaswani [23] found in the course of the experiment that the model gradient changes greatly in the transformer's early stages of training, which may cause Adam to update too many of the model parameters in the early stages of training, possibly resulting in scattered training results. For this reason, Vaswani used warmups to assist Adam when training the transformer. When using warmups, there are warmup steps; when the number of training sessions is smaller than that of the warmup steps, the learning rate of training will be increased gradually with reference to the equation, to a point when the number of training sessions is larger than that of the warmup steps, before the learning rate is gradually dialed down, as shown in Figure 14. The equation for warmup steps is as follows:

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}) \quad (4)$$

$lrate$ is the learning rate used by the optimizer, and d_{model} is the dimension size set by the transformer in the embedding layer. $step\_num$ stands for the number of epochs during the training, and $warmup\_steps$ is the value set for warmup steps. Vaswani [23] used warmups to avoid scattered training results in the early stages using a transformer, as well as to minimize the gradient of loss.

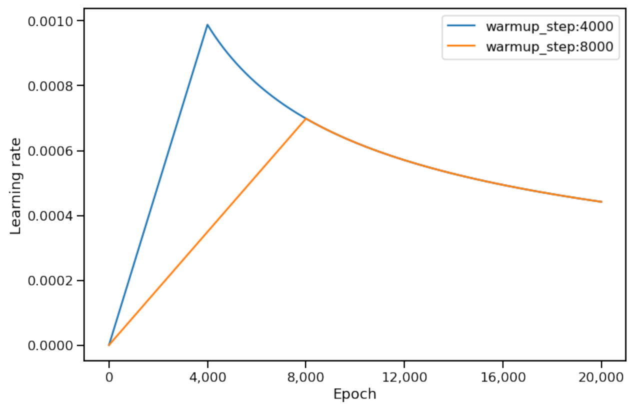


Figure 14. Warmup (learning rate initially set to 0, with warmup step at 4000 for the blue line and 8000 for the orange).

3.6. *Regularization*

This section explains the regularization used to improve the result accuracy from the training model.

3.6.1. Label Smoothing

Label smoothing is used during model training to reach a prediction target with soft one-hot encoding, so that the output of the model can be adjusted to the prediction target. Label smoothing prevents the model from generating over-confident results, and the equation of adopting label smoothing is as follows:

$$y_c^{LS} = y_c(1-\alpha) + \frac{\alpha}{C} \quad (5)$$

y_c is the value of the predicted target for the c-th class, and α is the parameter for the smoothing. When α equals 0, we have the value of the original prediction target, while when α equals to 1, the distribution of the prediction target will be uniform.

Label smoothing first appeared in Szegedy's paper [32], and although label smoothing raises the uncertainty for the model and reduces the accuracy during training, it improves the accuracy during validation. Vaswani's paper [23] used label smoothing to raise their scores in BLEU. Note that BLEU (bilingual evaluation understudy score) is a metric for evaluating the quality of text which has been machine-translated from one natural language to another. BLEU indicates how similar the candidate text is to the reference text, with values closer to one representing more similar texts [33]. This paper refers to the label smoothing used in Vaswani's paper [23] and sets the value of parameter α as 0.1 to improve the accuracy during validation. This also allows the model to obtain better results when using the temperature-controlled stochastic sampling method with top-k [31].

3.6.2. Dropout

One of the challenges in machine learning training process is over-fitting. In past research, some used model combination as a solution, but this method requires several different models for training, and the large amount of training data as well as computational costs make it even more difficult to be adopted. In 2014, Hinton [34] proposed Dropout, a training method similar to model combination. Dropout rules out some units in the neural network and trains the model with a new thinned neural network instead. The loss resulting from the thinned neural network during the training will be updated by back-propagation to generate a modified thinned neural network. After the update, the model retrieves the units that have been removed earlier to restore the original neural network. The above process will be repeated until the training is finished. The following is the equation of the neural network before Dropout is adopted:

$$z_i^{(l+1)} = w_i^{(l+1)} y^l + b_i^{(l+1)} \tag{6a}$$

$$y_i^{(l+1)} = f(z_i^{(l+1)}) \tag{6b}$$

After Dropout is adopted:

$$r_j^l \sim \text{Bernoulli}(p) \tag{7a}$$

$$\hat{y}^{(l)} = r^l * y^{(l)} \tag{7b}$$

$$z_i^{(l+1)} = w_i^{(l+1)} \hat{y}^l + b_i^{(l+1)} \tag{7c}$$

$$y_i^{(l+1)} = f(z_i^{(l+1)}) \tag{7d}$$

l is the position of neural network hidden layers; ${z_i}^{(l)}$ is the output of the i-th unit of Layer l; f is any activation function; $y_i^{(l)}$ is the output of the i-th unit of Layer l after going through the activation function f. $b_i^{(l)}$ and $w_i^{(l)}$ are the bias and weight of the i-th unit of Layer l, and $r_j^{(l)}$ is the vector from Bernoulli random variables which generates 1 with probability p.

Using Dropout in a neural network is like sampling multiple thinned neural networks from the original network. When there are n units in the neural network, we can sample 2^n kinds of thinned neural networks. That is, when training a neural network using Dropout, it can be seen as training a thinned neural network with 2^n kinds of sets, just like the model combination explained above. In this paper, Dropout is used in the model, and the probability of Dropout is set to 0.1.

4. Experiment

In this section, we introduce the data source and evaluation in our experiments.

4.1. Data Source

We have two datasets, namely pop music and classical music. In our experiment, we train two different model using the two datasets separately. The dataset used for pop music is the training dataset provided by Huang [28], which contains 775 pieces, including MIDI files of Western pop, Korean pop music, and music from Japanese anime. The dataset used for classical music is the recorded MIDI files of 290 pieces played by the contestants in the Piano-e-Competition in 2018. Before the deep learning model was trained, the datasets of each music style were divided into 80/10/10, which represent 80% training data, 10% validation data, and 10% test data.

4.2. Evaluation

In the experiments, we trained two sets of model parameters (stack for six layers; stack for three layers) separately on the pop and classical music datasets. At the end of the training, we compared the results by verifying the KL Divergence loss from the datasets, and the results are shown in the tables below. In the results for the two styles, the divergence loss in the model with a stack for six layers is the minimum. Table 3 shows the experimental results with the pop dataset, and Table 4 the classical dataset.

Table 3. Results with popular music dataset.

POP Music (Validating Data)	KL Divergence Loss
Transformer-based model (stack for six layers)	0.00323
Transformer-based model (stack for three layers)	0.00416

Table 4. Results with classical music dataset.

Classical Music (Validating Data)	KL Divergence Loss
Transformer-based model (stack for six layers)	0.00371
Transformer-based model (stack for three layers)	0.00437

4.3. Listening Test

In addition to the data analysis shown in Sec. 4.2, we also designed a listening test to evaluate the results generated by the model. We came up with a total of 20 questions in the listening test in the form of an online questionnaire, with ten questions each for the pop and classical genres. Each test included a piece of music, paired with a question. Before the test, the subjects were asked whether or not he/she is a music professional who understands basic music theory and has played a musical instrument for more than six years. At the beginning of the test, the subjects must first listen to the music before they answered the 20 questions and reached the end of the listening test.

The folder (at `https://reurl.cc/73XArb`) contains all the twenty music clips in the listening test. In the folder, there are two subdirectories: "`Pop Testing Music`" and "`Piano_e Testing Music`". For each subdirectory, there are ten midi files, for instance, "`Question 1(original).midi`" and "`Question 2(model).midi`". Regarding the file name, we further explain with the following examples: "`Question 1(original).midi`" indicates that the music is for question 1, and all the midi is cut from the original music file. "`Question 2(model).midi`" indicates that the music is for question 2, and part of the midi (i.e., the music transition) is generated through our approach. Therefore, in our experiment, the subject (listener) has no idea which midi file is from original music and which is generated using our approach.

The music pieces in the test were sequences of 768 REMI token events, consisting of ten sequential segments of 768 REMI token events randomly selected from both the pop and classical musical style datasets. Five segments with 256 REMI token events in the middle were replaced by 256 model-generated REMI token events, which were sampled using the temperature-controlled stochastic sampling method with the top-*k* algorithm [31]. The other five segments of REMI token events remained the way they had been generated. In the test, the subjects were asked to comment on the fluency of the music clips, based on a scale from 1 to 5, with 5 being the most fluent and 1 the least, as shown in Figure 15.

In this test, we collected 26 subjects, four of whom are music professionals who know basic music theory and have played musical instruments for more than six years, and the rest are non-music professionals. In pop music, the model-generated clips scored 3.45 on average and clips from original pieces scored 4.19; in classical music, the model-generated clips scored 3.02 on average and clips from original pieces scored 3.55.

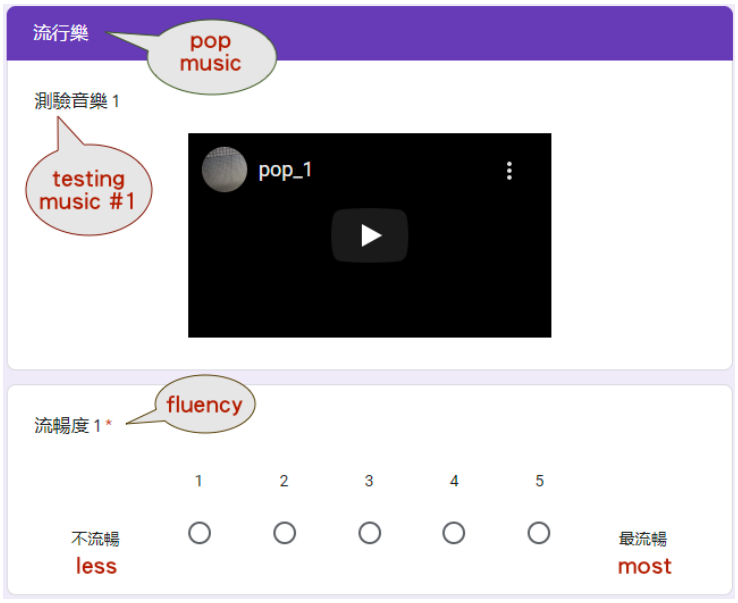


Figure 15. The screenshot of the listening test. (The text in the interface is in Chinese. We provide the English translation indicated with the box)

In the listening experiment, we use a boxplot to illustrate the distribution of the score of each test. In reference to Figure 16, the test scores of original music clips are more concentrated than those of the model-generated ones. In addition, only one or two model-generated music clips are considered to be more fluent than most of original music clips. Compared with original music clips, more than half of the model-generated music sounds choppy and rough.

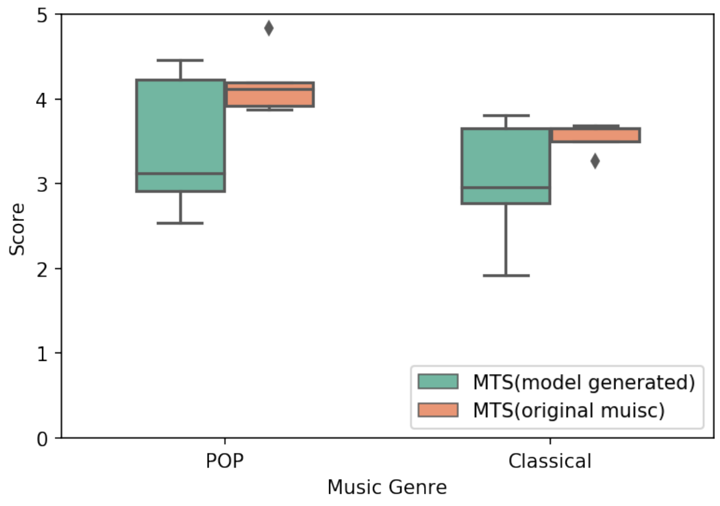


Figure 16. The boxplot of test scores.

5. Conclusions

This paper explores music transition, and the goal is to generate a music transition sequence (MTS) that fills in the gaps between a preceding music sequence and a following one so that they can be connected. In the experiment, both pop and classical music datasets were used to train the Transformer-based model, where an assumption had been made that the two music sequences (preceding ones and following ones) and the generated MTS all consist of 256 REMI token events. Under this condition, the Transformer-based model was trained using a total of 1000 epochs, and a comparison of the two sets of model parameters (stack for six layers; stack for three layers) shows that the model with six layers has a slimmer validation dataset loss. In addition, this paper performed twenty questions in the listening test. The music sequences in ten of the questions were existing music clips (768 REMI token events), and the remaining ten questions consisted of model-generated MTSs that had been used to replace those of existing music clips (replacing the middle 256 REMI token events of the original 768 ones with 256 events generated by the model). The test results showed that although the existing music sequences scored higher on average than the model-generated ones did, the average score of the model-generated ones was above 3, which was not the worst. With improvement, it is believed that this experiment can obtain comparable results in future studies.

Author Contributions: J.-L.H. conceived and designed the proposed method, analyzed the data, performed the computation work, prepared figures and/or tables, authored and reviewed drafts of the paper, and approved the final draft. S.-J.C. implemented the proposed method, performed the experiments, performed the computation work, and analyzed the data. All authors have read and agreed to the published version of the manuscript.

Funding: This study is financially supported by the Ministry of Science and Technology, Taiwan, under Contract No. 108-2221-E-030-013-MY2, and by Fu Jen Catholic University, under Contract No. A0110010. The funders did not take part in study design, data collection and analysis, decision to publish, or manuscript preparation.

Data Availability Statement: The data presented in the listening test of this study are openly available, and can be found here: https://reurl.cc/73XArb.

Conflicts of Interest: The authors declare that there are no competing interests.

References

1. Fernández, J.D.; Vico, F. AI methods in algorithmic composition: A comprehensive survey. *J. Artif. Intell. Res.* **2013**, *48*, 513–582.
2. DuBois, R.L. Applications of Generative String-Substitution Systems in Computer Music. Ph.D. Dissertation, Columbia University, New York, NY, USA, 2003.
3. Wilson, A.J. A Symbolic Sonification of L-Systems. In Proceedings of the International Computer Music Conference (ICMC 209), Montréal, QC, Canada, 16–21 August 2009.
4. Bulley, J.; Jones, D. Variable 4: A dynamical composition for weather systems. In Proceedings of the International Computer Music Conference (ICMC 2011), Huddersfield, UK, 31 July–5 August 2011.
5. Pestana, P. Lindenmayer systems and the harmony of fractals. *Chaotic Model. Simul* **2012**, *1*, 91–99.
6. Bryden, K. Using a Human-in-the-Loop Evolutionary Algorithm to Create Data-Driven Music. In Proceedings of the 2006 IEEE International Conference on Evolutionary Computation, Vancouver, BC, Canada, 16–21 July 2006; pp. 2065–2071; doi:10.1109/CEC.2006.1688561.
7. Fox, C. Genetic Hierarchical Music Structures. In Proceedings of the International FLAIRS Conference, Melbourne Beach, FL, USA, 11–13 May 2006; pp. 243–247.
8. Peck, J.M. Explorations in Algorithmic Composition: Systems of Composition and Examination of Several Original Works. Master's Thesis, State University of New York, Oswego, NY, USA, 2011.
9. Kuo, P.; Li, T.S.; Ho, Y.; Lin, C. Development of an Automatic Emotional Music Accompaniment System by Fuzzy Logic and Adaptive Partition Evolutionary Genetic Algorithm. *IEEE Access* **2015**, *3*, 815–824, doi:10.1109/ACCESS.2015.2443985.
10. Thomas, M.T. *Vivace: A Rule Based Al System for Composition*; Michigan Publishing, University of Michigan Library: Ann Arbor, MI, USA, 1985.
11. Pachet, F.; Roy, P.; Barbieri, G. Finite-length Markov processes with constraints. In Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, Barcelona, Spain, 19 July 2011.
12. Grachten, M. JIG: Jazz improvisation generator. In Proceedings of the MOSART Workshop on Current Research Directions in Computer Music, Barcelona, Spain, 15–17 November 2001; pp. 1–6.
13. Thornton, C. Hierarchical markov modeling for generative music. In Proceedings of the International Computer Music Conference (ICMC 2009), Montreal, QC, Canada, 16–21 August 2009.
14. Browne, T.M.; Fox, C. Global Expectation-Violation as Fitness Function in Evolutionary Composition. In *Proceedings of the Workshops on Applications of Evolutionary Computing*; Giacobini, M., Brabazon, A., Cagnoni, S., Di Caro, G.A., Ekárt, A., Esparcia-Alcázar, A.I., Farooq, M., Fink, A., Machado, P., Eds.; Springer: Berlin/Heidelberg, Germany, 2009; pp. 538–546.
15. Coca, A.E.; Romero, R.A.F.; Zhao, L. Generation of composed musical structures through recurrent neural networks based on chaotic inspiration. In Proceeding of the 2011 International Joint Conference on Neural Networks, San Jose, CA, USA, 31 July–5 August 2011; pp. 3220–3226, doi:10.1109/IJCNN.2011.6033648.
16. Adiloglu, K.; Alpaslan, F.N. A machine learning approach to two-voice counterpoint composition. *Knowl.-Based Syst.* **2007**, *20*, 300–309.

17. Klinger, R.; Rudolph, G. Evolutionary composition of music with learned melody evaluation. In Proceedings of the International Conference on Computational Intelligence, Man-machine Systems and Cybernetics (CIMMACS'06), Venice Italy, 20–22 November 2006.
18. Manaris, B.; Roos, P.; Machado, P.; Krehbiel, D.; Pellicoro, L.; Romero, J. A corpus-based hybrid approach to music analysis and composition. In *Proceedings of the National Conference on Artificial Intelligence*; AAAI Press, 2007; Volume 1, p. 839 – 845.
19. Diaz-Jerez, G. Composing with Melomics: Delving into the Computational World for Musical Inspiration. *Leonardo Music J.* **2011**, *21*, 13–14, doi:10.1162/LMJ_a_00053.
20. Marques, V.M.; Reis, C.; Machado, J.A.T. Interactive Evolutionary Computation in music. In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Istanbul, Turkey, 10–13 October 2010; pp. 3501–3507, doi:10.1109/ICSMC.2010.5642417.
21. Miranda, E.R. Cellular automata music: An interdisciplinary project. *J. New Music Res.* **1993**, *22*, 3–21.
22. Dorin, A. *LiquiPrism: Generating Polyrhythms with Cellular Automata*; Georgia Institute of Technology: Atlanta, Georgia, 2002.
23. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*; Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2017; Volume 30, pp. 5998–6008.
24. Tay, Y.; Dehghani, M.; Bahri, D.; Metzler, D. Efficient Transformers: A Survey. *arXiv* **2020**, arXiv:2009.06732.
25. Huang, C.Z.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 Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
26. Dai, Z.; Yang, Z.; Yang, Y.; Carbonell, J.; Le, Q.; Salakhutdinov, R. Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Florence, Italy, 28 July–2 August 2019; pp. 2978–2988, doi:10.18653/v1/P19-1285.
27. Child, R.; Gray, S.; Radford, A.; Sutskever, I. Generating Long Sequences with Sparse Transformers. *arXiv* **2019**, arXiv:1904.10509.
28. Huang, Y.S.; Yang, Y.H. Pop Music Transformer: Beat-Based Modeling and Generation of Expressive Pop Piano Compositions. In *MM'20, Proceedings of the 28th ACM International Conference on Multimedia, Seattle, WA, USA, 12–16 October 2020*; Association for Computing Machinery: New York, NY, USA, 2020; pp. 1180–1188. doi:10.1145/3394171.3413671.
29. Oore, S.; Simon, I.; Dieleman, S.; Eck, D.; Simonyan, K. This time with feeling: Learning expressive musical performance. *Neural Comput. Appl.* **2020**, *32*, 955–967.
30. Shaw, P.; Uszkoreit, J.; Vaswani, A. Self-Attention with Relative Position Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2018; pp. 464–468.
31. Keskar, N.; McCann, B.; Varshney, L.R.; Xiong, C.; Socher, R. CTRL: A Conditional Transformer Language Model for Controllable Generation. *arXiv* **2019**, *arXiv:1909.05858*.
32. Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; Wojna, Z. Rethinking the Inception Architecture for Computer Vision. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 2818–2826.
33. Papineni, K.; Roukos, S.; Ward, T.; Zhu, W.J. BLEU: A Method for Automatic Evaluation of Machine Translation. In *ACL '02, Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, Philadelphia, PA, USA, 7–12 July 2002*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2002; pp. 311–318, doi:10.3115/1073083.1073135.
34. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.