Museformer: Transformer with Fine- and Coarse-Grained Attention for Music Generation

Botao Yu[†], **Peiling Lu**[‡], **Rui Wang**[‡], **Wei Hu**[†*], **Xu Tan**[‡*],
Wei Ye[§], **Shikun Zhang**[§], **Tao Qin**[‡], **Tie-Yan Liu**[‡]
[†]State Key Laboratory for Novel Software Technology, Nanjing University, China
[‡]Microsoft Research Asia
[§]National Engineering Research Center for Software Engineering, Peking University, China
btyu@foxmail.com, {peil,ruiwa,xuta,taoqin,tyliu}@microsoft.com,
whu@nju.edu.cn, {wye,zhangsk}@pku.edu.cn

Abstract

Symbolic music generation aims to generate music scores automatically. A recent trend is to use Transformer or its variants in music generation, which is, however, suboptimal, because the full attention cannot efficiently model the typically long music sequences (e.g., over $10{,}000$ tokens), and the existing models fall short in generating the repetitive music structures. In this paper, we propose Museformer, a Transformer with a novel fine- and coarse-grained attention for music generation. Specifically, with the fine-grained attention, a token of a specific bar directly attends to all the tokens of the bars that are most relevant to music structures (e.g., the previous 1st, 2nd, 4th and 8th bars, selected via similarity statistics); with the coarse-grained attention, a token only attends to the summarization of the other bars rather than each token of them so as to reduce the computational cost. The advantages are two-fold. First, it can capture both music structure-related correlations via the fine-grained attention, and other contextual information via the coarse-grained attention. Second, it is efficient and can model over $3\times$ longer music sequences compared to its full-attention counterpart. Both objective and subjective experimental results demonstrate its ability to generate long music sequences with high quality and better structures.[1]

1 Introduction

Symbolic music generation aims at generating music scores automatically and has drawn more and more attention in recent years [1–3]. Since music can be represented in organized sequences of discrete tokens just like text, Transformer-based models, which have been demonstrated to work well on text generation [4, 5], are increasingly applied in music generation [3, 6–11] and have made great success. While the self-attention mechanism empowers Transformer to capture the complex correlations in music, there are two ubiquitous challenges to solve for this task: 1) *Long sequence modeling*. Music sequences are typically very long, especially for multi-instrument polyphonic music where the lengths can usually exceed $10{,}000$. The quadratic complexity of full attention limits its scalability to that length. 2) *Music structure modeling*. Music has its unique structures, where a piece can usually repeat some patterns of a previous piece, occasionally with some variations, after either a short or a long distance (see Figure 1 for an example). Successfully generating reasonable structures would make the music more realistic just like human-made music.

[*]Wei Hu and Xu Tan are the corresponding authors. This work was partially done while the first author was interning at Microsoft Research Asia.

[1]The generated music samples can be found at `https://ai-muzic.github.io/museformer`. The source code can be found at `https://github.com/microsoft/muzic`.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

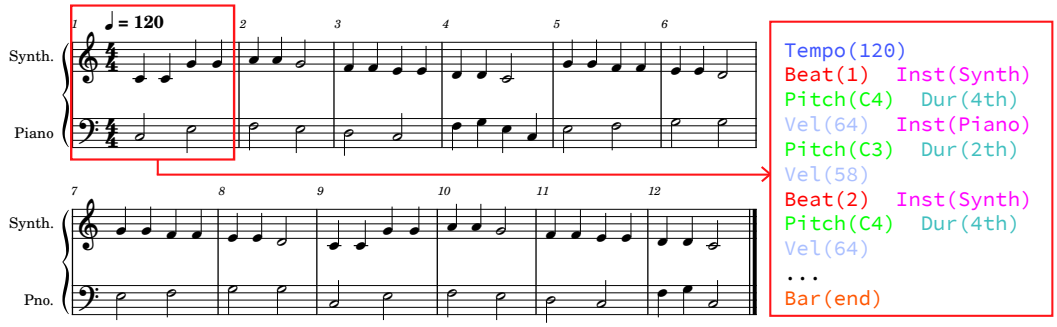


Figure 1: The music score of *Twinkle, Twinkle, Little Star* and its corresponding token representation. Every two consecutive bars on the *Synth.* track have the same rhythmic pattern, and the 9th - 12th bars repeat the 1st - 4th bars with an interval of 8 bars. Structures embodied as repetitions and imitations are common in music.

Although many Transformer variants in natural language processing (NLP) [12] have been proposed to handle long sequences (the first challenge), they cannot well model the music structures (the second challenge). According to the basic principles of these models, we can classify them into two types. The first type is called *local focusing*. Models of this type, e.g., Transformer-XL [13] and Longformer [14], mainly focus on part of the input sequence, and drop the rest tokens to reduce the cost. However, the parts that they focus on cannot contain many of the essential ranges important to music structures, and directly dropping the rest tokens may lead to the loss of some important information. The second type is called *global approximation*. Models of this type such as Linear Transformer [15] utilize linearized attention or sequence compression over the whole input sequence to approximate the token pair-wise attention. While the approximation effectively reduces the complexity, they cannot precisely capture the correlations between related parts and accordingly fall short in generating exact repetitions. We will review more existing long-sequence Transformers in §2. Directly applying these models on music generation is suboptimal, and it is desirable to design an efficient model that can well model long music sequences as well as their structures.

In this paper, we propose to unify the above two types of models, which can well fit the characteristics of music. Our motivation is based on the observation that the importance is not uniformly distributed over the music sequence, and thus we do not need to treat all the tokens equally. Intuitively, to generate music with repetitive structures, the most important information the model should directly refer to, when generating a music bar, lies in those bars that tend to be repeated in the current bar. We call these bars *structure-related bars*. For the other bars that are less important, approximation should do the trick. To this end, the proposed fine- and coarse-grained attention takes two different schemes towards different bars: with the fine-grained attention, tokens of the structure-related bars are directly attended to, to well learn the structure-related correlations; with the coarse-grained attention, information of the other bars are captured via summarization [16], rather than attending to each token of them, to decrease the computation and space complexity, and meanwhile retain the necessary information of these bars. The structure-related bars are selected according to the statistics on human-made music. They are not necessarily the most recent and consecutive bars but can include distant ones, to cater to long-term structures in music. Our main contributions in this paper are summarized as follows:

- We propose Museformer, a Transformer model with a novel fine- and coarse-grained attention for music generation. It captures the correlations of structure-related bars via fine-grained attention for learning the music structures, as well as the necessary and concentrated information of the other bars through their summarization via coarse-grained attention.
- We propose to select the structure-related bars based on similarity statistics on human-made music, which can help decrease the perplexity and generate music that exhibits better structures.
- The computation complexity and space complexity are reduced greatly to nearly linear in practice, which enables Museformer to scale up to long music sequences.
- Experimental results show that Museformer can generate music of full-song length with high quality and better structures.

2 Related Work

2.1 Symbolic Music Generation

Symbolic music generation aims to exploit machines to compose music scores automatically. It has been attracting more and more people to work on it, and the solutions continuously evolve from rule-based models [1] to probabilistic models [17, 18, 2] to deep learning models [6–11].

In recent years, the Transformer-based models have achieved great success in many text generation tasks [4, 5], thus have also been increasingly applied in the similar music generation tasks. Huang et al. [3] apply Transformer in symbolic music generation for the first time, and show that it can achieve better performance compared to previous deep learning models such as recurrent neural networks. While Transformer shows promising results in music generation, the quadratic complexity of the attention mechanism limits its applications on the typically long music sequences. To resolve this problem, researchers on symbolic music generation come up with different solutions. One popular solution is to design a new representation method to represent musical information in fewer tokens, such as compound word in [19]. While this solution indeed decreases the lengths of music sequences to a certain extent, full-song music sequences are still too long for Transformer to handle. Another solution is to use a long-sequence Transformer variant as the backbone. For example, Huang and Yang [20] use Transformer-XL [13], and Hsiao et al. [19] use Linear Transformer [15]. Although these models can process long sequences, they are initially designed for NLP tasks and cannot well model the music structures. We will introduce and discuss more about the long-sequence Transformers in the following section.

2.2 Long-Sequence Transformers

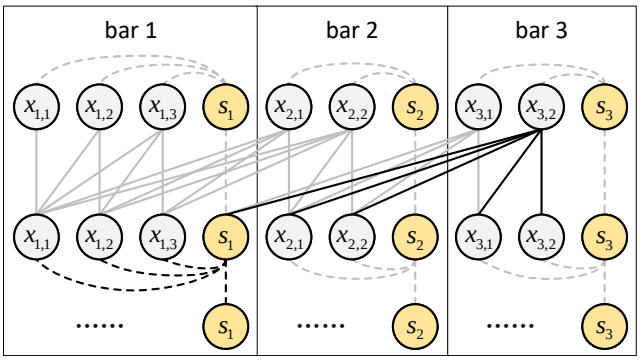
Many Transformer variants have been proposed to tackle long-sequence tasks [12], which in general can be categorized into the following types. 1) Recurrent Transformer, which encodes sequences chunk by chunk [13, 21, 22]. 2) Sparse attention, which sparsifies the attention layout with either predefined patterns [14, 23, 24, 16, 25], or with content-based patterns [26–30]. 3) Linearized attention, which replaces the exponent of inner product of features with multiplication of feature maps [15, 31–33]. 4) Compression-based attention, which reduces the number of queries or key-value pairs by compressing the contextual representations [34–36]. In addition, some models also try to combine multiple techniques, such as Compressive Transformer [21] that combines recurrent and compression-based methods, Poolingformer [35] and Transformer-LS [37] that combine sparse attention and compression-based methods.

Existing works on music generation directly adopt some of those long-sequence Transformers to process long music sequences, but it is suboptimal due to the unique structures of music. In general, music has many repetitive or similar pieces, many of which are distant to each other, and the distance is measured by time units such as bar or beat instead of the number of tokens. Therefore, the receptive fields of the existing recurrent Transformers or sparse attention methods cannot cover the many ranges of structure-related content. Although the linearized or compression-based attention can cover the whole sequence, they do not precisely capture the correlation between each pair of tokens and may fall short in generating repeating or similar music pieces.

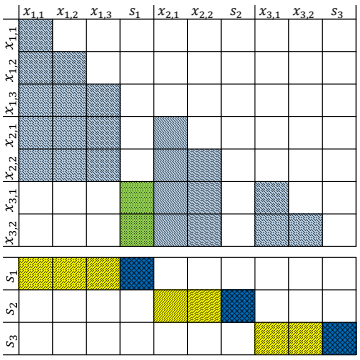
3 Museformer

In order to well model long music sequences as well as their music structures, we propose Museformer. The general idea is that we do not need to focus on the whole sequence with the same importance level given that the complexity of pair-wise full attention is unacceptably high, but instead we combine two different attention schemes – fine-grained attention for the structure-related bars, and coarse-grained attention for the other bars. Museformer basically follows the original Transformer architecture [38], and a novel fine- and coarse-grained attention (FC-Attention) is designed to replace the original self-attention module to tackle the challenges in long music sequence modeling.

Museformer takes a music token sequence $X = X_1, \ldots, X_b$ as input, where b denotes the number of music bars, and the i-th bar $X_i = x_{i,1}, \ldots, x_{i,|X_i|}$ contains $|X_i|$ tokens. For the i-th bar where $i = 1, \ldots, b$, we insert a *summary token* s_i after it to facilitate local information aggregation in FC-Attention. The summary token sequence is denoted by $S = s_1, \ldots, s_b$. After the insertion, the



(a) Information flow. The dashed/solid lines depict the summarization/aggregation step. The dark lines represent the information flow towards $x_{3,2}$.



(b) Attention layouts. The bottom/top part corresponds to the summarization/aggregation step.

Figure 2: The fine- and coarse-grained attention. It shows a toy example of 3 bars, where for each bar, only the previous 1st bar is regarded as the structure-related bar.

overall token sequence becomes $X_1, s_1 \cdots, X_b, s_b$. The embedding layer embeds the overall token sequence into a vector space, and concatenates bar embedding and beat embedding as the position information [10], followed by a linear projection. The Museformer layers then model the contextual representation, and the hidden state output by the last layer is fed into a softmax classifier to predict the next token.

In the following part, we first give a brief formulation to the attention mechanism (§3.1). Then, we introduce the details of FC-Attention (§3.2), followed by the selection of the structure-related bars (§3.3). Finally, we discuss the merits of Museformer (§3.4).

3.1 Preliminary: Attention

The attention mechanism [38] is the basis of FC-Attention. It receives two sequential inputs, namely source $\boldsymbol{X} \in \mathbb{R}^{n_{\text{src}} \times d}$ and target $\boldsymbol{X}' \in \mathbb{R}^{n_{\text{tgt}} \times d}$, where n_{src} and n_{tgt} are the sequence lengths of source and target respectively, d is the embedding dimension. It computes contextual representation for each $\boldsymbol{x}'_i \in \mathbb{R}^{1 \times d}$ in target $\boldsymbol{X}'$ as

$$\text{Attn}(\boldsymbol{x}'_i, \boldsymbol{X}) = \text{softmax}\Big(\frac{\boldsymbol{x}'_i \boldsymbol{W}_Q (\boldsymbol{X}\boldsymbol{W}_K)^T}{\sqrt{d}}\Big)\boldsymbol{X}\boldsymbol{W}_V, \tag{1}$$

where $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V \in \mathbb{R}^{d \times d}$ are the trainable parameters. In practice, we employ the multi-head version of attention, but omit it from the equation for simplicity.

3.2 Fine- and Coarse-Grained Attention

The basic idea of FC-Attention is that, instead of directly attending to all the tokens which causes the quadratic complexity, a token of a specific bar only directly attends to the structure-related bars that are essential for generating structured music (fine-grained attention), and for the other bars, the token only attends to their summary tokens to obtain concentrated information (coarse-grained attention). To achieve this, we first summarize the local information of each bar through the *summarization* step, and then aggregate the fine-grained and coarse-grained information through the *aggregation* step. Figure 2 visualizes the process.

Summarization In the summarization step, we aggregate the information of each bar into the corresponding summary token. For the i-th bar, given the representation of summary token $\boldsymbol{s}_i \in \mathbb{R}^{1 \times d}$, and that of the music tokens $\boldsymbol{X}_i = [\boldsymbol{x}_{i,1}, \ldots, \boldsymbol{x}_{i,|X_i|}] \in \mathbb{R}^{|X_i| \times d}$, the summarization of this bar is

$$\tilde{\boldsymbol{s}}_i = \text{Attn}\big(\boldsymbol{s}_i, [\boldsymbol{X}_i, \boldsymbol{s}_i]\big), \tag{2}$$

where $\text{Attn}(\cdot)$ is defined in Equation 1, $[\cdot]$ is the concatenation operation. In this step, each summary token attends to music tokens of its corresponding bar as well as the summary token itself.

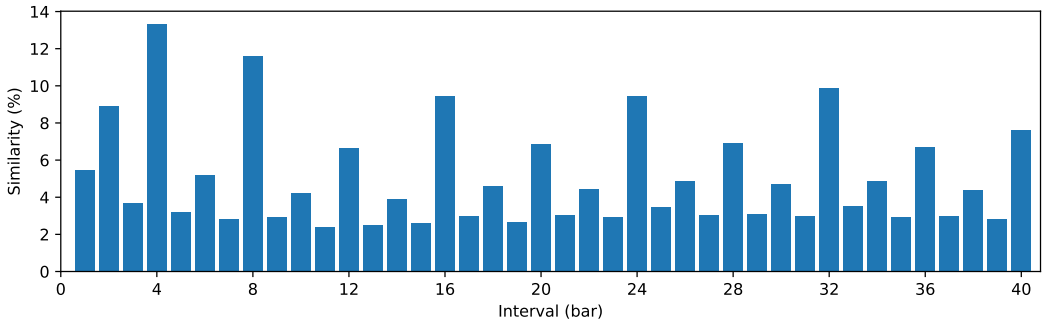


Figure 3: The similarity distribution of the training set used in our experiments. It is calculated over the melody track because melody best shows the structure of a song. We omit the similarity when $t = 0$ because the similarity between a bar and itself is always 1.0.

Aggregation In the aggregation step, we aggregate the information of the tokens belonging to the structure-related bars or within the current bar, as well as the summarization of the other bars, so as to update the contextual representations of music tokens. The updated representation for $x_{i,j}$ is

$$\tilde{\boldsymbol{x}}_{i,j} = \text{Attn}\Big(\boldsymbol{x}_{i,j}, [\boldsymbol{X}_{R(i)}, \boldsymbol{X}_{i,k\leq j}, \tilde{\boldsymbol{S}}_{\bar{R}(i)}]\Big), \tag{3}$$

where $R(i)$ is the set of the indices of the structure-related bars with respect to the i-th bar, and $\bar{R}(i)$ is the set of the indices of other previous bars. $\boldsymbol{X}_{R(i)}$ is the matrix formed by stacking $\{\mathbf{X}_i \mid i \in R(i)\}$. Similarly, $\tilde{\boldsymbol{S}}_{\bar{R}(i)}$ is the matrix formed by stacking $\{\tilde{\boldsymbol{s}}_i \mid i \in \bar{R}(i)\}$. $\boldsymbol{X}_{i,k\leq j}$ is the matrix formed by stacking $\{\boldsymbol{x}_{i,k} \mid \boldsymbol{x}_{i,k} \in \boldsymbol{X}_i,\ k \leq j\}$. In other words, the music token $x_{i,j}$ only attends to those music tokens that belong to the structure-related bars, and its previous tokens within the current bar. For the other bars, it only attends to their summary tokens.

3.3 Structure-Related Bar Selection

Considering that the structure-related bars are expected to contain those bars that are most likely to be repeated by the current bar to be generated, we propose to pinpoint them by similarity statistics, and the bars with high similarities should be selected. Specifically, for each song in the training set, we calculate the similarity between each pair of the bars, which is defined as

$$l_{i,j} = \frac{|N(i) \cap N(j)|}{|N(i) \cup N(j)|}, \tag{4}$$

where $N(i)$ is the set of music notes within the i-th bar, and two notes are considered equal when their pitches, duration, and onset positions within their bars are all the same. The value of $l_{i,j}$ ranges from 0.0 to 1.0. If two bars are exactly the same, this value equals 1.0. We then calculate the average similarity of those bar pairs whose intervals are t over the training set D, which is formulated as

$$L_t = \text{Mean}(\sum_{D} \sum_{j=i+t} l_{i,j}). \tag{5}$$

In this paper, we call the distribution of the similarity with respect to the bar intervals the similarity distribution, and show that of the training data in Figure 3. As we can see, it shows an obvious periodical pattern – a music bar tends to be more similar to its previous 2 bars, and also to the previous 4-th bar or its multipliers in most cases. We further conduct the similarity statistics on different datasets involving music of various genres and styles. The results shown in Appendix A interestingly indicate that this pattern is universally applicable to the music of the great diversity. We believe that it can be regarded as a general rule applicable to most music in our daily life.

Based on the statistical results, we carefully select 8 bars – the previous 1st, 2nd, 4th, 8th, 12th, 16th, 24th, and 32nd bars, as the default structured-related bars, since they can cover the most similar bars in most cases. The number of selected structure-related bars is a trade-off between efficiency and information richness. One can select more or other bars according to the computation resources and

the specific similarity distribution of the used dataset. We have to admit that the selected bars cannot always cater to any song, but as long as it covers the most cases, it can already enable Museformer to generate music with better structures.

3.4 Merits of Museformer

We explain why Museformer is suitable for the music generation task.

First, the receptive fields of the fine-grained attention comply with the characteristics of music and can cover most of the structure-related information. Unlike previous models that adopt fixed token-level patterns [14, 37] or learn content-based patterns [26, 30], which may not cover the most important ranges to be referred to for generating music structures, we directly use the structure-related bars derived from human-made music. These bars can include both neighboring ones and distant ones, which enables Museformer to generate both short-term and long-term music structures.

Second, the coarse-grained attention can preserve necessary information for generating better music. In general, FC-Attention can be categorized into sparse attention, but unlike the conventional sparse attention that simply drops a large amount of information, which may limit the model's capability especially for music where mutual connections abound, the coarse-grained attention preserves the information of the other bars so as to provide rich clues for generation.

Third, The combination of the fine- and coarse-grained attention enables Museformer to handle long music sequences efficiently. Compared to the number of all the music tokens, the number of tokens in the structure-related bars and the summary tokens is much smaller, so the memory consumption and the running time are greatly reduced. It is essential for this task where sequences are quite long.

4 Experiments

4.1 Experiment Settings

We give a brief introduction to the experiment settings. Please refer to Appendix B for more details.

Dataset and Music Representation We conduct experiments on the widely-used Lakh MIDI (LMD) dataset [39], which contains multi-instrument music in the format of MIDI. After preprocessing, the final dataset contains $29{,}940$ songs ($1{,}727$ hours), each song contains 95 bars on average, and the instruments are normalized into the 6 basic ones [10] namely square synthesizer, piano, guitar, string, bass and drum, with the square synthesizer playing the melody. We use a REMI-like representation method [20] to transfer MIDIs into token sequences, where the musical information (instrument, bar line, note position, pitch, duration, etc.) is represented in separate tokens (see Figure 1 for an example). Each song is represented by $15{,}042$ tokens on average, and each bar is represented by 158 tokens on average. We randomly split the songs by $8/1/1$ for training/validation/test, respectively.

Implementation Museformer is implemented with PyTorch[2] and fairseq[3]. For the efficient computation of FC-Attention, we write CUDA kernels to generate the attention layouts for each sample, and then transfer the layouts into a blocksparse form and compute with SparTA[4] [40].

Model and Training Configurations The core parameters of Museformer include layer number $= 4$, hidden size $= 512$, number of attention heads $= 8$, and FFN hidden size $= 2{,}048$. During training, the batch size is set to 4 songs. Following [38], we use the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\varepsilon = 10^{-9}$, and the learning rate is warmed up over the first $16{,}000$ steps linearly to a peak value of 5×10^{-4}, and then decreases based on the inverse square root of the steps. The L2 weight decay is set to 0.01. During inference, we use top-k sampling with $k = 8$. Generation continues until the end-of-sentence token is generated or it reaches the maximum length of $20{,}480$.

Compared Models We compare Museformer with 4 representative Transformer-based models, most of which have been adopted in music generation:

[2] https://pytorch.org

[3] https://github.com/pytorch/fairseq

[4] https://github.com/microsoft/SparTA

- **Music Transformer** [3]: a vanilla Transformer with a memory-efficient "skewing" relative position embedding implementation.
- **Transformer-XL** [13]: a recurrent Transformer that encodes the sequence chunk by chunk and uses the gradient-stopped representations of previous chunks as the memory.
- **Longformer** [14]: a model with a sliding window sparse attention.
- **Linear Transformer** [15]: a model that uses a kernel-based attention of linear complexity.

All the compared models are set with comparable hyper-parameters as Museformer. For Music Transformer that uses the full attention and thus cannot model long sequences at once due to memory limit, we chunk each song into multiple samples during training, and apply the model to generate long sequences during validation and inference to test its generalization on long music sequences.

Objective Evaluation We use the following objective metrics to evaluate the models:

- **Perplexity (PPL)**: a common metric to measure whether a generative model can correctly predict future tokens. The smaller, the better. To see the models' performances on different lengths, it is calculated on the first $1{,}024$, $5{,}120$, or $10{,}240$ tokens of each sample.
- **Similarity Error (SE)**: the error between the similarity distribution of training data and generated music, to evaluate the models' ability to generate music with realistic structures. It is defined as

$$\text{SE} = \frac{1}{T}\sum_{t=1}^{T}|\hat{L}_t - L_t|, \tag{6}$$

  where $\hat{L}_t$ and L_t are the average similarities (defined in Equation 5) of the generated music and the training data respectively. We set $T = 40$ in our experiments, and $\hat{L}_t$ is calculated on 100 generated music pieces for each model. The smaller the value is, the more similar the structures of the generated music are to human-made music.

Subjective Evaluation The most canonical way to evaluate a music generation model is human listening test. We apply each model to randomly generate 100 music pieces, and invite 10 people, where 7 of them have music-related learning experiences, to score these music pieces. Specifically, for each participant, we randomly construct 5 groups, where each group contains 5 music pieces that are generated by Museformer and the 4 compared models, respectively. Participants are asked to score these music pieces from 1 (lowest) to 10 (highest) over the following subjective metrics:

- **Musicality**: whether it is pleasant and interesting, and real enough just like human-made music.
- **Short-term structure**: whether it shows good structures in neighboring content, such as good repetitions and reasonable development.
- **Long-term structure**: whether it shows good structures in long distance, such as song-level repetitions and long-distance connections.
- **Overall**: an overall score. We also calculate **preference score** based on this overall score, which is defined as the ratio of winning times (obtaining the highest overall score within a group).

4.2 Comparison with Previous Models

Table 1 shows the results of the objective evaluation. As we can see: 1) Music Transformer achieves a comparable PPL as other models on short music sequences ($1{,}024$ tokens), but undergoes a severe deterioration on longer sequences, which implies that a model trained on short music sequences cannot well generalize to long music sequences. Accordingly, an appropriate long-sequence Transformer model is needed for modeling the long music sequences. 2) Although the receptive field of Linear Transformer can cover the whole sequence, its PPLs show no superiority compared to other models, which may because that the kernel-based attention falls short on accurately capturing the complex correlations of music. 3) The proposed Museformer can consistently achieve the best PPLs on various sequence lengths, especially on a larger length, which demonstrates the effectiveness of Museformer on the music generation task. 4) The results on SE demonstrate that the music generated by Museformer has the structures that are most similar to the human-made music. We provide the similarity distribution for each model and more discussion in Appendix C.

Table 1: The results of objective evaluation and ablation study. The numbers in the parentheses for PPLs are sequence lengths.

	PPL (1,024)	PPL (5,120)	PPL (10,240)	SE (%)
Music Transformer	1.66	1.77	2.55	2.49
Transformer-XL	**1.64**	1.45	1.43	15.66
Longformer	1.65	1.46	1.45	5.25
Linear Transformer	1.86	1.67	1.64	1.97
Museformer (ours)	**1.64**	**1.41**	**1.35**	**0.95**
w/o coarse-grained	1.65	1.42	1.38	1.08
w/o bar selection	1.65	1.43	1.39	6.39

Table 2: The results of subjective evaluation. ST and LT stand for short-term and long-term respectively. For all the subjective metrics, mean $\pm$ standard deviation is reported. Pref stands for preference score.

	Musicality	ST structure	LT structure	Overall	Pref
Music Transformer	6.00 ± 2.21	6.90 ± 1.76	5.30 ± 2.58	5.90 ± 1.90	0.20
Transformer-XL	6.10 ± 2.19	7.40 ± 1.81	6.26 ± 2.78	6.44 ± 2.01	0.34
Longformer	6.46 ± 1.81	7.60 ± 1.47	6.18 ± 2.54	6.44 ± 1.72	0.24
Linear Transformer	6.06 ± 1.99	6.92 ± 2.03	5.78 ± 2.64	6.30 ± 1.84	0.24
Museformer (ours)	$\mathbf{6.88 \pm 1.95}$	$\mathbf{7.86 \pm 1.51}$	$\mathbf{6.72 \pm 2.74}$	$\mathbf{7.12 \pm 1.81}$	**0.46**

Furthermore, we present the results of the subjective evaluation in Table 2, which show that Museformer achieves the best performance. Specifically, 1) Museformer gets the highest scores on all the metrics. 2) On the structure-related metrics, especially the long-term structure, Museformer exceeds other models by a large gap, indicating that the proposed FC-Attention can empower the model to capture the correlations in distant bars.

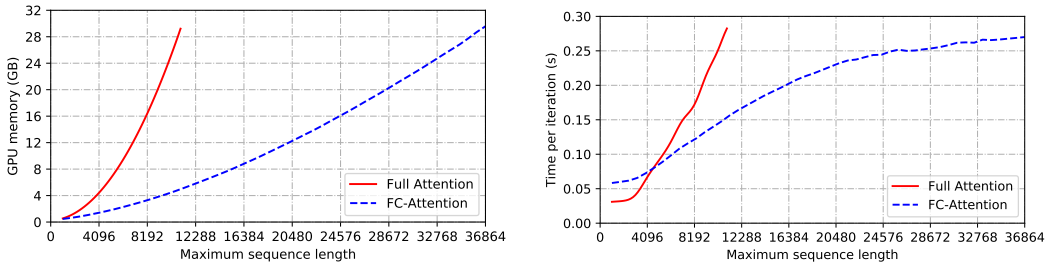
We further do pairwise comparisons over the subjective evaluation results. Table 3 shows the number of wins/ties/losses based on the overall scores, as well as the p-values of the Wilcoxon signed rank test. Museformer obtains more wins than the compared models, and the p-values indicate that Museformer achieves statistically significant improvements ($p < 0.05$).

4.3 Ablation Study

We verify the effectiveness of the Museformer components as follows: 1) To see the effectiveness of the coarse-grained attention, we compare Museformer with the setting **w/o coarse-grained**, where each music token attends to the music tokens in the structure-related bars and its previous ones in the current bars, and no summary token of any bar. 2) To see the effectiveness of the structure-related bar selection for the fine-grained attention, we compare Museformer with the setting **w/o bar selection**, where we use the most recent 8 bars for the fine-grained attention instead of the structure-related bars selected via the similarity statistics.

Table 3: The results of pairwise comparisons on the subjective overall scores.

	Wins	Ties	Losses	p-value
Museformer VS Music Transformer	33	7	10	0.0003
Museformer VS Transformer-XL	25	14	11	0.0375
Museformer VS Longformer	28	8	14	0.0424
Museformer VS Linear Transformer	29	7	14	0.0128



(a) GPU memory consumption.

(b) Training time per iteration.

Figure 4: Memory consumption and running time of FC-Attention compared to the full attention.

From the results presented in the bottom block of Table 1, we can observe that: 1) Museformer consistently achieves better performances on both PPLs and SE compared to the two ablation settings, which demonstrates the effectiveness of the coarse-grained attention and the structure-related bar selection. 2) The coarse-grained attention preserves the necessary information of the bars other than the structured-related bars, and can consistently help decrease the PPLs. 3) The contribution of the structure-related bar selection grows when the sequence length is getting longer, and this is reasonable because that in a longer music sequence, there tends to be more long-term structures, and the fine-grained attention can directly capture these correlations. 4) Embodied on SE, the bar selection is contributory to the structures of generated music. Please refer to Appendix C for more details. In addition to the above two settings, we have also tried disabling the fine-grained attention for previous bars, where a music token only directly attends to its previous tokens within its bar, and attends to all the previous bars via the summary tokens. The result indicates that only summarized information and no precise token-level information for the previous bars is insufficient for the model to generate coherent music.

4.4 Complexity Analysis

Suppose the sequence length is n, the average bar number is b, the average bar length is m, and the number of selected structure-related bars is k. For FC-Attention, the time complexity of the summarization step is $O(n)$, and that of the aggregation step is $O\big((km+b)n\big)$, so the overall complexity is $O\big((km+b)n\big) = O\big((km+n/m)n\big)$. Although the complexity is still proportional to the square of n, the typically large divider m greatly reduces the complexity. Thus, the specific complexity is between linear and quadratic. Note that the computation of FC-Attention depends on the input content, i.e., the number of bars and the number of tokens in each individual bar, the complexity cannot be precisely formulated, and the efficiency in real applications is more noteworthy.

To see the efficiency of FC-Attention in real applications, we train Museformer and its full attention counterpart on the validation set, and record their memory consumption and running time. With batch size $= 1$, we increase the maximum sequence length until we use up the 32GB memory of an Nvidia V100 GPU, and record the peak memory consumption[5] and the running time for one-epoch training. Figure 4 shows the results. We observe that, the memory consumption of Museformer increases at a nearly linear rate with respect to the maximum sequence length, and can process over $3\times$ longer music sequences than its full attention counterpart, making it capable of generating full-song music. Museformer also achieves faster training time when the maximum sequence length is greater than around $5{,}000$, which is common for music sequences.

4.5 Case Study

Figure 5 shows a snippet of a song generated by Museformer. As we can see, on the *Strings* track, the 13th - 16th bars repeat the 9th - 12th bars with an interval of 4 bars, which is a short-term structure. The 25th - 32nd bars repeat the 9th - 16th bars with an interval of 16 bars, and there are reasonable variations in the 27th bar compared to the 11th bar, which shows a long-term structure. This case demonstrates that Museformer can generate music with both short-term and long-term structures, and

[5] Obtained by torch.cuda.max_memory_allocated().

not only exact repetitions but also some variations. Other than the two exhibited tracks, other tracks, such as piano and drum, have more variations on the two similar segments, which creates reasonable development of music. Please refer to our demo page for more information.

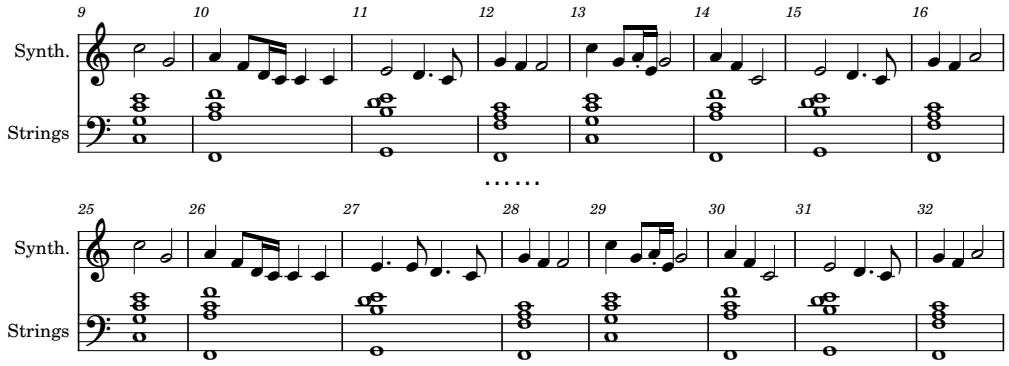


Figure 5: A snippet of a generated song.

5 Conclusion

To solve the long sequence modeling and music structure modeling challenges in symbolic music generation, we propose Museformer with a novel fine- and coarse-grained attention. The fine-grained attention is applied to structure-related bars for specifically learning the structure-related correlations, while the coarse-grained attention is applied to the summarization of the other bars for getting a sketch of them. We propose to select the structure-related bars via bar-pair similarity statistics. Experimental results show that Museformer is efficient and performs best in generating full-song level music with good structures.

Museformer is not perfect yet, and we would like to discuss about its limitations and possible future explorations. First, since Museformer takes random samplings during inference and does not receive manual control, it can hardly ensure every generated music piece to be well-structured in an expected way. Techniques to enhance its reliability and controllability can be further explored. Furthermore, the musicality and creativity of the generated music is still far behind that of the human-made music, which remains a problem to all the existing music generation models. We believe that more sophisticated music representation and large models trained on large-scale data can help alleviate this problem. Finally, we anticipate Museformer's adaptation to more tasks and domains. It can be easily and reasonably applied on music understanding tasks. For NLP tasks, since natural languages do not have the periodical patterns as music, how to determine the semantically related sentences, paragraphs, or documents is an interesting challenge to solve.

Acknowledgments

This work is supported by National Natural Science Foundation of China (No. 61872172).

References

[1] H. A. G. Salas, A. F. Gelbukh, H. Calvo, and F. G. Soria, "Automatic music composition with simple probabilistic generative grammars," *Polibits*, vol. 44, pp. 59–65, 2011.

[2] A. K. Yanchenko and S. Mukherjee, "Classical music composition using state space models," *arXiv preprint arXiv:1708.03822*, 2017.

[3] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, I. Simon, C. Hawthorne, N. Shazeer, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck, "Music transformer: Generating music with long-term structure," in *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

[4] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," *OpenAI*, 2018.

[5] J. Li, T. Tang, W. X. Zhao, and J.-R. Wen, "Pretrained language model for text generation: A survey," in *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2021, pp. 4492–4499.

[6] C. Donahue, H. H. Mao, Y. E. Li, G. W. Cottrell, and J. J. McAuley, "Lakhnes: Improving multi-instrumental music generation with cross-domain pre-training," in *Proceedings of International Society for Music Information Retrieval Conference (ISMIR)*, 2019, pp. 685–692.

[7] A. Muhamed, L. Li, X. Shi, S. Yaddanapudi, W. Chi, D. Jackson, R. Suresh, Z. C. Lipton, and A. J. Smola, "Symbolic music generation with transformer-gans," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 35, 2021, pp. 408–417.

[8] A. Roberts, J. Engel, C. Raffel, C. Hawthorne, and D. Eck, "A hierarchical latent vector model for learning long-term structure in music," in *Proceedings of International Conference on Machine Learning (ICML)*, vol. 80, 2018, pp. 4364–4373.

[9] Z. Ju, P. Lu, X. Tan, R. Wang, C. Zhang, S. Wu, K. Zhang, X. Li, T. Qin, and T.-Y. Liu, "Telemelody: Lyric-to-melody generation with a template-based two-stage method," *arXiv preprint arXiv:2109.09617*, 2021.

[10] Y. Ren, J. He, X. Tan, T. Qin, Z. Zhao, and T.-Y. Liu, "Popmag: Pop music accompaniment generation," in *Proceedings of ACM International Conference on Multimedia (MM)*, 2020, pp. 1198–1206.

[11] Z. Sheng, K. Song, X. Tan, Y. Ren, W. Ye, S. Zhang, and T. Qin, "Songmass: Automatic song writing with pre-training and alignment constraint," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2021, pp. 13 798–13 805.

[12] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A survey of transformers," *arXiv preprint arXiv:2106.04554*, 2021.

[13] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. Le, and R. Salakhutdinov, "Transformer-XL: Attentive language models beyond a fixed-length context," in *Proceedings of Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019, pp. 2978–2988.

[14] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020.

[15] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers are RNNs: Fast autoregressive transformers with linear attention," in *Proceedings of International Conference on Machine Learning (ICML)*, 2020, pp. 5156–5165.

[16] Z. Ye, Q. Guo, Q. Gan, X. Qiu, and Z. Zhang, "Bp-transformer: Modelling long-range context via binary partitioning," *arXiv preprint arXiv:1911.04070*, 2019.

[17] M. M. Farbood and B. Schöner, "Analysis and synthesis of palestrina-style counterpoint using markov chains," in *Proceedings of International Computer Music Conference (ICMC)*, 2001.

[18] M. Allan, "Harmonising chorales in the style of johann sebastian bach," *Master's*, 2002.

[19] W.-Y. Hsiao, J.-Y. Liu, Y.-C. Yeh, and Y.-H. Yang, "Compound word transformer: Learning to compose full-song music over dynamic directed hypergraphs," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 35, 2021, pp. 178–186.

[20] Y.-S. Huang and Y.-H. Yang, "Pop music transformer: Beat-based modeling and generation of expressive pop piano compositions," in *Proceedings of ACM International Conference on Multimedia (MM)*, 2020, pp. 1180–1188.

[21] J. W. Rae, A. Potapenko, S. M. Jayakumar, C. Hillier, and T. P. Lillicrap, "Compressive transformers for long-range sequence modelling," in *Proceedings of International Conference on Learning Representations (ICLR)*, 2020.

[22] S. Ding, J. Shang, S. Wang, Y. Sun, H. Tian, H. Wu, and H. Wang, "ERNIE-Doc: A retrospective long-document modeling transformer," in *Proceedings of Annual Meeting of the Association for Computational Linguistics and International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, 2021, pp. 2914–2927.

[23] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019.

[24] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed, "Big bird: Transformers for longer sequences," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 17 283–17 297.

[25] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran, "Image transformer," in *Proceedings of International Conference on Machine Learning (ICML)*, vol. 80, 2018, pp. 4055–4064.

[26] N. Kitaev, L. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," in *Proceedings of International Conference on Learning Representations (ICLR)*, 2020.

[27] A. Roy, M. Saffar, A. Vaswani, and D. Grangier, "Efficient content-based sparse attention with routing transformers," *Transactions of the Association for Computational Linguistics (TACL)*, vol. 9, pp. 53–68, 2021.

[28] X. Li, Y. Meng, M. Zhou, Q. Han, F. Wu, and J. Li, "Sac: Accelerating and structuring self-attention via sparse adaptive connection," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, 2020, pp. 16 997–17 008.

[29] Y. Tay, D. Bahri, L. Yang, D. Metzler, and D.-C. Juan, "Sparse Sinkhorn attention," in *Proceedings of International Conference on Machine Learning (ICML)*, vol. 119, 2020, pp. 9438–9447.

[30] C. Wu, F. Wu, T. Qi, B. Jiao, D. Jiang, Y. Huang, and X. Xie, "Smart bird: Learnable sparse attention for efficient and effective transformer," *arXiv preprint arXiv:2108.09193*, 2021.

[31] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, D. Belanger, L. Colwell *et al.*, "Masked language modeling for proteins via linearly scalable long-context transformers," *arXiv preprint arXiv:2006.03555*, 2020.

[32] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser *et al.*, "Rethinking attention with performers," *arXiv preprint arXiv:2009.14794*, 2020.

[33] H. Peng, N. Pappas, D. Yogatama, R. Schwartz, N. Smith, and L. Kong, "Random feature attention," in *Proceedings of International Conference on Learning Representations (ICLR)*, 2021.

[34] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," *arXiv preprint arXiv:2006.04768*, 2020.

[35] H. Zhang, Y. Gong, Y. Shen, W. Li, J. Lv, N. Duan, and W. Chen, "Poolingformer: Long document modeling with pooling attention," in *Proceedings of International Conference on Machine Learning (ICML)*, vol. 139, 2021, pp. 12 437–12 446.

[36] A. Vyas, A. Katharopoulos, and F. Fleuret, "Fast transformers with clustered attention," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, 2020, pp. 21 665–21 674.

[37] C. Zhu, W. Ping, C. Xiao, M. Shoeybi, T. Goldstein, A. Anandkumar, and B. Catanzaro, "Long-short transformer: Efficient transformers for language and vision," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, 2021, pp. 17 723–17 736.

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. ukasz Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 5998–6008.

[39] C. Raffel, *Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching*. Columbia University, 2016.

[40] N. Zheng, B. Lin, Q. Zhang, L. Ma, Y. Yang, F. Yang, Y. Wang, M. Yang, and L. Zhou, "Sparta: Deep-learning model sparsity via tensor-with-sparsity-attribute," in *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2022, pp. 213–232.

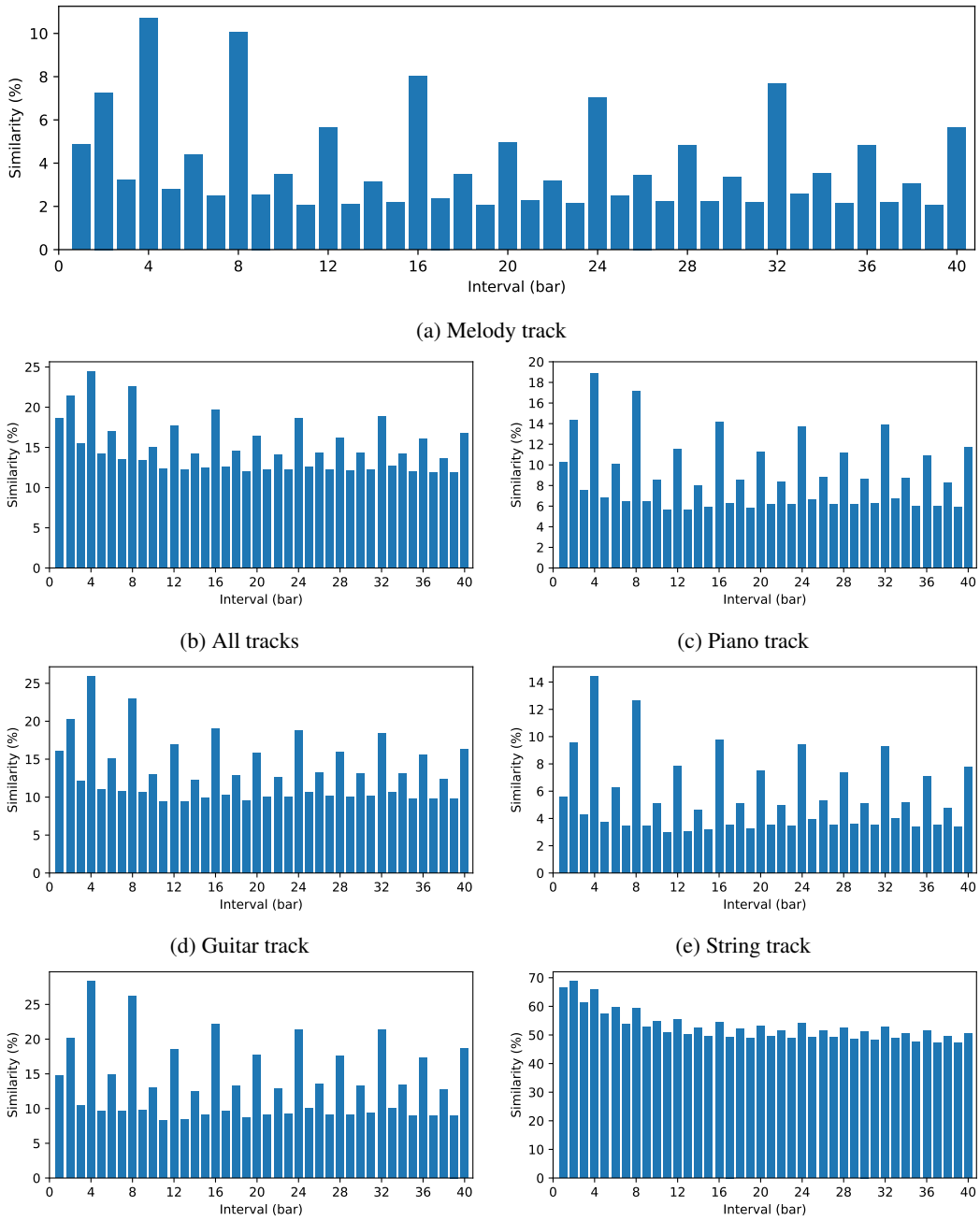
[41] J. Liu, Y. Dong, Z. Cheng, X. Zhang, X. Li, F. Yu, and M. Sun, "Symphony generation with permutation invariant language model," *arXiv preprint arXiv:2205.05448*, 2022.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] We do not see any potential negative societal impacts of our work.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [Yes]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Similarity Statistics on Different Datasets

We conduct the similarity statistics introduced in §3.3 on the whole LMD dataset that we use, and exhibit the results in Figure 6. We can observe that the periodical structure pattern, that a music bar tends to be more similar to its previous 2 bars, and also to the previous 4-th bar or its multipliers in most cases, is also satisfied on this set of music. Specifically, this rule holds on all of the instrument tracks except the drum track. This makes sense because the percussive instruments usually play a same rhythmic pattern over and over throughout a song.



(a) Melody track

(b) All tracks

(c) Piano track

(d) Guitar track

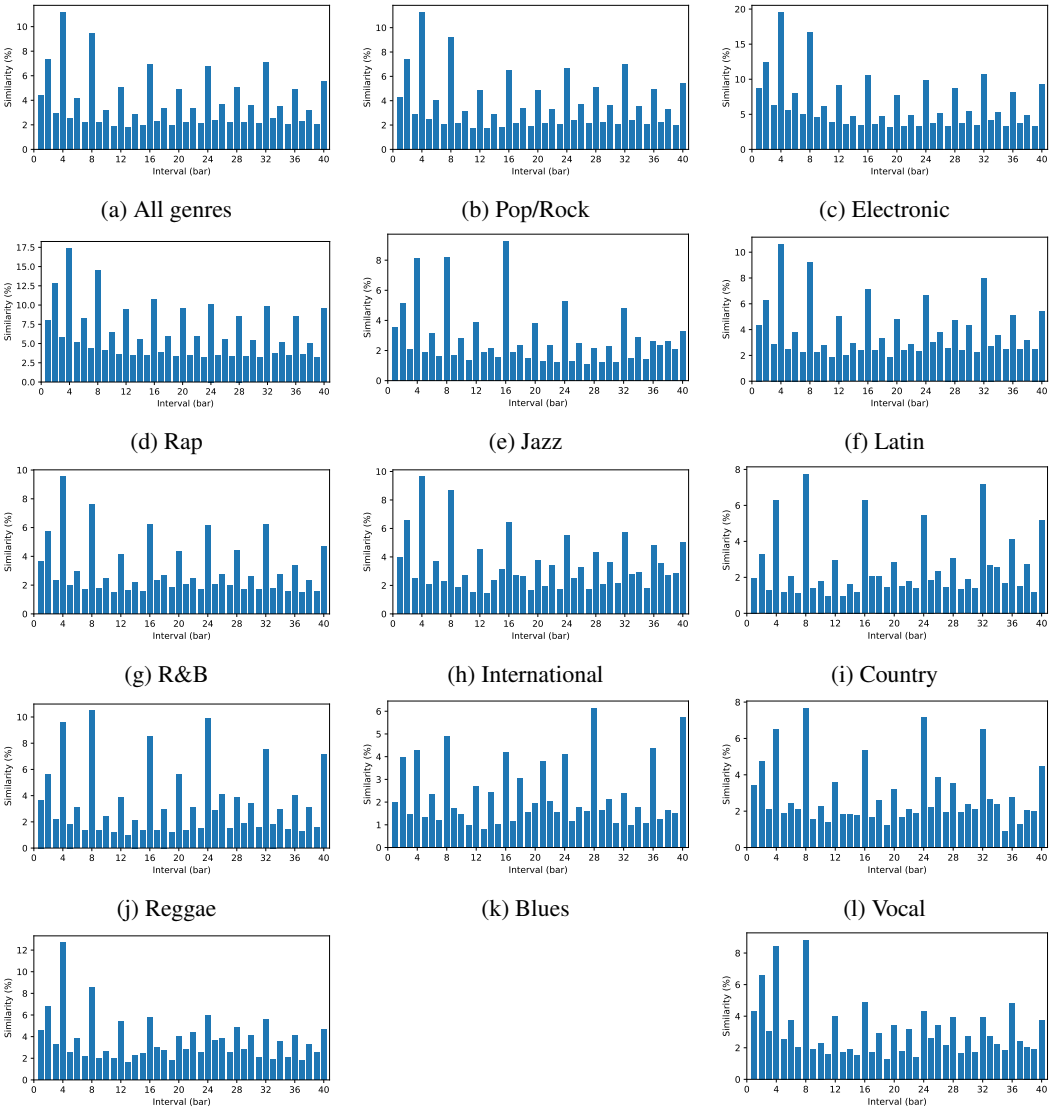
(e) String track

(f) Bass track

(g) Drum track

Figure 6: The similarity distribution of the LMD dataset we use.

To see the structure pattern of music of different genres, we conduct the same similarity statistics on the Top-MAGD dataset [6], which annotates altogether 13 music genres to the songs. Figure 7 shows that although different genres have their specific distributions, the general pattern is still applicable to all of these genres.



(a) All genres (b) Pop/Rock (c) Electronic

(d) Rap (e) Jazz (f) Latin

(g) R&B (h) International (i) Country

(j) Reggae (k) Blues (l) Vocal

(m) Folk (n) New Age

Figure 7: The similarity distribution of the melody track of the different genres in TopMAGD.

To see whether the pattern still holds on other style of music, we conduct the statistics on the Symphony dataset [41], and exhibit the distribution in Figure 8. Since it is not easy to tell the melody tracks for the symphony music, the reported result is calculated over all the tracks. Due to the existence of some instruments that play more repetitions, e.g., drums, the differences among the similarities over the intervals are not that significant, but it still presents the same tendency.

[6] `http://www.ifs.tuwien.ac.at/mir/msd/TopMAGD.html`

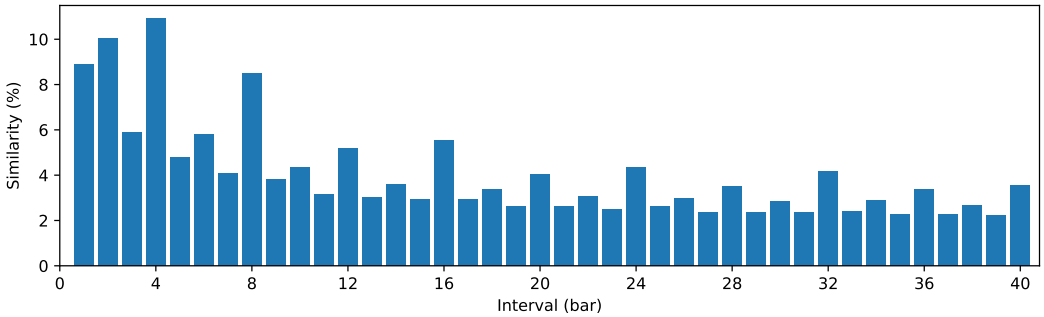


Figure 8: The similarity distribution of all tracks of the Symphony dataset.

B Details of Experiment Settings

B.1 Dataset Construction

We use the LMD dataset [39] in our experiments, and perform the following cleaning and processing to ensure the data quality:

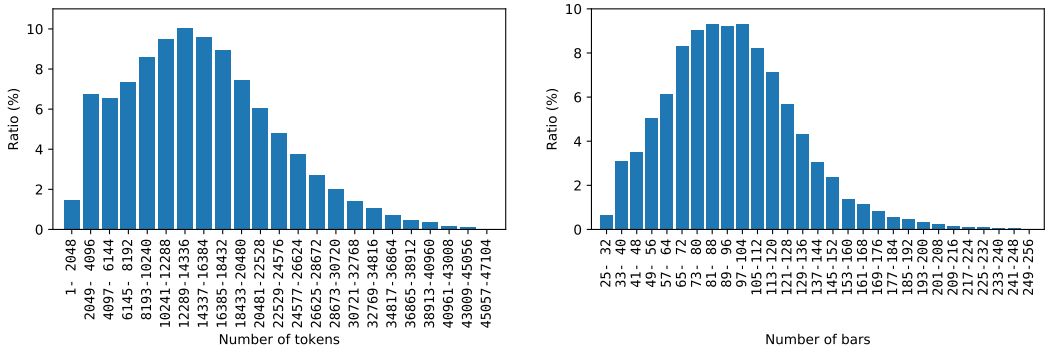
- **Track Compression**: We first compress various tracks into 6 tracks [10], namely square synthesizer, piano, guitar, string, bass and drum, with the square synthesizer playing the melody.
- **Note Position and Duration Normalization**: To ensure that the bar splitting is correct and the duration of notes recorded in MIDI files conforms to the musically perfect duration (e.g., quarter notes held for exact 1 beat, 8-th notes held for exact half a beat), we use MuseScore[7] to normalize the note position and duration.
- **Data Filtering**: Since this dataset is crawled from the Internet and contains many samples of low quality, we use a set of heuristic rules presented in Table 4 to filter them out and keep the samples of good quality and reasonable lengths.
- **Pitch Normalization**: We normalize the pitches to transfer the tonality to "C major" or "A minor".

Table 4: Our filtering rules.

Type	Rule	Purpose
Duplication	Remove the duplicated samples that have the same duration and the same numbers of bars, notes, distinct note positions, and instruments.	To remove the duplicated samples.
Musical features	Only keep the samples of time signature 4/4. Only keep the samples that have at least 2 instruments and have the square synthesizer (the melody track). Only keep the samples whose tempo values (the performance speed) are not less than 24 and not larger than 200. Only keep the samples whose pitch values are not less than 21 (A0) and not larger than 108 (C8). Only keep the samples with maximum note duration not longer than 16 beats (4 bar). Remove the samples that contain 4 or more empty bars, or the pitch/duration values are the same for all the notes.	To remove musically complicated or erroneous samples.

We then represent each MIDI file into a sequence of tokens using a REMI-like [20] method. The bar lines are inferred automatically based on the time signature and the note onset positions. The statistics of the number of tokens and the number of bars are shown in Figure 9. The average number

[7] https://musescore.org/



(a) Statistics of number of tokens.

(b) Statistics of number of bars.

Figure 9: Length statistics. The vertical axes represent the ratios of those samples that are in the corresponding ranges.

of tokens is 15,042, the average number of bars is 95, and the average number of tokens per bar is 158. Most samples are longer than 10,000 tokens, making it essential to design Museformer for efficiently modeling the long music sequences.

B.2 Implementation Details

Museformer is equipped with a dynamic sparse attention mechanism, which requires constructing distinct attention layouts for each sample according to the token ranges of bars. Since the dynamic sparse attention layout is not regular as sliding windows [14, 24] or fixed-length blocks [23], it is very challenging to leverage the GPU parallel computation techniques to speed up training and decrease the memory consumption. Note that we cannot pad each bar into a fixed-length sequence because the lengths of bars vary drastically and it would introduce a lot of padding tokens that lead to unacceptable sequence lengths. To achieve an efficient implementation, we utilize a blocksparse method that splits the attention layouts into fixed-size square blocks, and only computes those blocks where there is at least one query-key/value pair expects computation. In our implementation, all the summary tokens of the input bars are put before the music tokens to facilitate the computation, and thus the summarization step and the aggregation step in FC-Attention can be transferred into computing *summary-to-all* (summary tokens attending to summary tokens and music tokens) and *music-to-all* (music tokens attending to summary tokens and music tokens) attention, respectively, and their attention layouts are shown in Figure 10. We take the following 3 steps to compute each of the two attention processes: 1) *attention layout generation* to generate the full attention layout according to the bar ranges; 2) *layout block-sparsification* to transfer the full layout into a blocksprase layout; 3) *blocksparse computation* to compute the blocksparse multiplications and softmax operations.

Attention Layout Generation According to the bar splitting of each sample, we fill "true" on the corresponding areas of a Boolean tensor to construct the attention layout (shown on the left part of Figure 10). To speed up the construction, we collect the begin and end indices of the bars ahead of time, and write a CUDA kernel to fill it for all the bars simultaneously.

Layout Block-Sparsification As visualized in Figure 10, the full attention layouts are block-sparsified with a fixed-size square (the block size set to 32 in our experiments), and diminished into the blocksparse layouts.

Blocksparse Computation We leverage SparTA [40] to compute the blocksparse multiplications and softmax operations of attention. It only computes for the shaded areas on the blocksparse layouts in Figure 10.

Since the attention layouts are the same for all the layers and heads in our setting, we only construct the attention layouts once for each sample, and cache them as well as the SparTA kernels for reuse, which saves a lot of memory and time.

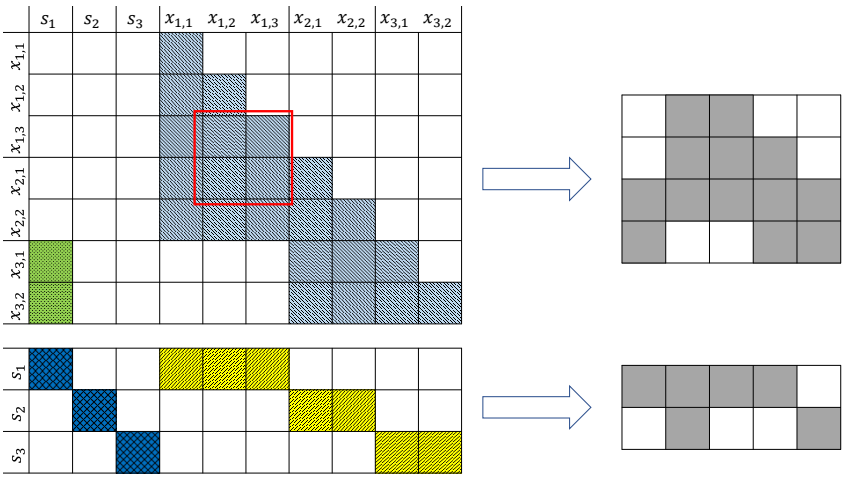


Figure 10: Visualization of the blocksparse implementation. The full attention layouts are on the left, which are exactly the same layouts shown in Figure 2b except that the summary tokens are put before the music tokens to facilitate our implementation. If the block size is 2, the corresponding blocksparse layouts are shown on the right. Note that it is only a toy example, and in real cases, the sequences are much longer and the sparsity of the layouts are much larger than what is depicted here.

B.3 Detailed Model and Training Configurations

For Museformer and all the compared models, the basic model and training configurations introduced in §4.1 are set same. For Museformer that involves two separate attention processes (summarization and aggregation), to make the parameter size comparable with other models, the projection matrices for projecting the targets (i.e., queries) are shared for the two attention computation, and we add different biases for summary tokens and music tokens respectively. It is the same for projecting the sources (i.e., keys and values), except that we use different projection parameters for $\tilde{\boldsymbol{S}}_{N(i)}$ in Equation 3. All the models have comparable amount of trainable parameters: Museformer 16.1M, Music Transformer 16.6M, Transformer-XL 13.9M, Longformer 15.3M, Linear Transformer 13.2M, with acceptable differences due to different architectures and implementations.

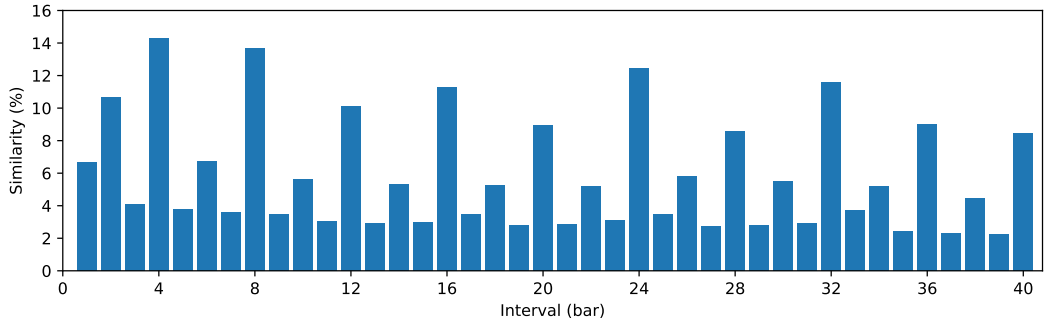
All the models are trained on 4 Nvidia V100 32GB GPUs with fp16. Since in Museformer, a token directly attends to the tokens of 8 previous bars (the selected structure-related bars) and the current bar via the fine-grained attention, for a fair comparison, we set the window sizes for Longformer to $1{,}408$, which is approximately the number of tokens that 9 bars contain. For Music Transformer that uses full attention and cannot process long sequences at once, we chunk the input sequence into blocks of fixed size $1{,}408$, and manipulate the batch size and the update frequency to ensure that the it is updated the comparable number of times within each epoch as other models. For Transformer-XL, the chunk size and the memory size are also set to $1{,}408$. During inference, all the models are applied to generate sequences not shorter than $2{,}048$ and not longer than $20{,}480$, until the end-of-sentence token is generated. The top-k sampling is used, and k is set to 8.

C Similarity Distribution of Generated Music

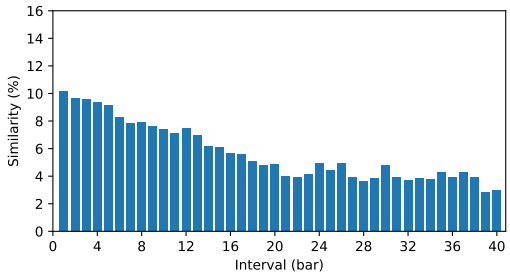
Compared to other models (the compared models and the ablation settings), the music generated by Museformer has the similarity distribution most similar to that of the training data, as its SE reported in Table 1 is the smallest. However, the value of SE may not comprehensively represent the structural characteristics, so we further display and discuss about the specific distributions in this section.

Figure 11 shows the similarity distributions. We can observe that: 1) The distribution of Museformer is very similar to that of the training data (shown in Figure 3). The quantity is close, and the contour shows the same periodical pattern, i.e., the previous 2 bars, the previous 4-th bar as well as its multipliers, have relatively large similarities. 2) The distributions of Music Transformer and Linear Transformer show no periodical pattern. It implies that the Music Transformer model trained on short sequences cannot generate well-structured music of long lengths, and Linear Transformer cannot well

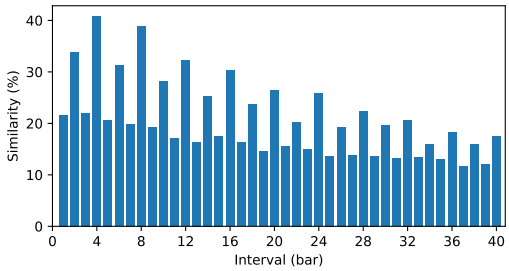
capture the structure-related correlations even though its receptive field covers the whole sequence. 3) The distributions of Transformer-XL and Longformer show the tendency of the periodical pattern, and the similarity decreases as the interval increases in general. It indicates that the two models that directly attend to every token within only the recent range have the ability to generate periodical repetitions of short distances, but fall short for long-term structures. 4) It seems that compared to the quantity of the similarity, the contour (i.e., the periodical pattern) is more relevant to the human scoring on the structure-related metrics. The distributions of Transformer-XL and Longformer show the pattern, and their subjective scores on short-term and long-term structures (shown in Table 2) are relatively high, and are higher than Music Transformer and Linear Transformer whose distributions fail to show the periodical pattern. However, the quantity of the similarity may also influence human decisions. For example, the quantities of the similarities of Transformer-XL is relatively high, which indicates that too many repetitions are generated. For these samples, human scorers sometimes think them annoying and eventually give lower scores on musicality. 5) The ablation setting, Museformer w/o coarse-grained, has slightly larger SE compared to Museformer, and its distribution clearly shows the periodical pattern. Thus, the coarse-grained attention does not contribute a lot on the music structures. The distribution of Museformer w/o bar selection shows the tendency of the periodical pattern, and the similarity decreases in general as the interval increases, similar to those of Transformer-XL and Longformer. It indicates that the structure-related bars are contributory to generating music with both short-term and long-term structures.



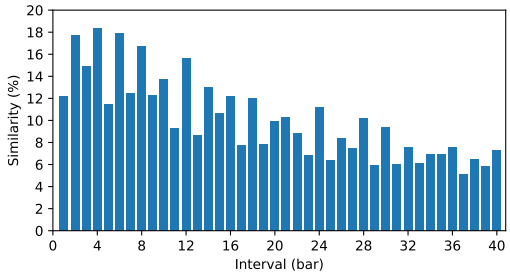
(a) Museformer



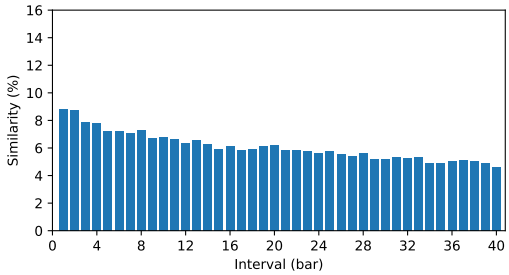
(b) Music Transformer



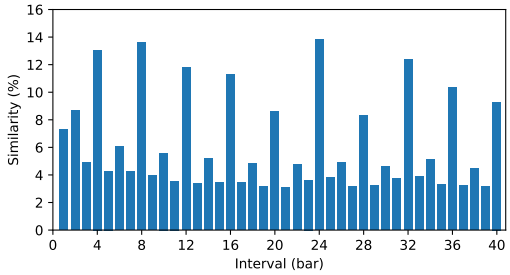
(c) Transformer-XL



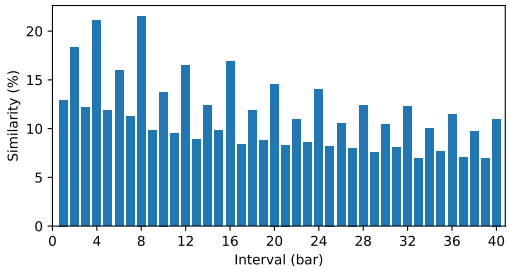
(d) Longformer



(e) Linear Transformer



(f) Museformer w/o coarse-grained



(g) Museformer w/o bar selection

Figure 11: The similarity distribution of the melody track of the music generated by different models.