# Segment-Factorized Full-Song Generation on Symbolic Piano Music

**Ping-Yi Chen**[1] **Chih-Pin Tan**[2] **Yi-Hsuan Yang**[2]
[1] National Cheng Kung University [2]National Taiwan University
{a931eric,tanchihpin0517}@gmail.com, yhyangtw@ntu.edu.tw

## Abstract

We propose the Segmented Full-Song Model (SFS) for symbolic full-song generation. The model accepts a user-provided song structure and an optional short seed segment that anchors the main idea around which the song is developed. By factorizing a song into segments and generating each one through selective attention to related segments, the model achieves higher quality and efficiency compared to prior work. To demonstrate its suitability for human–AI interaction, we further wrap SFS into a web application that enables users to iteratively co-create music on a piano roll with customizable structures and flexible ordering.

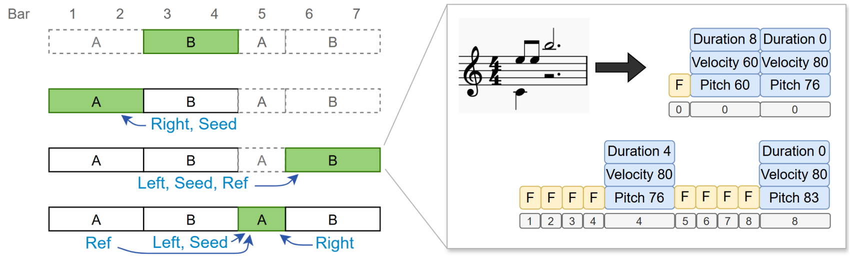
## 1 Introduction

Symbolic music generation has become a prominent research topic in recent years. Previous studies have explored various aspects, including model architectures (Hadjeres et al., 2017; Huang et al., 2018; Min et al., 2023; Yuan et al., 2025), representations (Huang and Yang, 2020; Hsiao et al., 2021), controllability (Wu and Yang, 2023a), arrangement (Zhao and Xia, 2021; Tan et al., 2024a,b), and structural modeling (Wu and Yang, 2023b; Tan et al., 2022; Shih et al., 2023; Wang et al., 2024). Among these tasks, full-song generation remains particularly challenging, as models must not only generate long sequences efficiently but also preserve coherence across the overall song structure.

Consider how human composers typically work. A composer often begins by devising a theme and a high-level song structure, places the theme within the song, and then fills in the remaining sections. This process is only partially autoregressive: when composing a specific section, the composer usually refers to the most relevant context rather than revisiting the entire song, which would be both impractical and inefficient. Wang et al. (2024) proposed a four-stage generation framework, which we refer to as *WholeSong*, adopting a coarse-to-fine hierarchical approach with selective autoregressive conditioning to better model global song structure. However, their approach relies on a diffusion backbone, which requires executing the entire diffusion process for each segment, resulting in an inefficient generation procedure. Moreover, their system encodes all context uniformly into the same conditioning space, without incorporating higher-level concepts such as themes or motifs.

In this paper, we propose **S**egmented **F**ull-**S**ong generation model (SFS), which decomposes a full song into segments using a rule-based segmentation algorithm and employs a customized Transformer to generate each segment autoregressively, conditioned only on structurally relevant preceding segments and context explicitly defined by the given structure. By flexibly selecting relevant segments, our model aligns with the way human composers approach songwriting: first envisioning a theme, then composing segments in a flexible order, which enables a more natural and interactive workflow in application. In the experiments, we show that our model outperforms the approach of WholeSong in terms of structural coherence and motif awareness, as evaluated in a user

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: AI for Music.



(a)

(b)

Figure 1: (a) Generative process with $(\hat{s}_1, \hat{e}_1) = (1, 2)$, $(\hat{s}_2, \hat{e}_2) = (3, 4)$, $(\hat{s}_3, \hat{e}_3) = (5, 5)$, $(\hat{s}_4, \hat{e}_4) = (6, 7)$, $\hat{l}_{1:4} = (A, B, A, B)$, and $o_{1:4} = (2, 1, 4, 3)$. See Section 2.1 for notation. (b) Example of our music language: orange refers to frame tokens, blue refers to note tokens, and gray refers to inferred positions. The `[Duration 0]` token indicates that a note's offset is set by the next onset of the same pitch or the next bar line.

study. We open-source our model implementation and trained weights[1] , as well as an interactive web interface for the model.[2] A demo page is also provided for listening to generations from our model.[3]

# 2 Methodology

## 2.1 Segment-Factorized Full-Song Generation

SFS focuses on transforming a fixed music structure into a complete song while adhering to specified themes. Therefore, we assume that the music structure[4] is provided by the user. Given a song with $N$ bars $\{B_1, B_2, \dots, B_N\}$ and $M$ segments, we represent its structure as $(\hat{s}_{1:M}, \hat{e}_{1:M}, \hat{l}_{1:M})$, where $(\hat{s}_i, \hat{e}_i, \hat{l}_i)$ denote the start bar, end bar, and label of the $i^{\text{th}}$ segment, respectively, indexed from the beginning of the song. Rather than generating music strictly in chronological order, our model is capable of generating segments in an arbitrary order. Let $\{o_1, \dots, o_M\}$ denote the order in which the segments are generated[5]. Using this order, we define the annotations $s_i = \hat{s}_{o_i}$, $e_i = \hat{e}_{o_i}$, and $l_i = \hat{l}_{o_i}$, so that $(s_i, e_i, l_i)$ specifies the $i^{\text{th}}$ segment to be generated. We can then factorize the joint probability of the entire song as:

$$P(B_{1:N} \mid \hat{s}_{1:M}, \hat{e}_{1:M}, \hat{l}_{1:M}) = \prod_{i=1}^{M} P\big(B_{s_i:e_i} \,\big|\, B_{s_1:e_1}, \dots, B_{s_{i-1}:e_{i-1}}, s_{1:M}, e_{1:M}, l_{1:M}\big). \tag{1}$$

For simplicity, we refer to the $i$th segment $B_{s_i:e_i}$ as $\text{Seg}_i$ in the following discussion.

From our observations, musicians tend to focus on the main musical idea while selectively attending to specific contexts to ensure coherence across the song. Inspired by this, we define four types of essential information (referred to as the *context*): Left, Right, Seed, and Ref. Here, Left and Right denote the nearest existing segments to the left and right of the target segment, Seed represents the segment carrying the main idea of the song, and Ref is a reference segment with the same label among the existing segments (Figure 1(a)). The precise definition of the context is provided in Appendix A. Instead of attending to all previously generated tokens, the model attends to these four segments at the token level, while all existing segments are encoded into a compact representation through a global vision module $G$. The approximated joint probability is formulated as:

$$\begin{aligned} &P(B_{1:N} \mid s_{1:M}, e_{1:M}, l_{1:M}) \\ &\qquad \approx \prod_{i=1}^{M} P\Big(\text{Seg}_i \,\Big|\, \text{Left}_i, \text{Right}_i, \text{Seed}_i, \text{Ref}_i, s_i, e_i, G(\text{Seg}_{1:i-1}, s_{1:i-1}, e_{1:i-1})\Big), \end{aligned} \tag{2}$$

[1] `https://github.com/eri24816/segmented-full-song-gen`

[2] `https://github.com/eri24816/co-compose`

[3] `https://sfs-demo.eri24816.tw`

[4] `Music structure` refers to the combination of musical form and lengths of segments.

[5] We train the model on all permutations, enabling adaptation to any user-specified order at inference.

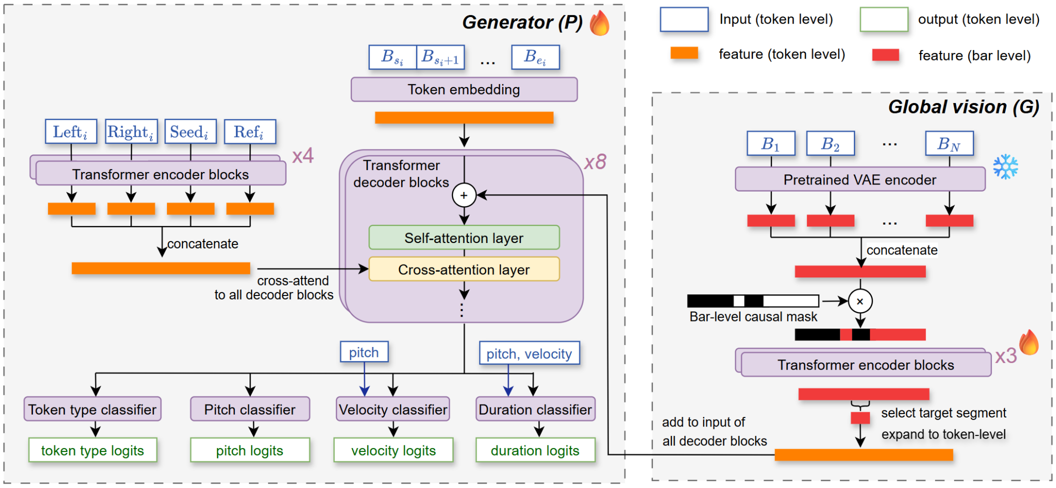


Figure 2: Model architecture of the Segmented Full-Song Model. At the output heads of the Generator (bottom middle), pitch, velocity, and duration are generated sequentially due to their dependencies. During training, the velocity classifier receives the ground-truth pitch, and the duration classifier receives the ground-truth pitch and velocity. During inference, they instead receive sampled values.

where $\mathrm{Left}_i$, $\mathrm{Right}_i$, $\mathrm{Seed}_i$, and $\mathrm{Ref}_i$ are token-level references extracted from previously generated segments $\mathrm{Seg}_{1:i}$ that are most relevant to the current segment according to the given structure. The global vision module $G$ provides a coarse summary of all previously generated segments, giving the model awareness of the song's overall content.

### 2.2 Tokenization

We represent music using a frame-based representation. The note onsets and offsets are first quantized into frames of frames of length $1/8$ beat. For each frame, a *frame token* is added to the sequence, followed by *note tokens* representing the notes that onset at that frame, sorted in ascending pitch order. The sequence then continues with the frame token and note tokens of the next frame. A note token is composed of three sub-tokens: pitch, velocity, and duration (see Figure 1(b)). Positional information is encoded using model-specific embeddings (see Appendix D for details).

### 2.3 Implementation

The model consists of two components: the global vision encoder $G$ and the song generator $P$ (see Figure 2). The global vision encoder $G$ employs a pretrained VAE encoder to embed each bar in $\mathrm{Seg}_{1:i-1}$, converting tokens into bar-level embeddings. The generator $P$ is a Transformer conditioned on two sources: (i) the global vision output, incorporated into the decoder via in-attention (Wu and Yang, 2023a), and (ii) the context $(\mathrm{Left}_i, \mathrm{Right}_i, \mathrm{Seed}_i, \mathrm{Ref}_i)$, provided through the generator's encoder. To represent positional information in both $G$ and $P$, we apply positional encodings specifically designed for our model, with details provided in Appendix D.

## 3 Evaluation and Discussion

We compare our model against two baselines: a *flat* model and WholeSong. The flat model is a GPT-like Transformer trained on 8-bar fragments, which can be regarded as an ablation of our model without structural conditioning. For a fair comparison, we only use the last 3 levels of Wang's model for inference because the form is a given condition. Also, we modify the inference code of WholeSong to support seed conditioning by fixing the given seed segment throughout the diffusion process across all three levels of generation. All models are trained on our in-house dataset, consisting of 32,090 pop piano performances for training and 3,615 for testing.

Table 1: Comparision between models

| | | SI | | | User Study | |
|---|---|---|---|---|---|---|
| **Model** | **Inference Speed** | $\mathbf{SI}_{2-8}$ | $\mathbf{SI}_{8-16}$ | $\mathbf{SI}_{16+}$ | **O** | **A** |
| SFS (Ours) | 2.03 beat/sec. | 0.3286 | **0.2264** | **0.1109** | 3.14 | **3.59** |
| WholeSong | 0.197 beat/sec. | 0.3234 | 0.2262 | 0.0860 | 3.02 | 3.16 |
| Flat | 5.68 beat/sec. | **0.3426** | 0.1990 | 0.0409 | **3.36** | 2.34 |
| Datset | - | 0.4398 | 0.3827 | 0.3300 | 4.00 | 4.07 |

For objective evaluation, we sample 45 songs from the test set. Using their structural specifications, we condition both our model and WholeSong to generate corresponding songs. The flat model instead simply generates unconditional pieces of the same length. We then compute the average Structureness Indicator (SI) (Wu and Yang, 2020) at short (2–8 bars), mid (8–16 bars), and long (16+ bars) ranges, which reflects the ability of a model to maintain structural consistency.

For subjective evaluation, we generate 16 samples with our model and WholeSong, conditioned on seed segments and structures from the test set. The flat model again produces unconditioned sequences. We then conduct a user study in which each participant listens to the seed, followed by generations from the models and the original song. Participants rate each piece on a 1–5 scale along two aspects: **O**verall quality (O) and **A**dherence to seed (A). We collect responses from 44 participants and report the mean scores.

Results of both evaluations are shown in Table 1. Compared to WholeSong, our model achieves stronger adherence to the seed and slightly higher scores in both overall quality and structureness. However, a substantial gap remains relative to real data, highlighting the need for further improvement in full-song generation. We suspect that the higher overall quality reported for the flat model arises from its fluency, which is often preferred by users, as it always generates in a forward direction.

In terms of efficiency, our model achieves real-time generation at an average of 2.03 beat/second , about 10× faster than WholeSong, despite operating at twice the temporal resolution (1/8 beat vs. 1/4 beat). This real-time capability enables streaming output to a user interface during generation, improving user experience in applications such as interactive composition and live performance.

Despite the segment-level correspondence, we notice that the generated music sometimes lack smooth phrase-level transitions and progression across the full song. The segments may appear thematically consistent yet loosely connected, without the natural buildup and flow that typically define sections such as the introduction, verse, chorus, and outro. This suggests the need for a higher-level planning mechanism to guide how each phrase develops within the overall song form in future works.

## 4 Web Interface

We build a web interface for our model to showcase human–AI co-composition.[2] It provides (i) a structure editor for defining song structure and seed, and (ii) a piano-roll editor where users can edit notes or let the model generate selected ranges, enabling iterative, arbitrary-order song completion. The application also serves as a general interface for full-song music generation via an abstract Python API that can connect to different models. Usage instructions are in Appendix F.

## 5 Conclusion

We proposed the Segmented Full-Song Model (SFS) and demonstrated its capability in generating complete songs. Experiments show that SFS achieves stronger adherence to seeds, improved structural consistency, and higher subjective quality compared to prior work, while being an order of magnitude faster, enabling real-time streaming. We also highlighted its potential for human–AI co-creation through a web-based composition interface. We look forward to further explorations of model designs that align more closely with human creative workflows and open new possibilities for human–AI interaction in music-making.

## References

Gaëtan Hadjeres, François Pachet, and Frank Nielsen. Deepbach: a steerable model for bach chorales generation. In *International conference on machine learning*, pages 1362–1371. PMLR, 2017.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer. *arXiv preprint arXiv:1809.04281*, 2018.

Lejun Min, Junyan Jiang, Gus Xia, and Jingwei Zhao. Polyffusion: A diffusion model for polyphonic score generation with internal and external controls. *arXiv preprint arXiv:2307.10304*, 2023.

Shenghua Yuan, Xing Tang, Jiatao Chen, Tianming Xie, Jing Wang, and Bing Shi. Diffusion-based symbolic music generation with structured state space models. *arXiv preprint arXiv:2507.20128*, 2025.

Yu-Siang Huang and Yi-Hsuan Yang. Pop music transformer: Beat-based modeling and generation of expressive pop piano compositions. In *Proceedings of the 28th ACM international conference on multimedia*, pages 1180–1188, 2020.

Wen-Yi Hsiao, Jen-Yu Liu, Yin-Cheng Yeh, and Yi-Hsuan Yang. Compound word transformer: Learning to compose full-song music over dynamic directed hypergraphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(1):178–186, May 2021. doi: 10.1609/aaai.v35i1.16091. URL `https://ojs.aaai.org/index.php/AAAI/article/view/16091`.

Shih-Lun Wu and Yi-Hsuan Yang. Musemorphose: Full-song and fine-grained piano music style transfer with one transformer vae. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 31:1953–1967, 2023a.

Jingwei Zhao and Gus Xia. Accomontage: Accompaniment arrangement via phrase selection and style transfer. *arXiv preprint arXiv:2108.11213*, 2021.

Chih-Pin Tan, Shuen-Huei Guan, and Yi-Hsuan Yang. Picogen: generate piano covers with a two-stage approach. In *Proceedings of the 2024 International Conference on Multimedia Retrieval*, pages 1180–1184, 2024a.

Chih-Pin Tan, Hsin Ai, Yi-Hsin Chang, Shuen-Huei Guan, and Yi-Hsuan Yang. Picogen2: Piano cover generation with transfer learning approach and weakly aligned data. *arXiv preprint arXiv:2408.01551*, 2024b.

Shih-Lun Wu and Yi-Hsuan Yang. Compose & embellish: Well-structured piano performance generation via a two-stage approach. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2023b.

Chih-Pin Tan, Alvin WY Su, and Yi-Hsuan Yang. Melody infilling with user-provided structural context. *arXiv preprint arXiv:2210.02829*, 2022.

Yi-Jen Shih, Shih-Lun Wu, Frank Zalkow, Meinard Müller, and Yi-Hsuan Yang. Theme transformer: Symbolic music generation with theme-conditioned transformer. *Trans. Multi.*, 25:3495–3508, January 2023. ISSN 1520-9210. doi: 10.1109/TMM.2022.3161851. URL `https://doi.org/10.1109/TMM.2022.3161851`.

Ziyu Wang, Lejun Min, and Gus Xia. Whole-song hierarchical generation of symbolic music using cascaded diffusion models. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=sn7CYWyavh`.

Shih-Lun Wu and Yi-Hsuan Yang. The jazz transformer on the front line: Exploring the shortcomings of ai-composed music through quantitative measures. In *Proceedings of the 21th International Society for Music Information Retrieval Conference (ISMIR)*, October 2020.

# A Definition of the Context

To provide contextual information for generating a target segment $\mathrm{Seg}_i$, the context is selected from segments that have been generated, $\mathrm{Seg}_{1:i-1}$, with the following rule:

- $\mathrm{Left}_i$: the nearest existing segment to the left of $\mathrm{Seg}_i$, helps generate a smooth transition from that segment.
$$\mathrm{Left}_i = \mathrm{Seg}_n \quad \text{where} \quad n = \arg\max_{j<i}\{e_j \mid e_j \le s_i\}$$
(or $\varnothing$ if no such $j$ exists)
- $\mathrm{Right}_i$: the nearest existing segment to the right of $\mathrm{Seg}_i$, helps generate a smooth transition to that segment.
$$\mathrm{Right}_i = \mathrm{Seg}_n \quad \text{where} \quad n = \arg\min_{j<i}\{s_j \mid s_j \ge e_i\}$$
(or $\varnothing$ if no such $j$ exists)
- $\mathrm{Seed}_i$: The seed segment carries the main idea of the song (motifs and overall style) to all other segments. It is defined as the first segment of the song, based on the assumption that the main idea is the first thing to be written down when composing music.
$$\mathrm{Seed}_i = \begin{cases} \varnothing & \text{if } i = 1 \\ \mathrm{Seg}_1 & \text{if } i > 1 \end{cases}$$
- $\mathrm{Ref}_i$: A reference segment with the same label occurring earlier. The earliest segment is chosen if multiple segments are eligible:
$$\mathrm{Ref}_i = \mathrm{Seg}_n \quad \text{where } n = \min_j\{j \mid l_j = l_i, j < i\}$$
(or $\varnothing$ if no such $j$ exists)

In practice, to limit computational cost, each of the four context types is truncated to a maximum length of 8 bars. For Left, we retain the right-most 8 bars, while for Right, Seed, and Ref, we retain the left-most 8 bars.

# B Experiment Setting

The dataset consists of performances of piano covers of pop music from YouTube, ranging from 4 to 200 bars. On average, each song contains 4,394 tokens, or 54.6 tokens per bar. It includes 32,090 songs for training and 3,615 songs for testing. All songs are transported to C major or A minor key. The structural labels of the pieces are assigned automatically using our segmentation algorithm described in Appendix C.

The procedure to construct one training sample is:

1. Sample a song in the dataset.
2. Obtain its structure $(\hat{s}_{1:M}, \hat{e}_{1:M}, \hat{l}_{1:M})$ using the segmentation algorithm.
3. Identify the label that appears on the most bars, in which we assume the theme is located. Among the segments with that label, select the one that starts closest to the song's middle, and assign it to $o_1$.
4. Assign a segment to $o_2$ with similar way we assign $o_1$, but select the second-most frequent label.
5. Fill $o_{3:M}$ with the remaining segments in a random order to construct an arbitrary sequence $o$ as specified by the user.
6. Randomly pick one segment to be the target segment. If its length is greater than 8 bars, randomly sample an 8-bar fragment inside it, so it can fit our model's receptive field.
7. Obtain the context segments with the definition described in Appendix A.

We train the model for 2 million steps over 127 hours on a single RTX 4090 GPU. The batch size is 12. Optimization is performed with Adam, minimizing negative log-likelihood (NLL) *summed* over tokens rather than averaged. The learning rate decays exponentially from $1 \times 10^{-4}$ to $5 \times 10^{-6}$ through the training.

# C Details about the segmentation algorithm

## C.1 Similarity Metric

The segmentation algorithm is based on our similarity metric between two bars of music $s(\cdot, \cdot)$. It is constructed as follows.

First, given two bars of music $a$ and $b$, each considered as a set of notes, we define the *note overlap score* $f(a, b)$ as the maximum number of one-to-one matches between notes in $a$ and $b$, divided by $\max(|a|, |b|)$. A valid match requires that the notes share the same pitch and that their onset times differ by at most one frame (i.e., $1/8$ beat).

To increase the weight of the skyline in the note overlap score, we define the skyline of a bar

$$\text{SKY}(a) = \left\{ n \in a \,\middle|\, \nexists m \in (a - \{n\}) \text{ such that } \frac{m.\text{pitch} - n.\text{pitch}}{|m.\text{onset} - n.\text{onset}|} > \frac{\text{1 octave}}{\text{1 beat}} \right\}$$

and revise the overlap score to jointly consider all notes and skylines:

$$\tilde{f}(a, b) = 0.5\, f(a, b) + 0.5\, f(\text{SKY}(a), \text{SKY}(b)).$$

Finally, the similarity between two bars is defined as

$$s(a, b) = \max\bigl(\tilde{f}(a, b),\ \tilde{f}(a, b^{8va}),\ \tilde{f}(a, b^{8vb})\bigr),$$

where $b^{8va}$ and $b^{8vb}$ denote transpositions of $b$ one octave up and down, respectively.

## C.2 Segmentation Algorithm

Given a song divided into $N$ bars $B_1, \ldots, B_N$, the segmentation proceeds as follows:

1. **Similarity matrix.** Compute the bar-wise similarity matrix

   $$\mathbf{S}_{i,j} = s(B_i, B_j),$$

   where $s(\cdot, \cdot)$ is the similarity between two bars.
2. **Adjacency regularization.** This step encourages consecutive label assignment. First, construct a banded adjacency matrix $\mathbf{A}$ with ones on the main diagonal and the first diagonals above and below it. Then form the adjusted similarity matrix

   $$\mathbf{M} = (1 - \alpha) \max(0.3, \mathbf{S}) + \alpha \mathbf{A}.$$

3. **Spectral embedding.** Compute the unnormalized graph Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{M}$, where $\mathbf{D}$ is the diagonal degree matrix. Let $\lambda_1 \leq \cdots \leq \lambda_N$ be the eigenvalues of $\mathbf{L}$, and choose

   $$k = \arg \max_{2 \leq j \leq K_{\max}} (\lambda_j - \lambda_{j-1}),$$

   i.e., the index of the largest eigen-gap up to $K_{\max}$.
4. **Clustering.** Take the first $k$ eigenvectors of $\mathbf{L}$, normalize them, and run $k$-means clustering to obtain bar labels $\ell_1, \ldots, \ell_N$.
5. **Segmentation.** Identify split points at positions where $\ell_i \neq \ell_{i-1}$. Using these split points, the start bar, end bar, and label of each segment in the song are determined.

In practice, we adopt

$$\alpha = 0.7, \quad K_{\max} = \min\Bigl(6, \left\lfloor \tfrac{N}{8} \right\rfloor\Bigr),$$

which we find to work well on our dataset.

We spot two limitations of the algorithm. First, it does not provide semantic labels (verse, chorus, etc.). Also, it can't identify consecutive repeated segments. Instead of reporting two segments with the same label, it reports one big segment.

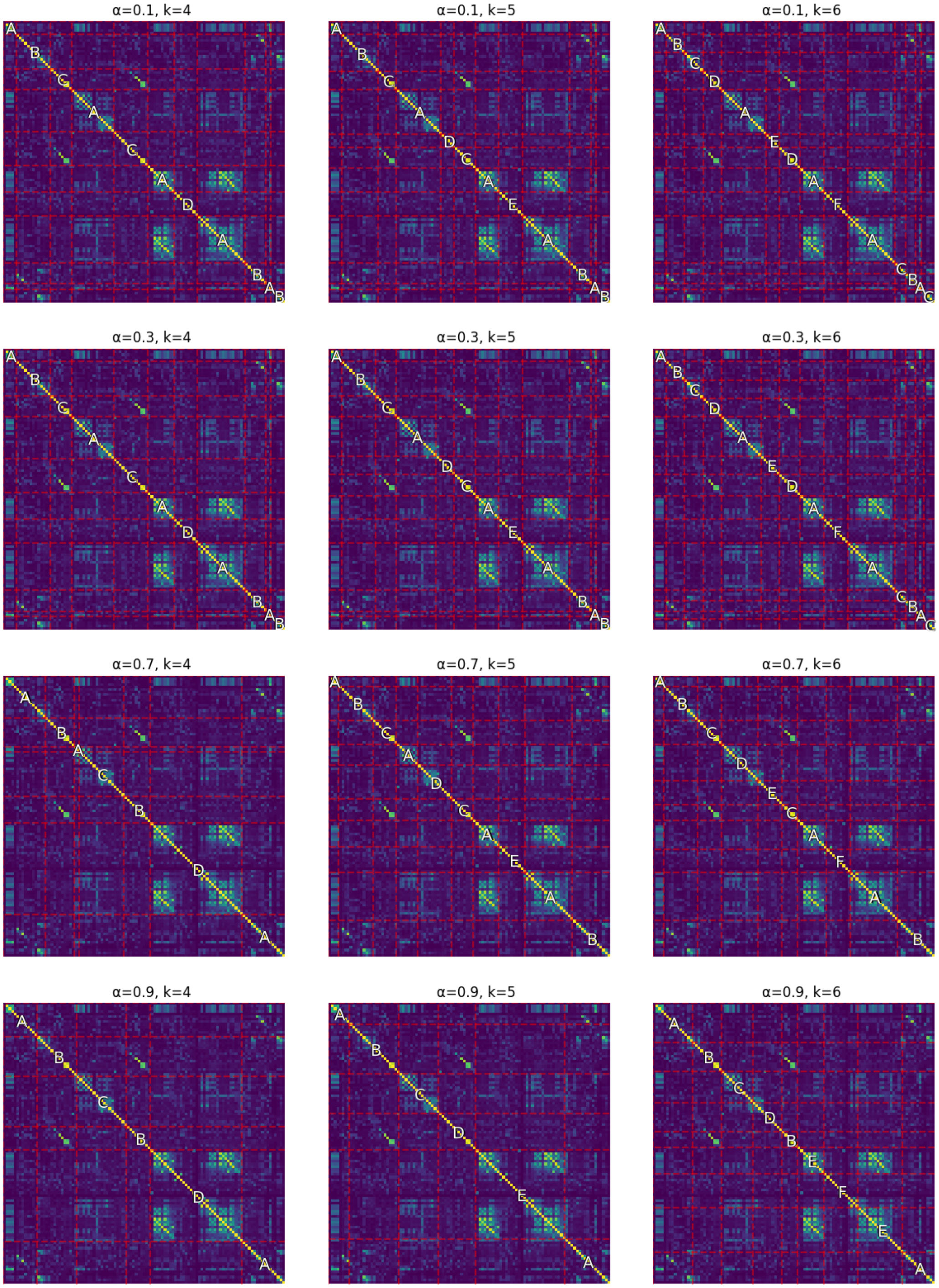


Figure 3: The segmentation result of song *Hikaru Nara - Your Lie in April OP1 [Piano]*(`https://www.youtube.com/watch?v=zsVAbS8xmaU`) using our algorithm with different settings of $\alpha$ and $k$. The setting we actually use for this song $\alpha = 0.7$ and $k = 6$. The heatmap shows the similarity matrix of the song, where purple to yellow indicates 0 to 1.

# D Positional Encodings Used in Our Model

The positional information is not directly embedded into frame tokens but instead provided through positional embeddings. When predicting a token, the position assigned to the query of each attention block always corresponds to the token's position in music, measured in frames.

Three types of positional encodings are used in our model: Start-End positional encoding (Start-End PE), Sub-beat positional encoding (Sub-beat PE), and RoPE. We use a concatenation of Start-End PE relative to song, Start-End PE relative to segment, and Sub-beat PE and add it to every token-level input to provide complete information about the local and the global position. RoPE is used in every self-attention and cross-attention layer.

Start–End positional encoding provides the model with information about a token's relative position within a segment or song. It is constructed by concatenating two standard sinusoidal encodings: one representing the distance from the start and the other representing the distance from the end.

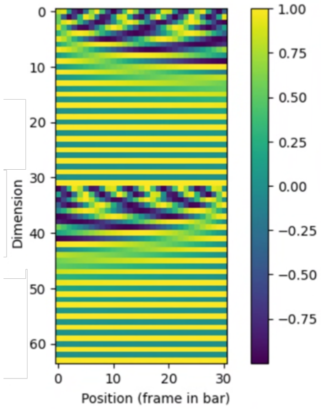


Figure 4: Start–End positional encoding

Sub-beat positional encoding indicates the frame index inside a bar. It is a concatenation of a one-hot vector and a binary vector.

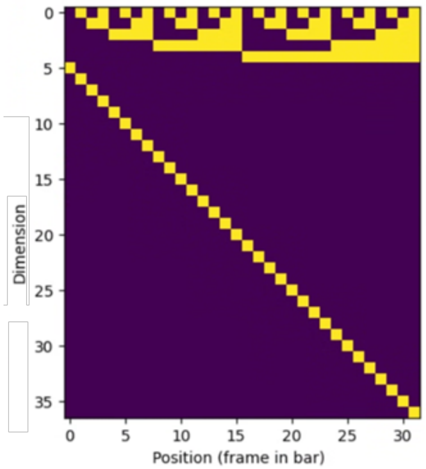


Figure 5: Sub-beat positional encoding. Purple indicates 0, and yellow indicates 1.

Notably, all positional embedding in our model uses time frames (instead of tokens' positions in sequence) as position indices, which provides a more natural representation.

## E Details about the User Study

To construct the listening materials for the user study, we sample 16 songs from the test set whose seed segments consist of 8 bars. For each song, we use the seed and structure to condition our model and WholeSong. For the flat model, we generate songs of the same length without additional conditioning.

In the user study, each participant evaluates four instances of musical pieces associated with the same seed and structure in a shuffled order. These four instances are: generation from our model, generation from WholeSong, generation from the flat model, and the real sample from the dataset. In the study, we refer to these as *generations*.

For each generation, the participant first listens to the seed, then to the generation, provided with the following description:

- **Seed:** a given short musical idea.
- **Generation:** a complete piece of music generated based on the seed.

Participants then rate each generation on a 5-point Likert scale (1 = lowest, 5 = highest) according to the following criteria:

1. **Adherence to Seed:**
   How much does the generated piece retain the seed's musical idea? How similar is the overall style or mood of the generated piece compared to the seed?
2. **Structureness:**
   How good is the generated piece's structure as a complete composition? Consider, for example, does the piece have a clear form (e.g. intro, verse, chorus, outro etc.) and have a reasonable emotional development?
3. **Overall Quality:**
   How good is the overall quality of the generated piece? How good does the generated piece sound to you?

We collect 44 responses from 21 amateur participants (with no or fewer than 3 years of music-related training), 19 experienced participants (3 or more years of training), and 4 professionals, and we report the mean scores.

## F Instructions for Using Our Web Interface

The interface consists of the following components:

1. **Structure editor.** Lets users define the song structure, which serves as a condition for generation. The structure can be specified beforehand or adjusted during composition. Drag the left or right edge of a segment to resize it, or click a segment to assign its label.
2. **Piano roll.** The workspace where users and AI collaborate on music. Users can use the space bar to toggle play, click on empty space to create a note, drag to move a note, and right-click to delete a note. Use the scroll wheel to pan, and hold `Control` while scrolling to zoom.
3. **Assets.** Provides several 8-bar MIDI assets that users can drag into the piano roll as starting material or to combine with an existing composition. Users can also import additional assets from their own disk. Click on an asset to preview it.
4. **Bar selection.** Enables quick selection of one or multiple bars, which can then be used with the command palette.
5. **Command palette.** Click *Generate!* to let the AI generate content for the selected range, or use other buttons to perform different operations.

To play with it, here's a workflow we recommend:

1. First, specify the desired structure of the song. Using the default structure is also acceptable.

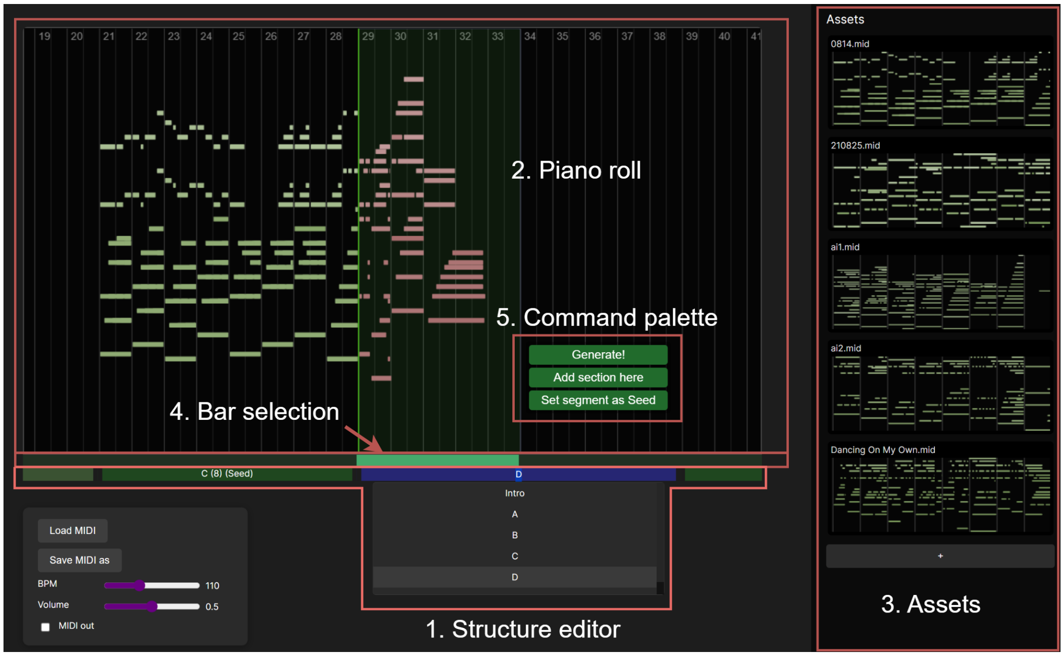


Figure 6: User interface of our web Interface.

2. Next, set the seed segment. You can do this by dragging in an asset, composing it yourself in the built-in or an external editor, or letting the AI generate it. Because the application is not yet ideal for detailed note editing, we suggest composing in another program and importing the MIDI as an asset.
3. Finally, generate the remaining segments in any order, one or several bars at a time. Incorporate human composition when you have a clear idea or wish to refine the AI's output. As our model does not allow fine-grained control, generating an entire segment at once may yield overly random results. In such cases, shorten the generation range and apply rejection sampling guided by human evaluation.

Note that users can listen to the AI's generation in real time by playing the music immediately after clicking the *Generate!* button, as the output streams to the client. For this feature to work reliably, the BPM should be kept below about 120 (or occasionally 100), based on our tests on an RTX 4090. This limit may vary depending on hardware and runtime conditions.