SDMUSE: STOCHASTIC DIFFERENTIAL MUSIC EDITING AND GENERATION VIA HYBRID REPRESENTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

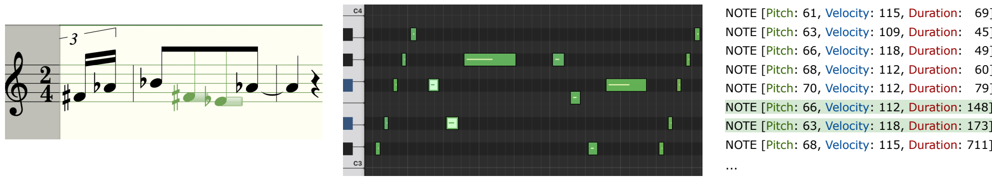
While deep generative models have empowered music generation, it remains a challenging and under-explored problem to edit an existing musical piece at fine granularity. In this paper, we propose SDMuse, a unified Stochastic Differential Music editing and generation framework, which can not only compose a whole musical piece from scratch, but also modify existing musical pieces in many ways, such as combination, continuation, inpainting, and style transferring. The proposed SDMuse follows a two-stage pipeline to achieve music generation and editing on top of a hybrid representation including pianoroll and MIDI-event. In particular, SDMuse first generates/edits pianoroll by iteratively denoising through a stochastic differential equation (SDE) based on a diffusion model generative prior, and then refines the generated pianoroll and predicts MIDI-event tokens auto-regressively. We evaluate the generated music of our method on *ailabs1k7* pop music dataset in terms of quality and controllability on various music editing and generation tasks. Experimental results demonstrate the effectiveness of our proposed stochastic differential music editing and generation process, as well as the hybrid representations.

1 INTRODUCTION

With the development of deep learning and generative models, automatic music composition has received much research attention (Dong et al., 2018; Huang & Yang, 2020; Ren et al., 2020; Ju et al., 2021; Wu & Yang, 2021; Zhang et al., 2022), and also has a lot of successful applications, such as movie soundtracks, virtual singers, and auxiliary composing. However, as the aesthetics of music are diverse for different groups of people, or even for each individual, there is no musical piece in the world that simultaneously satisfies everyone's taste for any scenario. In many cases, one may feel unsatisfied with certain bars of a musical piece, if not with the whole piece, and he or she would like to edit them, which is impossible except for a professional such as a music composer. AI solutions like symbolic music generation methods (Huang & Yang, 2020; Hsiao et al., 2021) can help, but they would regenerate a completely new piece that may be totally different from the former one without preserving those "satisfying" parts. The same thing happens when someone wants to extend an existing musical piece, modify the style or combine several pieces. Therefore, besides the effort on improving generative quality (Ren et al., 2020; Ju et al., 2021) in a broad sense, it is also crucial and much demanded to study how to edit and modify existing musical pieces.

There are only a few works studying music editing tasks, mainly focusing on global music style transfer (Cífka et al., 2020; Wu & Yang, 2021) or music genre transfer (Brunner et al., 2018). The most related work (Wu & Yang, 2021) to ours achieves music style transfer with Transformer VAE on pre-defined musical attributes like rhythmic intensity and polyphony. The musical attributes are song-level, prohibiting users from editing certain part of a musical piece. In addition, users can only control and edit the pre-defined attributes, which severely limits its application. Other works (Cífka et al., 2020; Brunner et al., 2018) have similar limitations. So far, no one has explored editing musical pieces at fine granularity in the same flexible way as image editing, such as inpainting (Pathak et al., 2016) and outpainting (Wang et al., 2014).

Existing state-of-the-art music generation works (Hsiao et al., 2021; Ren et al., 2020; Zhang et al., 2022) with high generative quality are based on MIDI-event representation (an illustration of MIDI-event is in Figure 1c). As validated by the success of these methods, the MIDI-event representation is



(a) Music sheet. (b) Pianoroll. (c) MIDI-event tokens.

Figure 1: The illustrations of different symbolic music representations.

appropriate for the model to generate and model music performance details such as velocity, but not for humans to perceive and edit since the data format is not intuitive enough. Another widely used music representation pianoroll (an illustration of pianoroll is in Figure 1b) is closer to the way of human perception, but the methods (Dong et al., 2018; Yang et al., 2017) based on it cannot achieve comparable generation quality with MIDI-event-based ones. In this work, we consider using both of the representations, which we call hybrid representation, to achieve fine-grained music editing.

Specifically, we define a series of fine-grained music editing tasks and propose a unified Stochastic Differential Music editing and generation framework via hybrid representation, named SDMuse. SDMuse can not only generate a whole musical piece from scratch either unconditionally or conditioned on some control signals (such as chord progression), but also edit existing musical pieces in different ways, including stroke-based generation/editing, inpainting, outpainting, combination and style transferring. As pianoroll is easier for humans to understand and edit, and MIDI-event sequence is more suitable for generation, we design a two-stage pipeline based on a hybrid representation. In the first stage, a diffusion model generative prior is applied and the pianorolls are obtained by iteratively denoising through a stochastic differential equation (SDE). The progressive generation feature of our diffusion model allows us to implement a series of music editing operations in this stage. In the second stage, the generated pianoroll will be refined with more precise music performance details (velocity, fine-grained onset position, etc.) by generating the final MIDI-event sequence in an auto-regressive manner, enjoying the benefits of MIDI-event representation for high-quality music generation.

We evaluate SDMuse on *ailabs1k7* pop music dataset (Hsiao et al., 2021) in terms of *quality* and *controllability* in various music editing and generation tasks. Both objective and subjective results demonstrate the effectiveness of the stochastic differential process and hybrid representation in SDMuse. To our best knowledge, we are the first one to formulate and address fine-grained music editing, which aims to edit existing musical pieces at fine granularity according to diverse demands and provide humans with new ways to collaborate with models during music composition. The generated samples can be found on our demo page `https://SDMuse.github.io/posts/sdmuse/`.

2 Background

2.1 Symbolic music representation

Most previous symbolic music generation works are based on two most common music representations: pianoroll and MIDI-event. Pianoroll-based approaches (Yang et al., 2017; Dong et al., 2018) use pianorolls to represent music scores, with the horizontal axis representing time and the vertical axis representing pitch. Pianoroll is just like an image, so pianoroll-based methods use image-based operations to generate music. Pianoroll is closer to the way of humans perceiving musical pieces, making it more suitable for being understood and edited by humans. However, most state-of-the-art music generation works are MIDI-event-based approaches (Huang & Yang, 2020; Hsiao et al., 2021). They convert a musical piece to a MIDI-event token sequence, and use methods from natural language processing to deal with the token sequence. MIDI-event-based methods can better learn the temporal dependency between different musical events, so as to show more robust generation performance. We list the advantages and disadvantages of these two representations as shown in Table 1 and explain them in detail in Appendix A. To summarize, pianoroll is more appropriate for extracting and controlling perceptive information like structure, while the MIDI-event sequence is more ideal for generating and modeling precise music performance details, such as velocity and

Table 1: Differences between MIDI-event and pianoroll.

	MIDI-event	Pianoroll
Pros & Cons	✓ precise details (velocity, precise position) ✓ regard a note as the unit ✓ robust to generate music pieces ✗ need more data to learn the embedding	✓ prior music information ✓ reflect music structure directly ✓ easy to control and adjust ✗ treat onset and other pos samely
Suitable Usage Scenarios	generating & modeling	perceiving & editing

fine-grained onset position. In this work, we propose SDMuse with a two-stage pipeline, including pianoroll and MIDI-event generation stages that apply hybrid representation to take advantage of these two symbolic music representations.

2.2 Symbolic music editing

Though tremendous progress is made in symbolic music generation and other automatic music composition tasks, there are only a handful of works regarding symbolic music editing tasks. The existing music editing works mainly focus on global music style transfer (Cífka et al., 2020; Wu & Yang, 2021) and music genre transfer (Brunner et al., 2018). Cífka et al. (2020) proposed a system for music accompaniment style transfer, generating accompaniment with the content from content input and the style from style input. Wu & Yang (2021) changed the style of a musical piece by given song-level musical attributes (e.g. rhythmic intensity and polyphony). Brunner et al. (2018) applied GAN-based methods from the field of computer vision to transfer a musical piece from source genre to target genre. However, these works can only edit the music at the song level with pre-defined attributes or labels, limiting the ways of interaction. This work aims to implement fine-grained music editing and achieve flexible collaboration between humans and models during music composition.

2.3 Stochastic differential equations (SDE) for editing

To recover the data from noise, Song et al. (2020) proposed a stochastic differential equation (SDE) to smoothly transform a complex data distribution to a known prior distribution by slowly injecting noise. Similar to diffusion probabilistic models (Ho et al., 2020; Liu et al., 2021; Mittal et al., 2021), SDE-based generative models can be used to convert an initial Gaussian noise vector to a data point in real-world data distribution. As described in Song et al. (2020), we denote $\mathbf{x}(t) \in \mathbb{R}^d$, where $t \in [0, T]$ represents time. Suppose that $\mathbf{x}(0) \sim p_{data}$ is a sample from data distribution, and $\mathbf{x}(T) \sim p_T$ is from prior distribution. The forward SDE process can be formulated as:

$$\mathrm{d}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}, \tag{1}$$

where $\mathbf{w}$ is a standard Brownian motion, and $\mathbf{f}(\cdot, t)$, $g(\cdot)$ are the drift coefficient and the diffusion coefficient of $\mathbf{x}(t)$ respectively. And the reverse SDE (Anderson, 1982) is:

$$\mathrm{d}\mathbf{x}(t) = [\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \mathrm{log} p_t(\mathbf{x})]\mathrm{d}t + g(t)\mathrm{d}\overline{\mathbf{w}}, \tag{2}$$

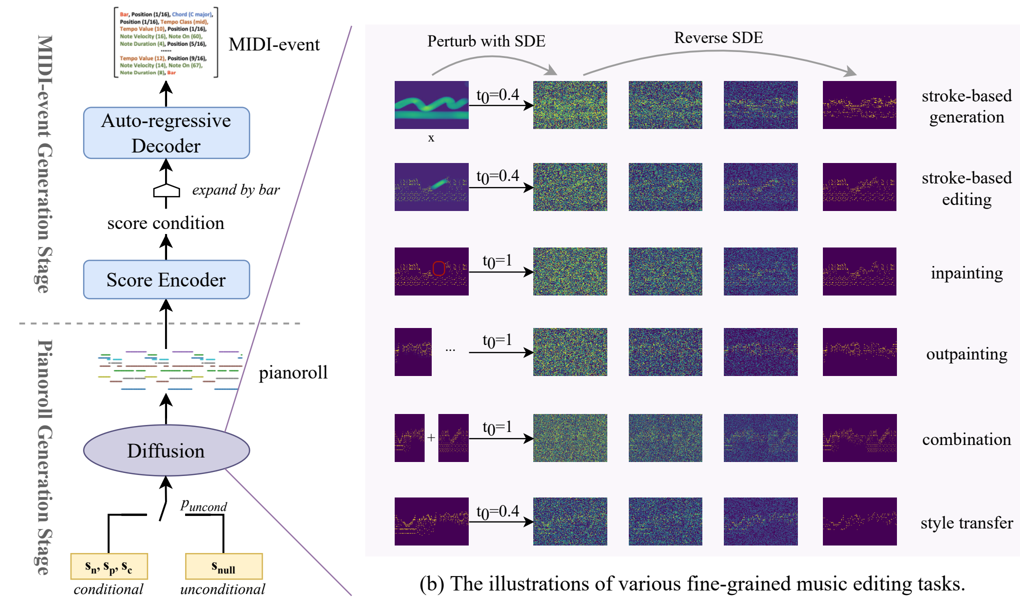
where $\overline{\mathbf{w}}$ is a standard Wiener process and $\nabla_{\mathbf{x}} \mathrm{log} p_t(\mathbf{x})$ is the noise-perturbed score function. There are two different SDEs according to different noise perturbations: Variance Exploding SDE (VE-SDE) and Variance Preserving SDE (VP-SDE). In this paper, we use VP-SDE for conducting experiments to verify our framework. The forward process of VP-SDE is:

$$\mathrm{d}\mathbf{x}(t) = -\frac{1}{2}\beta(t)\mathbf{x}(t)\mathrm{d}t + \sqrt{\beta(t)}\mathrm{d}\mathbf{w}(t), \tag{3}$$

where $\beta(t)$ is a positive noise function. Denote the learned score model as $\boldsymbol{s_\theta}(\mathbf{x}(t), t))$. The reverse VP-SDE process can be solved by following the iteration rule:

$$\mathbf{x}_{n-1} = \frac{1}{\sqrt{1 - \beta(t_n)\Delta t}}(\mathbf{x}_n + \beta(t_n)\Delta t \boldsymbol{s_\theta}(\mathbf{x}(t_n), t_n)) + \sqrt{\beta(t_n)\Delta t}\, \mathbf{z_n}, \tag{4}$$

where $\mathbf{x}_N, \mathbf{z}_n \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, $n = N, N-1, \cdots, 1$, and Δt is time interval between $\mathbf{x}_n$ and $\mathbf{x}_{n-1}$.



(a) SDMuse pipeline. (b) The illustrations of various fine-grained music editing tasks.

Figure 2: Left part is the SDMuse pipeline containing two generation stages. Right part shows the process of various fine-grained music editing tasks. Best viewed in color mode with zoom-in.

In order to synthesize and edit images, Meng et al. (2021) "hijacked" the generative process of SDE-based generative models. Specifically, they added noise to smooth the given stroke paintings or images with stroke-edit to smooth out distortions but preserving the overall image structure. Then they used the noisy input to initialize the SDE and progressively remove the noise. Inspired by Meng et al. (2021), we build our SDMuse to edit and generate musical pieces through VP-SDE.

3 Methods

In this section, we first give a pipeline overview of SDMuse and then describe the design details of the pianoroll and MIDI-event generation stages respectively. Finally, we introduce the formulation and process of various fine-grained music editing tasks.

3.1 Pipeline overview

The overall pipeline of SDMuse is shown in Figure 2a, which consists of two consecutive stages: 1) pianoroll generation stage, which is based on a conditioned diffusion model generative prior and synthesizes pianorolls from scratch or edits existing pianorolls by iteratively denoising through SDE (Song et al., 2020); 2) MIDI-event generation stage, which generates MIDI-event sequences from the output pianorolls of the first stage by refining them with precise music performance details auto-regressively. These two stages can be trained separately and all condition signals in the first stage are at fine granularity, which can be extracted from the musical piece itself without extra data annotation process. The diffusion probabilistic model in the pianoroll generation stage enables SDMuse not only to compose the whole musical pieces from scratch unconditionally or conditioned on given control signals (e.g. note density), but also to modify existing musical pieces in different ways. The details of these two stages are introduced in the following subsections and some details of the model architectures are described in Appendix B.1.

3.2 Pianoroll generation stage

3.2.1 Training of diffusion probabilistic model

As shown in Figure 2, we involve several fine-grained control signals: note density $\boldsymbol{c}_n$, pitch distribution $\boldsymbol{c}_p$, and chord progression sequence $\boldsymbol{c}_c$ during the training process of the diffusion model to enable unconditional and conditional music generation/editing at the same time. These control signals can be extracted from the musical piece itself, and the way of extraction is listed in Appendix B.2. Given these control signals $\boldsymbol{c}_n$, $\boldsymbol{c}_p$, and $\boldsymbol{c}_c$, we can train a conditional diffusion model with the pairs of pianorolls and the corresponding control signals. In order to integrate unconditional and conditional music generation into the same model without an extra training process, we introduce a combined training strategy which can switch freely between two generation settings inspired by Nichol et al. (2021). Specifically, as illustrated in Figure 2a, we use the conditional training paradigm, but assign all control signals to a specific out-domain value ($\boldsymbol{c}_{null}$) with a certain probability p_{uncond} to train the model for unconditional music generation scenario. As mentioned in Section 2.1, one of the drawbacks of pianoroll representation is indiscriminate treatment of note onsets and other positions, which does not match realistic scenarios and affects the robustness of generation. To tackle this problem, we convert the pianoroll to onsetroll by only keeping onset information as described in Appendix B.3 for the training process of the diffusion model.

3.2.2 Generation from scratch

Starting from samples of $\mathbf{x}(T) \sim p_T$, where p_T is the prior distribution, we can generate musical pieces of $\mathbf{x}(0) \sim p_{data}$ unconditionally or conditioned by given control signals. That is to say, we are interested in the $p(\mathbf{x}|\boldsymbol{c})$, where

$$\boldsymbol{c} = \begin{cases} \boldsymbol{c}_{null}, & \text{unconditional generation,} \\ \{\boldsymbol{c}_n, \boldsymbol{c}_p, \boldsymbol{c}_c\}, & \text{conditional generation.} \end{cases} \tag{5}$$

Derived from the reverse-time SDE in Equation 2, the conditional reverse-time SDE can be described:

$$\mathrm{d}\mathbf{x}(t) = [\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x}|\boldsymbol{c})]\mathrm{d}t + g(t)\mathrm{d}\overline{\mathbf{w}}, \tag{6}$$

where $\overline{\mathbf{w}}$ is a standard Wiener process, $\nabla_{\mathbf{x}} \log p_t(\mathbf{x}|\boldsymbol{c})$ is the conditioned noise-perturbed score function, $\mathbf{f}(\cdot, t)$ and $g(\cdot)$ are the drift coefficient and diffusion coefficient of $\mathbf{x}(t)$ respectively. Thus, by given different $\boldsymbol{c}$ for the reverse SDE process, the pianoroll generation stage in SDMuse can achieve unconditional music generation and conditional music generation respectively.

3.2.3 Fine-grained editing

Similar to Meng et al. (2021), the diffusion probabilistic model can serve various fine-grained music editing tasks. We formulate and introduce the following fine-grained music editing tasks, which are illustrated in Figure 2b from top to bottom respectively. All of the following tasks can be implemented with the same algorithm framework (see Algorithm 1) with different process operations to obtain the input pianoroll $\mathbf{x}$ and mask $\boldsymbol{\Omega}$ and different reverse steps t_0. The mask $\boldsymbol{\Omega}$ demonstrates the regions that need to be reserved all the time by setting the value to 0 and the regions that need to be replaced by setting the value to 1. Besides the style transfer which requires control signals, other fine-grained music editing tasks can be conducted both in unconditional and conditional settings.

Stroke-based generation. Like stroke-based image generation mentioned in Meng et al. (2021), stroke-based generation aims to generate the realistic pianoroll with the given stroke pianoroll $\mathbf{x}$. The generated pianorolls are expected to balance between faithfulness and realism, which means that they should not only be similar to the given stroke pianoroll but also be authentic and reasonable. One can draw a stroke pianoroll with a specific structure, thus enabling structure-conditioned music generation. The mask $\boldsymbol{\Omega}$ is set to 1 for all regions and the reverse steps t_0 is set to 0.4.

Stroke-based editing. When someone is not satisfied with an existing pianoroll and wants to edit a certain part of it, the stroke-based editing can help. Given a pianoroll with stroke edits $\mathbf{x}$, we can generate a realistic pianoroll that follows the editing information, keeping the other parts from being changed. Stroke-based editing allows users to polish a given pianoroll to their liking, which is useful for eliminating bad cases and personalized music generation. The mask $\boldsymbol{\Omega}$ is set to 1 only for the edited regions and the reverse steps t_0 is set to 0.4.

Inpainting/outpainting. Similar to image inpainting (Yeh et al., 2017) and outpainting (Wang et al., 2014), we would like to reconstruct missing regions or extend the border of existing pianoroll $\mathbf{x}$. Inpainting can be used for music detail filling and outpainting can be used for music continuation, which is important to generate music pieces with flexible lengths. For inpainting, the mask $\boldsymbol{\Omega}$ is set to 1 for the missing regions and the reverse steps t_0 is set to 1. And for outpainting, we concatenate the pianoroll and a random initialized part as input $\mathbf{x}$ and set the mask $\mathbf{\Omega}$ to 1 only for the randomly initialized regions and the reverse steps t_0 to 1.

Combination. Another important scenario is combining several music segments together harmoniously, which can be applied when someone likes several segments and wants them to appear in the same musical piece. We concatenate the pieces with some parts which are sampled from the prior distribution p_T, and use this as input $\mathbf{x}$. Similar to outpainting, the mask $\mathbf{\Omega}$ is set to 1 only for the randomly initialized regions and the number of reverse steps t_0 is set to 1.

Style transfer. As described in Wu & Yang (2021), given an existing musical piece $\mathbf{x}$, we can change it to another style by adjusting some control signals such as rhythmic intensity, note density, pitch distribution, etc. For example, if the note density of a certain music piece increases, the music piece will sound more intense or upbeat. We can achieve precise local control cause control signals in SDMuse are fine-grained. The mask $\mathbf{\Omega}$ is set to 1 for all regions and t_0 is set to 0.4.

Algorithm 1 Fine-grained Music Editing (VP-SDE)

Require: $\mathbf{x}$ (the input pianoroll), $\mathbf{\Omega}$ (mask for edited regions), t_0 (reverse steps, SDE hyper-parameter), N (diffusion steps), K (total repeats)

$\Delta t \leftarrow \frac{t_0}{N}$

$\mathbf{x}_0 \leftarrow \mathbf{x}$

$\alpha(t_0) \leftarrow \prod_{i=1}^{N}(1-\beta(\frac{it_0}{N})\Delta t)$

for $k \leftarrow 1$ **to** K **do**

  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

  $\mathbf{x} \leftarrow \left[(\mathbf{1}-\mathbf{\Omega}) \odot \sqrt{\alpha(t_0)}\ \mathbf{x}_0 + \mathbf{\Omega} \odot \sqrt{\alpha(t_0)}\ \mathbf{x} + \sqrt{1-\alpha(t_0)}\ \mathbf{z}\right]$

  for $n \leftarrow N$ **to** 1 **do**

    $t \leftarrow t_0 \frac{n}{N}$

    $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

    $\alpha(t) \leftarrow \prod_{i=1}^{n}(1-\beta(\frac{it_0}{N})\Delta t)$

    $\mathbf{x} \leftarrow \left\{(\mathbf{1}-\mathbf{\Omega}) \odot \left(\sqrt{\alpha(t)}\ \mathbf{x}_0 + \sqrt{1-\alpha(t)}\ \mathbf{z}\right) + \mathbf{\Omega} \odot \left[\frac{1}{\sqrt{1-\beta(t)\Delta t}}\left(\mathbf{x} + \beta(t)\Delta t \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}, t) + \sqrt{\beta(t)\Delta t}\ \mathbf{z}\right)\right]\right\}$

  end for

end for

return $\mathbf{x}$

3.3 MIDI-EVENT GENERATION STAGE

The MIDI-event generation stage is designed for refining the generated pianorolls from the prior stage with more precise music performance details by generating the final MIDI-event sequence auto-regressively, thus being able to benefit from the advantages of MIDI-event representation for music generation. This stage contains a score encoder to encode music scores (pianoroll) into score condition and an auto-regressive decoder to generate MIDI-event tokens step by step. The score encoder (see Figure 5a in Appendix B.1) is a 12-layer convolution network with group normalization, which takes the synthesized pianorolls as input to generate the corresponding score condition. The score condition is then concatenated with *barpos embedding*, which is used to indicate the position and introduced in Ren et al. (2020), and tiled to the same length with decoder input (MIDI-event token sequence) according to the bar information from decoder input. We call the set of these operations as "*expand by bar*" and illustrate it in Figure 5b in Appendix B.1. Finally, the score condition can be added to the decoder input and forwarded to the auto-regressive decoder. The auto-regressive decoder is a transformer decoder (Vaswani et al., 2017; Dai et al., 2019), which can also be regarded as a conditional language model, helping with eliminating outliers predicted in pianorolls and adding more precise music performance details.

4 EXPERIMENTS

In this section, we first introduce the experimental setup including dataset, evaluation metrics, baselines, etc. Then we report the results of unconditioned and conditioned generation with SDMuse. And finally, we show the performance of SDMuse in various fine-grained music editing tasks with evaluation results and cases. The audio samples can be found in our demo page [1].

[1] https://SDMuse.github.io/posts/sdmuse/

Table 2: The evaluation metrics of SDMuse in terms of both *quality* and *controllability*.

	Quality	*Controllability*
Objective	pitch distribution similarity (PD) duration distribution similarity (DD)	control signal distance (CSD) overlap ratio (OR)
Subjective	overall perceptive score	overall consistency score

4.1 Experimental settings

Datasets. In our experiments, we use the *ailabs1k7* dataset introduced by Hsiao et al. (2021), which contains 1,748 pieces of pop piano performance. We process all the pieces in training set into 32-bar segments by sliding window, with window size of 32 bar and hopping size of 4 bar, thus obtain around 15,000 segments for the training of conditioned diffusion model in pianoroll generation stage and encoder-decoder in MIDI-event generation stage. For pianoroll, we set the granularity to 1 beat, which means the length of pianoroll n is the beat number of the corresponding musical piece. And for MIDI-event sequence, we follow the representation in Ren et al. (2020), ignoring the track and instrument information because the music of *ailabs1k7* dataset is single-track polyphonic music.

Evaluation metrics. We evaluate our results based on *quality* and *controllability*, for which we performed both objective and subjective assessments. As listed in Table 2, to evaluate *quality*, we use PD and DD scores introduced in Sheng et al. (2021) and conduct subjective evaluation to obtain the overall perceptive scores. On the other hand, for quantifying *controllability* of conditioned music generation and editing, we calculate the L_2 distance of control signals (CSD) between the given one and those of generated output. Also, when conducting fine-grained music editing like stroke-based generation, we compute the overlap ratio (OR) between the generated pianorolls and the input stroke pianorolls. And we evaluate the consistency of the edited samples in fine-grained music editing subjectively. Similar to Zhang et al. (2022); Guo et al. (2022), we invite 10 participants with music knowledge to give their scores (five-point scales, 1 for bad and 5 for excellent) of randomly selected samples. The detailed instruction given to annotators are posted in Appendix C.

Baselines. For comparison, we choose different types of symbolic music generation and style transfer models as our baselines: 1) REMI (Huang & Yang, 2020); 2) CPW (Hsiao et al., 2021) and 3) MuseMorphose (Wu & Yang, 2021). We use the official implementation of each model and train these three models with the same training set. Considering that we are the first one to conduct fine-grained music editing tasks, we only compare the quality of generated outputs with these baselines.

Model configuration. For the pianoroll generation stage, we use Gaussian diffusion model [2] and adjust the UNet architecture (Çiçek et al., 2016) to make sure it can take control signals as condition. And for the MIDI-event generation stage, we use a 4-layer transformer decoder and 12-layer convolution 1D encoder. Other details about the model hyper-parameters are listed in the Appendix B.4.

Training setup. The training data of both modules is cut into 32-bar segments. We train the diffusion model with diffusion step of 100 and use the linear noise schedule with max beta of 0.02. The training process of the pianoroll generation stage takes about 2 days on 1 A100 GPU with batch size of 32 pianorolls. And the MIDI-event generation stage are trained around 12 hours on 1 A100 GPU with batch size of 2000 MIDI-event tokens.

4.2 Generation from scratch

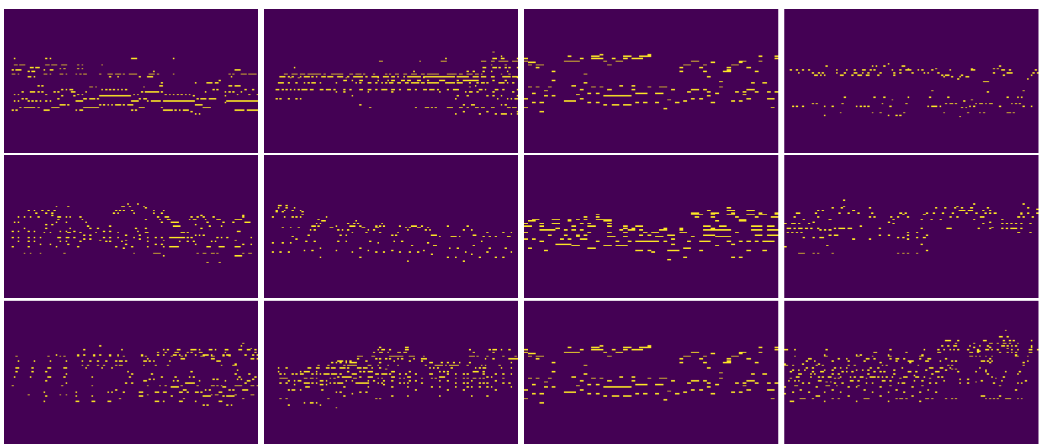
4.2.1 Unconditioned generation

We first evaluate the performance of SDMuse on the unconditioned music generation task by just setting the control signals $\boldsymbol{c}$ as $\boldsymbol{c}_{null}$ in the pianoroll generation stage. As shown in Table 3, denoted as SDMuse (unconditioned), while unconditioned generation is not our primary goal, we find that SDMuse achieves comparable results to the state-of-the-art music generation models, which indicates the effectiveness of our training strategy of diffusion probabilistic model that switches between unconditioned and conditioned generation settings. We present qualitative comparison results in Figure 3 by just showing the pianorolls extracted from the final outputs of SDMuse and baselines.

[2]https://github.com/openai/guided-diffusion

Table 3: Objective and subjective results of baseline systems in music generation, and SDMuse in generation from scratch tasks (both unconditioned and conditioned on given control signals) and fine-grained music editing tasks in terms of ***quality***. Settings with * notation have high PD, DD and subjective perceptive scores because they are based on existing musical pieces with only minor edits. The overall perceptive scores are calculated with 95% confidence intervals.

Task	Model / Setting	Objective		Subjective
		PD ↑	DD ↑	overall perceptive score ↑
	GT	–	–	4.07 (±0.09)
Generation	REMI (Huang & Yang, 2020)	0.82	0.76	3.52 (±0.07)
	CPW (Hsiao et al., 2021)	0.74	0.80	3.71 (±0.06)
	SDMuse (unconditioned)	0.84	**0.81**	3.69 (±0.06)
	SDMuse (conditioned)	**0.88**	0.79	**3.72** (±0.06)
Editing	MuseMorphose (Wu & Yang, 2021)*	0.68	0.81	3.80 (±0.06)
	SDMuse (stroke-based generation)	0.84	0.80	3.47 (±0.07)
	SDMuse (stroke-based editing)*	0.96	0.88	3.81 (±0.07)
	SDMuse (inpainting)*	0.96	0.87	3.70 (±0.06)
	SDMuse (outpainting)	0.79	0.75	3.63 (±0.07)
	SDMuse (combination)	0.86	0.83	3.59 (±0.09)
	SDMuse (style transfer)*	0.92	0.80	3.77 (±0.07)



(a) REMI (b) CPW (c) MuseMorphose (d) SDMuse (Ours)

Figure 3: The pianorolls extracted from the music generated by SDMuse and baseline systems.

4.2.2 Conditioned Generation

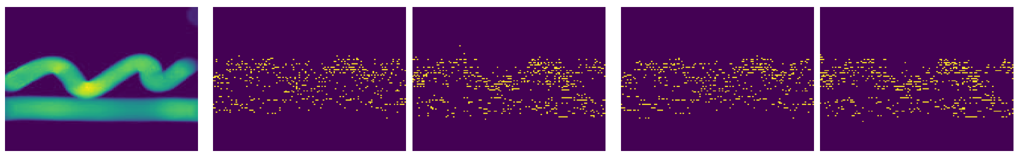
In order to assess the performance of SDMuse when generating musical pieces from scratch conditioned on given control signals, we use the control signals extracted from the test set during the pianoroll generation stage. The *quality* results are also listed in Table 3, denoted as SDMuse (conditioned), demonstrating that with the guidance of explicit music information from control signals, SDMuse can obtain better generation quality with reasonable listening experience compared to unconditioned generation. And the *controllability* results are presented in Table 4, indicating that SDMuse has the ability to generate musical pieces based on the control signals faithfully. Also, as a complement, we compare the note density $\boldsymbol{c}_n$ and the pitch distribution $\boldsymbol{c}_p$ extracted from output music with the given ones, illustrated in Figure 8c, for a visual demonstration of the faithfulness in terms of control signals during conditioned generation.

4.3 Fine-grained Editing

For aforementioned fine-grained editing tasks (see Section 3.2.3 for details), we edit existing musical pieces in corresponding ways and evaluate these tasks respectively. The *quality* results are shown

Table 4: Objective and subjective results of style transfer baseline and SDMuse in conditioned music generation task and various fine-grained music editing tasks in terms of ***controllability***. The subjective scores are calculated with 95% confidence intervals.

Model / Setting	Objective			Subjective
	CSD ($\boldsymbol{n}_n$) ↓	CSD ($\boldsymbol{n}_p$) ↓	OR ↑	overall consistency score ↑
MuseMorphose (Wu & Yang, 2021)	–	–	–	3.87 (±0.07)
SDMuse (conditioned)	0.06	0.15	–	3.77 (±0.06)
SDMuse (stroke-based generation)	–	–	0.85	3.43 (±0.10)
SDMuse (stroke-based editing)	–	–	0.81	3.89 (±0.08)
SDMuse (style transfer)	0.12	0.38	–	**4.02** (±0.06)



(a) Input. (b) Generated pianorolls. (c) Extracted pianorolls of final outputs.

Figure 4: The pianorolls generated by the pianoroll generation stage and extracted from the final output music generated by MIDI-event generation stage in stroke-based generation task.

in Table 3 and the *controllability* results are shown in Table 4. It is obvious that the tasks with only minor edits, such as stroke-based editing, inpainting and style transfer, perform well in both *quality* and *controllability*. For the stroke-based generation which highly depends on the given stroke pianorolls, there is a trade-off between the PD/DD scores and OR score. Figure 4 illustrates the process of stroke-based generation task, including the input stroke pianoroll $\mathbf{x}$, the output of pianoroll generation stage, and the pianorolls extracted from the final MIDI-event sequence. Illustrations of other fine-grained editing tasks and the final musical pieces can be found in our demo page [3].

4.4 Method analyses

We conduct more explorations on SDMuse and put the results in Appendix D due to the limited space, including: 1) the refinement performance of MIDI-event generation stage to verify the ability of eliminating outliers and adding music performance details; 2) the comparison among different embedding ways of control signals. In summary, it is observed that MIDI-event generation stage is good at refining pianorolls with outliers. And when involving control signals in conditioned diffusion model, word embedding and direct embedding outperform positional embedding.

5 Conclusion

In this paper, we propose SDMuse, a unified Stochastic Differential Music editing and generation framework via hybrid representations. SDMuse can not only compose whole musical pieces from scratch (both unconditionally and conditioned on given control signals), but also edit existing musical pieces in different ways according to various demands. As two different symbolic music representations, pianoroll is more appropriate for extracting and controlling perceptive music information, such as structure, while MIDI-event is more ideal for generating and modeling music performance details. Thus, SDMuse contains pianoroll and MIDI-event generation stages to take advantage of hybrid representations. The first stage is based on a diffusion model generative prior and synthesizes or edits pianorolls by iteratively denoising through SDE. And the second stage refines pianorolls with music performance details by generating MIDI-event sequences auto-regressively. Objective and subjective results on *ailabs1k7* dataset demonstrate the effectiveness of our proposed stochastic differential music editing/generation process and hybrid representations. In the future, we plan to deploy SDMuse as an interactive website to make it accessible to more people who are interested in it, as well as extend it to other music genres.

[3] https://SDMuse.github.io/posts/sdmuse/

REFERENCES

Brian DO Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326, 1982.

Gino Brunner, Yuyi Wang, Roger Wattenhofer, and Sumu Zhao. Symbolic music genre transfer with cyclegan. In *2018 ieee 30th international conference on tools with artificial intelligence (ictai)*, pp. 786–793. IEEE, 2018.

Özgün Çiçek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: learning dense volumetric segmentation from sparse annotation. In *International conference on medical image computing and computer-assisted intervention*, pp. 424–432. Springer, 2016.

Ondřej Cífka, Umut Şimşekli, and Gaël Richard. Groove2groove: one-shot music style transfer with supervision from synthetic data. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:2638–2650, 2020.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2978–2988, 2019.

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021.

Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Fenfei Guo, Chen Zhang, Zhirui Zhang, Qixin He, Kejun Zhang, Jun Xie, and Jordan Boyd-Graber. Automatic song translation for tonal languages. In *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 729–743, 2022.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 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. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 178–186, 2021. URL https://github.com/YatingMusic/compound-word-transformer.

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.

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*, pp. 1180–1188, 2020. URL https://github.com/YatingMusic/remi.

Zeqian Ju, Peiling Lu, Xu Tan, Rui Wang, Chen Zhang, Songruoyao Wu, Kejun Zhang, Xiangyang Li, Tao Qin, and Tie-Yan Liu. Telemelody: Lyric-to-melody generation with a template-based two-stage method. *arXiv preprint arXiv:2109.09617*, 2021.

Jinglin Liu, Chengxi Li, Yi Ren, Feiyang Chen, Peng Liu, and Zhou Zhao. Diffsinger: Diffusion acoustic model for singing voice synthesis. 2021.

Chenlin Meng, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. Sdedit: Image synthesis and editing with stochastic differential equations. *arXiv preprint arXiv:2108.01073*, 2021.

Gautam Mittal, Jesse Engel, Curtis Hawthorne, and Ian Simon. Symbolic music generation with diffusion models. *arXiv preprint arXiv:2103.16091*, 2021.

Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*, 2021.

Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2536–2544, 2016.

Yi Ren, Yangjun Ruan, Xu Tan, Tao Qin, Sheng Zhao, Zhou Zhao, and Tie-Yan Liu. Fastspeech: Fast, robust and controllable text to speech. *Advances in Neural Information Processing Systems*, 32, 2019.

Yi Ren, Jinzheng He, Xu Tan, Tao Qin, Zhou Zhao, and Tie-Yan Liu. Popmag: Pop music accompaniment generation. In *Proceedings of the 28th ACM International Conference on Multimedia*, pp. 1198–1206, 2020.

Zhonghao Sheng, Kaitao Song, Xu Tan, Yi Ren, Wei Ye, Shikun Zhang, and Tao Qin. Songmass: Automatic song writing with pre-training and alignment constraint. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 13798–13805, 2021.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Miao Wang, Yu-Kun Lai, Yuan Liang, Ralph R Martin, and Shi-Min Hu. Biggerpicture: data-driven image extrapolation using graph matching. *ACM Transactions on Graphics*, 33(6), 2014.

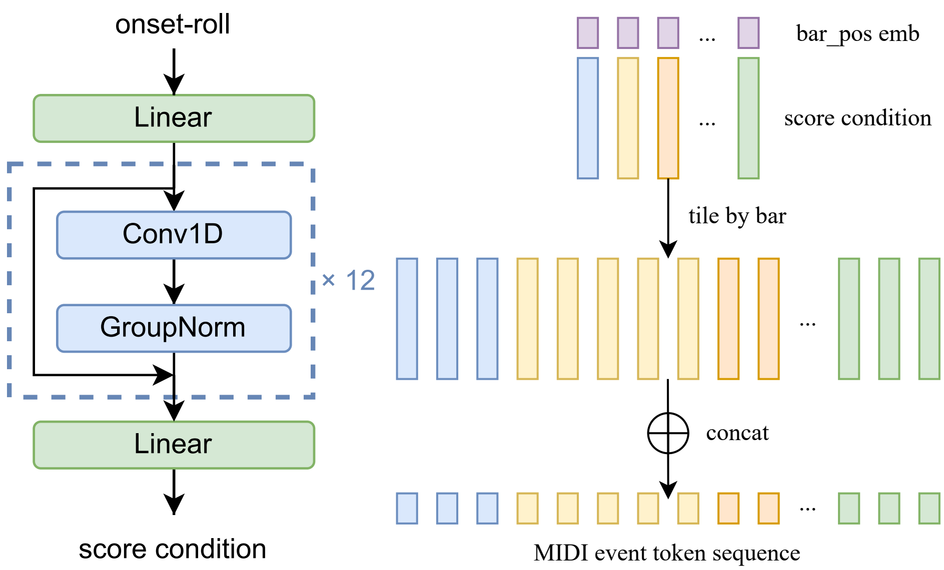
Shih-Lun Wu and Yi-Hsuan Yang. Musemorphose: Full-song and fine-grained music style transfer with one transformer vae. *arXiv preprint arXiv:2105.04090*, 2021. URL `https://github.com/YatingMusic/MuseMorphose`.

Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. Midinet: A convolutional generative adversarial network for symbolic-domain music generation. In *International Society for Music Information Retrieval*, 2017.

Raymond A Yeh, Chen Chen, Teck Yian Lim, Alexander G Schwing, Mark Hasegawa-Johnson, and Minh N Do. Semantic image inpainting with deep generative models. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5485–5493, 2017.

Chen Zhang, Yi Ren, Xu Tan, Jinglin Liu, Kejun Zhang, Tao Qin, Sheng Zhao, and Tie-Yan Liu. Denoispeech: Denoising text to speech with frame-level noise modeling. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 7063–7067. IEEE, 2021.

Chen Zhang, Luchin Chang, Songruoyao Wu, Xu Tan, Tao Qin, Tie-Yan Liu, and Kejun Zhang. Relyme: Improving lyric-to-melody generation by incorporating lyric-melody relationships. *arXiv e-prints*, pp. arXiv–2207, 2022.



(a) The structure of score encoder. (b) An illustration of *expand_by_bar* operator.

Figure 5: The details of score encoder and *expand_by_bar* operator in MIDI-event generation stage.

A Symbolic music representation

MIDI-event and pianoroll are two of the most common music representations in symbolic music generation works. As listed in Table 1, MIDI-event and pianoroll have their own advantages and disadvantages when representing a piece of symbolic music. An pianoroll (see Figure 1b) is like an image, with the horizontal axis representing time and the vertical axis representing pitch. It is closer to the way how humans perceive musical pieces and contains more prior music information. Pitch, duration, and relative position of notes are directly shown on it, so one can easily perceive the music structure, note density, etc. of an entire song when given a pianoroll, which makes it more suitable to be edited by humans.

Though the MIDI-event (see Figure 1c) contains all information in pianoroll, when converting the MIDI-event tokens into embeddings whose weights are randomly initialized, the correlations between different MIDI-events are lost and need to be relearned from the training data. Given the circumstance that the amount of high-quality symbolic music training data is limited, it is relatively easier for a deep learning model to extract musical information from pianoroll than MIDI-event. However, besides MuseGAN (Dong et al., 2018) that utilizes pianorolls, most symbolic music generation works (Huang et al., 2018; Huang & Yang, 2020; Zhang et al., 2022) are trained with MIDI-event sequences. This is due to MIDI-event sequence can carry more precise details than pianoroll, such as velocity, which can enhance the richness and expressiveness of the generated music performance. Furthermore, there is no difference in the way that a pianoroll treats onset and other positions of a note, which is not consistent with real music. MIDI-event, however, regards a note as the unit and is more robust to generate musical pieces.

B Model details

B.1 Details of model architecture

As shown in Figure 5a, the score encoder of the MIDI-event generation stage in SDMuse has a simple architecture: a 12-layer convolution network with group normalization, as well as two linear layers to process input pianoroll and output score condition. And the detailed illustration of *expand_by_bar* operator is in Figure 5b. The score condition is then concatenated with *barpos embedding*, which is used to indicate the position and introduced in Ren et al. (2020), and tiled to the same length with decoder input (MIDI-event token sequence) according to the bar information from decoder input. At last, the expanded score condition is add to decoder input.

B.2 Fine-grained control signals

Denote the processed pianoroll (onsetroll) as $X_o \in \{0, 1\}^{m \times n}$, where m means the number of pitch and n represents the length of pianoroll. Here we design three fine-grained control signals to provide precise controllability over the entire piece.

- Note density: for each piece of music, we can extract the note density vector $\boldsymbol{c}_n \in [0, 127]^{1 \times n}$, which indicates how many onsets occur concurrently at each timestep.
- Pitch distribution: the pitch distribution $\boldsymbol{c}_p \in [0, 127]^{m \times 1}$ is a vector that represents the distribution of note pitch over the whole musical piece. Specifically, the value in $\boldsymbol{c}_p$ of position p is the number of notes whose pitch is p.
- Chord progression: we extract the chord progression sequence $\boldsymbol{c}_c \in [0, 96]^{1 \times n}$ from the original musical piece, where 96 is the number of chord type.

In order to involve these fine-grained control signals into diffusion, we first convert $\boldsymbol{c}_n, \boldsymbol{c}_p, \boldsymbol{c}_c$ to embeddings $\boldsymbol{e}_n, \boldsymbol{e}_p, \boldsymbol{e}_c$ and tile them to the same shape of $m \times n$, which are then concatenated to x_t (the data at t step).

B.3 Onsetroll

Denote the original pianoroll in the dataset as $X_p \in \{0, 1\}^{m \times n}$, where m means the number of pitch, n means the length of pianoroll and the value is set to 1 when the corresponding position is belong to a note otherwise set to 0. As mentioned in Section 2.1, onset and other positions of the note are treated non-differently in pianoroll, however, in real music, onset is more important than other positions and it directly affects the listening experience. If an extra onset is predicted, it indicates an extra note, while if an extra other position is predicted, it just represents the corresponding note becomes slightly longer. Thus, to grasp the factor that takes care of most, we process the original pianoroll X_p to onsetroll X_o by only keeping the onset information and discarding duration information of each note. Specifically, only the value of the onset position in the pianoroll will remain 1 and other positions of notes will be set to 0.

B.4 Model configuration

B.4.1 Diffusion probabilistic model

Our diffusion probabilistic model is implemented based on the Gaussian diffusion model [4]. The size of the input pianoroll is 128×128 (contains 32 bars and 128 beats). We set the learning rate as $1e-4$, the diffusion step as 100. For the positive noise function $\beta(t)$ in VP-SDE, we follow Song et al. (2020); Ho et al. (2020); Dhariwal & Nichol (2021) and set:

$$\beta(t) = \beta_{start} + t(\beta_{end} - \beta_{start}), \tag{7}$$

where $\beta_{start} = 0.1$ and $\beta_{end} = 20$ in our implementation. And we found that if only using the conditioned diffusion model, β_{end} should be smaller, such as 5 to achieve better performance. The embedding dim of control signal embeddings $\boldsymbol{e}_n$, $\boldsymbol{e}_p$, $\boldsymbol{e}_c$ is set to 32 and p_{uncond} is set to 0.5.

B.4.2 Auto-regressive decoder

In our implementation, the auto-regressive decoder is based on a Transformer decoder with 4 layers. The output of the auto-regressive decoder is split into three parts: pitch, velocity, and duration, and the total loss is calculated by adding the cross entropy losses of these three parts together. The dropout of the auto-regressive decoder is 0.1, the hidden size is 256, and the number of heads is 4.

Instruction:

The workbook name corresponds to the folder name, and the duration of each music clip is about 30s-60s, so you can take a proper break to adjust after doing a folder.
Note: All files have been shuffled out of order and then grouped into folders, so files within a folder cannot be considered to belong to the same source.

Scoring criteria: (1: bad, 2: poor, 3: fair, 4: good, 5: excellent)

1. **Quality: The overall quality of the assigned clip, including whether it is pleasant to listen to and whether it has some structure and variability.**
    - 1: the music is very confusing, many wrong notes, no musicality;
    - 2: many wrong notes, but with some audible parts;
    - 3: some wrong notes, or a little confusing, with some musicality;
    - 4: a few wrong notes, the overall listening experience is pleasant, allowing some rhythmic chaos;
    - 5: almost no wrong notes, overall pleasant to listen to.
2. **Consistency: Specify the consistency of the assigned clip, whether there is a small fragment in the clip that is very abrupt, or whether the clip is overall harmonious.**
    - 1: many particularly obvious articulation failure or abruptness;
    - 2: one or two particularly obvious articulation failure or abruptness;
    - 3: one or two minor articulation failure or abruptness;
    - 4: the overall is harmonious and consistent, with a slight articulation or abruptness case;
    - 5: the overall is harmonious and consistent, no slight articulation or abruptness.

Figure 6: The instructions we give to participants of subjective evaluation part.

C SUBJECTIVE EVALUATION

We invite 10 people with musical knowledge to give their scores as our subjective evaluation. Here are the instructions we provide to them (Figure 6). The five-point scale of subjective evaluation is similar to MOS, which has been widely used in different speech synthesis work (Ren et al., 2019; Zhang et al., 2021).

D METHOD ANALYSES

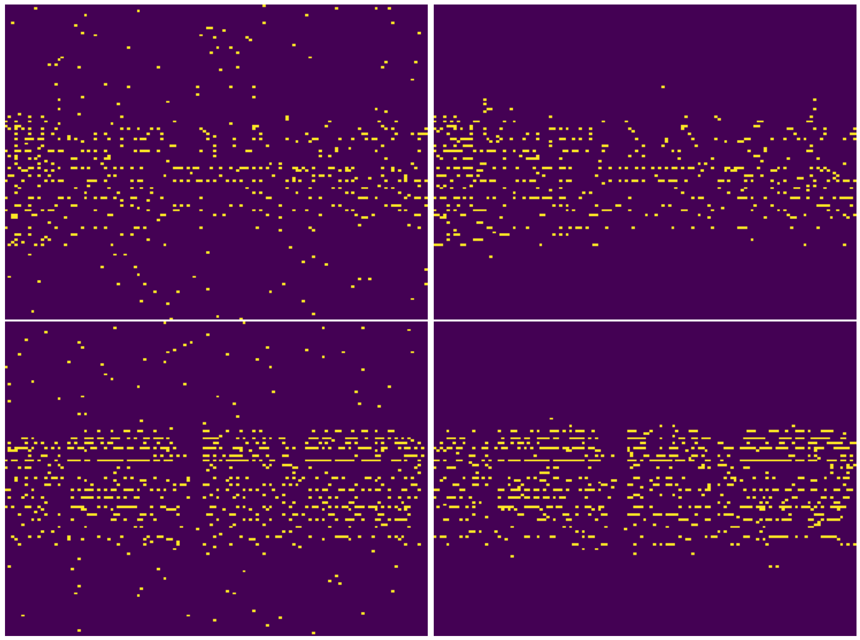
D.1 REFINEMENT PERFORMANCE

Here we illustrate the refinement performance of the MIDI-event generation stage. We compare the pianorolls input of this stage and the pianorolls extracted from the output MIDI-event sequences as in Figure 7. We manually added outliers to the input pianoroll to explore the ability of SDMuse in eliminating outliers. As shown in Figure 7, these added outliers are removed after the MIDI-event generation stage. Also, the positions of some notes are refined in the MIDI-event sequence generation stage auto-regressively. For a more intuitive listening experience, please refer to our demo page, where we post the audio synthesized from the output of two stages respectively for comparison.

D.2 EMBEDDING WAYS OF CONTROL SIGNALS

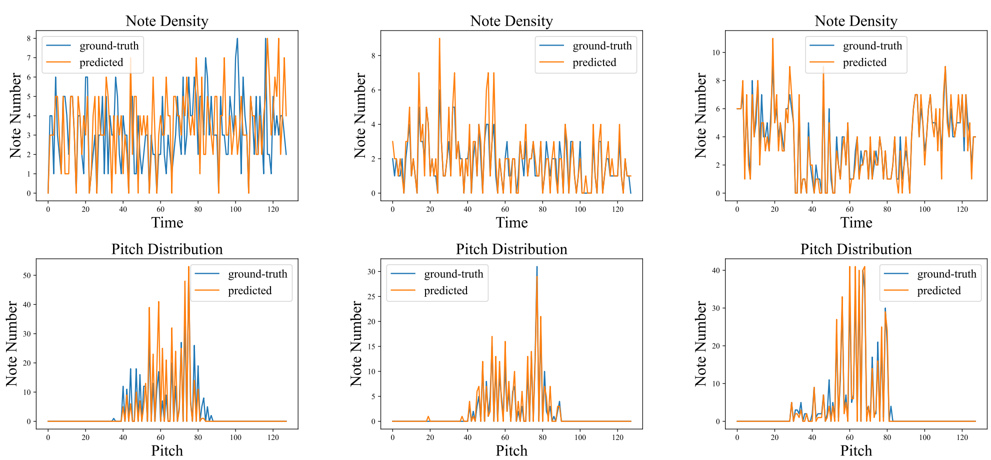
When involving control signals $\boldsymbol{c}_n$ and $\boldsymbol{c}_p$ in the diffusion probabilistic model of the pianoroll generation stage, there are several embedding ways: 1) positional embedding: convert the control signals into sinusoidal positional embeddings; 2) direct embedding: regard control signals as vectors and use them as embeddings directly; 3) word embedding: convert the control signals to randomly initialized word embeddings. As shown in Figure 8, the ways of direct embedding and word embedding show good performance in terms of the control signals' faithfulness.

[4]https://github.com/openai/guided-diffusion



(a) Input of MIDI-event generation stage. (b) Output of MIDI-event generation stage.

Figure 7: The input pianorolls of MIDI-event generation stage and the pianorolls extracted from the output MIDI-event sequences. We highlight some of the eliminated outliers with red boxes for better identify.



(a) Positional embedding. (b) Direct Embedding. (c) Word Embedding.

Figure 8: The visualization of the difference in $\boldsymbol{c}_n$ and $\boldsymbol{c}_p$ between input and output with different control signal's embedding ways.