DDSP-Based Neural Audio Synthesis Model with Continuous Timbre Controls

Aayush Kapur
School of Computer Science
McGill University
aayush.kapur@mail.mcgill.ca

Eto Sun
Schulich School of Music
McGill University
eto.sun@mail.mcgill.ca

Junrui Huang
Mechanical Engineering
McGill University
junrui.huang@mail.mcgill.ca

Yanting Zhou
Mechanical Engineering
McGill University
yanting.zhou@mail.mcgill.ca

Abstract

This project focuses on the development of a neural audio synthesis model that utilizes deep learning to generate audio signals. We aim to develop a deep learning model with controllable features in the latent space for timbre control based on the differentiable digital signal processing (DDSP) framework. Our proposed model replaces the original DDSP autoencoder architecture with a Variational Autoencoder (VAE) architecture, adds timbre descriptors as extra latent features to the decoder, and changes the loss function correspondingly to reflect timbre distance. Various objective and subjective evaluations are conducted to test the performance of our proposed model. Compared with the original model, our proposed model has higher accuracies for the "pitch" and timbre descriptors of the reconstruction. In addition, our VAE architecture can provide continuous control for the latent features. By providing intuitive controls for different aspects of timbre, our proposed model will empower composers, sound designers, and researchers to manipulate sound in a more nuanced and meaningful way.

1 Introduction

1.1 Background

Neural audio synthesis, which refers to synthesizing audio signals using neural networks, has yielded impressive results on various problems in the field of digital audio signal processing, such as musical instrument synthesis (Renault et al., 2022), timbre transposition (Engel et al., 2020), sound matching (Masuda and Saito, 2023), and performance rendering (Wu et al., 2022).

There are two main categories of neural audio synthesis approaches: autoregressive (AR) procedure-based and differentiable signal model-based. The first one utilizes the AR procedure to estimate the future value based on a linear combination of the past values of a signal. Neural network structures, such as recurrent neural networks (RNNs) and generative adversarial networks (GANs) can be employed to learn the AR coefficients for the AR procedure (Défossez et al., 2018; Engel et al., 2019). While these models are capable of generating realistic samples, they need to learn more parameters and train on a larger dataset, which makes them difficult to use in real-time.

The differentiable digital signal processing (DDSP) framework addresses this limitation by introducing a differentiable way to integrate traditional signal synthesis models with neural networks (Hayes et al., 2024). This allows for a more efficient neural network with fewer parameters, which enables

real-time applications. Intuitive parameters such as pitch and loudness can be incorporated into the latent space of the neural networks to provide control parameters for the traditional signal synthesis model (such as spectral modeling synthesis). However, existing DDSP implementations often rely on autoencoders to generate parameters for the spectral modeling component. This can lead to non-continuous control in the latent space, hindering precise manipulation of the synthesis procedure.

Timbre is a crucial aspect of sound perception, which can be considered a multidimensional perceptual attribute that, together with pitch and loudness, affects a sound's "quality" or "texture". However, controlling timbre in neural audio synthesis remains a challenge due to the complex and subjective nature of timbre itself, which exhibits a constraint to the neural audio synthesis model.

Here, we propose a novel DDSP-based neural audio synthesis model that addresses these limitations. Our model aims to achieve two key objectives: 1) ensure continuous control within the latent space of the neural network architecture, and 2) introduce controllable features in the latent space that directly influence specific aspects of timbre. To achieve these goals, we leverage a Variational Autoencoder (VAE) architecture and incorporate timbre descriptors into the model, enabling more meaningful control over the generated sound's characteristics. This approach has the potential to manipulate sound in a more meaningful way.

1.2 Related work

In this section, we will introduce the original DDSP model proposed by Engel et al. (2020), which will be the baseline model for future comparison with our proposed models. It consists of 2 major parts: the autoencoder part and the differentiable spectral modeling synthesis (SMS) part. The encoder extracts the "pitch"[1] (F0 in the diagram) and a low-dimensional embedding $\mathbf{z}$ from the input audio signal. The "loudness"[2] is also extracted as a latent feature. These latent features will be fed to the decoder to generate parameters of the differentiable SMS components. Finally, the output signal will be synthesized using the SMS method. The overall structure of the model is shown in Figure 1.

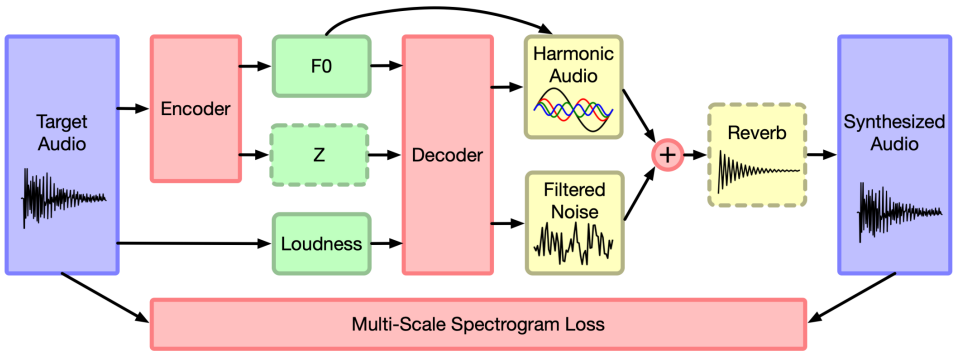

Figure 1: Overall structure of the original DDSP model in Engel et al. (2020).

We will now describe the details of the encoder, decoder, and differential SMS methods separately. The encoder first calculates the Mel Frequency Cepstrum Coefficients (MFCCs) from the input audio for capturing the spectral envelope. Then the MFCCs are passed through a normalization layer and a Gated Recurrent Unit (GRU). The output of the GRU will be fed into a linear layer to obtain the embedding $\mathbf{z}$. The "pitch" is estimated using a pre-trained CNN-based pitch estimation model named CREPE (Kim et al., 2018), and the "loudness" is extracted based on the A-weighting of the power spectrum, as in Hantrakul et al. (2019).

The decoder passes the extracted "pitch" (F0), "loudness", and $\mathbf{z}$ extracted in the previous step into a multilayer perceptron (MLP), respectively. Then, the outputs of these three MLPs are concatenated together and fed into a GRU. The output of the GRU will be further concatenated with the "pitch" and the "loudness". Finally, the concatenated result will pass through an MLP and then two linear layers

[1] Pitch is a perceptual variable for ordering sounds in a frequency-related scale, which is subjective. Therefore, we use the fundamental frequency (F0) as an estimation of pitch.

[2] Loudness is also a perceptual variable for ordering sounds in a sound pressure-related scale. Therefore, we use the perception-based weighting of the power spectrum as an estimation of loudness.

separately to get the parameters for synthesizing the harmonic component and noise component using the differentiable SMS method. The structure of the decoder is illustrated in Figure 2.

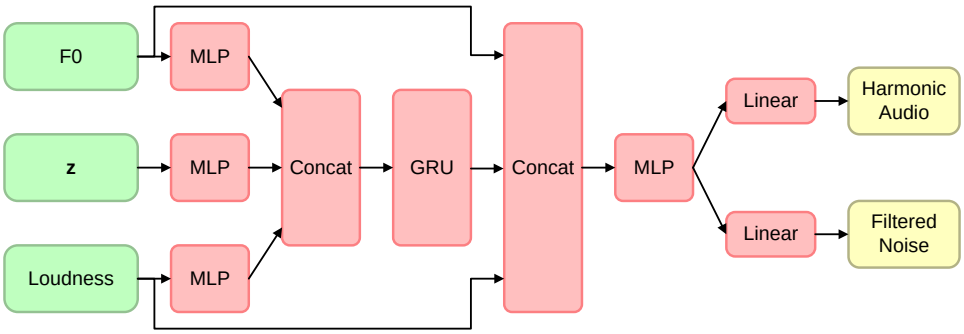

Figure 2: Structure of the decoder of the original DDSP model in Engel et al. (2020).

The SMS method represents an audio signal as a sum of sinusoids with plus filtered white noise (Serra and Smith, 1990). We assume that the sinusoids have slowly evolving amplitudes and frequencies so that we can express the *tonal* signal as:

$$y_{\text{tonal}}[n] = \sum_{k=1}^{K} A_k[n] \sin(\phi_k[n]) \tag{1}$$

where $A_k[n]$ is the time-varying amplitude of the k-th sinusoids and $\phi_k[n]$ is its instantaneous phase, which is obtained by integrating the instantaneous frequency $f_k[n]$:

$$\phi_k[n] = 2\pi \sum_{m=0}^{n} f_k[m] + \phi_{0,k} \tag{2}$$

where $\phi_{0,k}$ is the initial phase of the k-th sinusoids.

Here we further constrain our sinusoids to be harmonic, which means that the frequencies of these sinusoids are integer multiples of a fundamental frequency $f_0[n]$, i.e., $f_k[n] = kf_0[n]$.

The noise component y_{noise} can be obtained by filtering a white noise $\epsilon[n]$ with a linear time-variant (LTV) finite impulse response (FIR) filter h, which can be expressed as:

$$y_{\text{noise}}[n] = (\epsilon * h)[n] \tag{3}$$

where $*$ represents the convolution operator.

In practice, the white noise $\epsilon[n]$ is filtered in the frequency domain and converted to the time domain for more efficiency:

$$y_{\text{noise}}[n] = \text{IDFT}\{HX\}[n] \tag{4}$$

where H is the frequency response of the LTV-FIR filter h, X is the digital Fourier transform (DFT) of the white noise $\epsilon[n]$, and IDFT is the inverse digital Fourier transform (IDFT).

1.3 Organization of the paper

This paper is organized as follows: In Section 2, we present our VAE-based model with timbre control and describe the evaluation methods. In Section 3, we evaluate our proposed model and compare it with the model in the original DDSP paper. Finally, Section 4 summarizes this paper, outlines the strengths and limitations of our approach, and addresses some possibilities for future research.

2 Methodology

In this section, a more comprehensive problem formulation, the datasets and resources used, and the methodology implemented will be discussed in detail.

2.1 Problem Formulation and Constraints

This research aims to develop a neural audio synthesis model based on the DDSP framework that has the following three constraints.

First, the reconstruction signals should have high perceived quality. The ideal scenario is the two signals are identical in the time domain.

Second, the model should contain some controllable features in the latent space for specific timbre control. In this paper, we use objective timbre descriptors to estimate the perceptual dimensions of timbre (Peeters et al., 2011). The first two constraints lead to solving the following optimization problem:

$$\hat{\mathbf{y}} = \arg\min_{\mathbf{x}} \|f_T(\mathbf{y}) - f_T(\mathbf{x})\|_1 \text{ subject to } \|\mathbf{y} - \mathbf{x}\|_2^2 \leq \epsilon \tag{5}$$

where $\mathbf{y}$ is the input signal, $\hat{\mathbf{y}}$ is the reconstruction of $\mathbf{y}$, f_T is an operator that extracts timbre descriptors from a signal, and $\|\mathbf{x}\|_n$ is the ℓ_n norm of $\mathbf{x}$.

Third, the latent features in the latent space should have continuous control, which means the latent features should be formed as distributions.

2.2 Datasets and Resources

NSynth: We used the samples from the NSynth dataset (Engel et al., 2017). It is an audio dataset with 305,979 musical notes. Each sample note is a four-second, monophonic audio snippet with a 16kHz sampling rate. Further, the notes have been annotated using human evaluation and heuristic algorithms with the following information.

- **Source:** How the sound was produced for the note? Whether it was produced by acoustic instruments, electronic instruments or synthesized instruments?
- **Family:** Each instrument is a member of exactly one family which decides the note's family attribute.
- **Qualities:** Sonic qualities of the note. For example, there are spectral qualities such as bright and dark, and temporal qualities such as fast decay and long release.

In our project, we focused on a subset of samples from the NSynth dataset. We considered all the **acoustic source flute sound samples only**. We had a total of 6572 samples (802 MB in total). We implemented our deep learning model using the PyTorch framework[3]. The training was run on our setup locally. We had Nvidia RTX 3070 laptop with 8GB VRAM at our disposal.

2.3 Proposed Methods

2.3.1 Proposed Model 1: DDSP model with VAE

Pre-processing We extract the "pitch" (F0), "loudness", and MFCCs in the pre-processing stage to increase the computational efficiency of the code during training and testing. The F0 is obtained using the pre-trained CREPE[4], with hop size 256. The "loudness" is obtained by calculating the A-weighting of the power spectrum (Hantrakul et al., 2019). We use a Hann window with a window size of 2048 and a hop size of 256 for calculating the STFT. The MFCCs are calculated with window size 1024, hop size 256, 128 mels, and frequency from 20 Hz to 8000 Hz. Only the first 30 MFCCs are kept. We use the librosa library (McFee et al., 2023) to calculate the "loudness" and MFCCs. After pre-processing is performed, all features are stored as binary numpy arrays using `numpy.save` function as ".npy" files[5], which will be used in the training stage.

Encoder The original DDSP paper implements the traditional autoencoder in which each input is mapped to a specific output, which may not lead to continuous mapping. To add more diversity to the generated signal, we would like to learn a continuous distribution for the latent space $\mathbf{z}$, instead of

[3]Specifically, we use version 2.2.2 of PyTorch in `https://pytorch.org/`.

[4]We use the TensorFlow implementation of CREPE in `https://github.com/marl/crepe`.

[5]There are five binary ".npy" files saved, contains the original signals, "pitch", "loudness", MFCCs, and timbre descriptors respectively.

discrete values. Therefore, we achieve this by replacing the autoencoder in the original DDSP model with VAE. The VAE takes the extracted MFCCs as the input and passes them to a normalization layer (`nn.LayerNorm`) and then to a GRU layer (`nn.GRU`) with an input size of 30 and an output size of 512. After that, we will separate the network into 2 branches that generate the mean μ and the log variance $\log(\sigma^2)$. Each branch consists of a linear layer (`nn.Linear`) with 512 inputs and 16 outputs, a dropout layer (`nn.Dropout`) with a dropout rate of 0.01, and a LeakyReLU layer (`nn.LeakyReLU`) with a negative slope of 0.01. The output will be μ and $\log(\sigma^2)$. The VAE structure is illustrated in Figure 3.

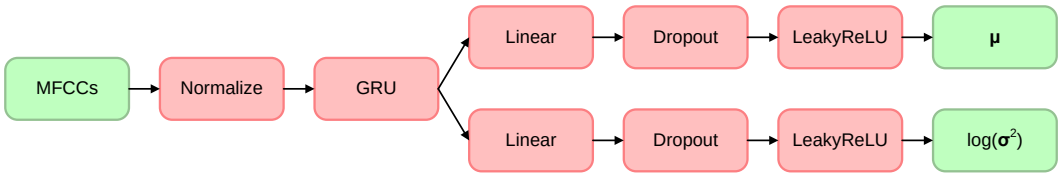

Figure 3: Structure of the proposed VAE component.

Decoder Our proposed decoder is similar to the decoder in the original DDSP paper. Since we change the original autoencoder to VAE, the decoder needs to sample points from the distribution generated by μ and $\log(\sigma^2)$ to obtain the latent embedding $\mathbf{z}$. To achieve this, we use the formula to randomly sample points from the generated distributions from the encoder:

$$\mathbf{z} = \mu + \epsilon \cdot \exp(\frac{1}{2}\log(\sigma^2)) \tag{6}$$

where ϵ is a random value sampled from the standard normal distribution.

Then, the preprocessed "pitch" (F0) and "loudness", and $\mathbf{z}$ in the encoder, will be fed into their MLP separately. We set the batch size to 16. Since our signals have a sampling rate $f_s = 16000$ Hz and a duration of 4 seconds, the number of blocks (or slices) of each signal will be 250. Therefore, the pitch and loudness tensors will have a size (16, 250, 1), and the $\mathbf{z}$ tensor will have a size (16, 250, 16).

Each MLP has three layers, each consisting of a linear layer, a normalization layer, and a LeakyReLU layer. The input size of the linear layer in the first layer corresponds to the input (pitch and loudness are 1, $\mathbf{z}$ is 16). The output sizes of all linear layers are set to 512, so the output tensor for each MLP will be (16, 250, 512).

The outputs of the three MLPs will be concatenated along the last axis of the tensors using `torch.cat` function and then fed into a GRU. The input size of the GRU is the MLP output's size times 3, and the output size is 512. The output of the GRU will be concatenated again with the pitch and loudness tensors and fed into another MLP (with the same structure as the previous one).

Then the output of the last MLP will go through 2 linear layers separately, one is used for synthesizing the tonal component (with an output size of 101) and another one is for the noise component (with an output size of 65). To synthesize the harmonic sinusoids in the tonal component, we first factorize the amplitudes $A_k[n]$ in Eq.(1) as:

$$A_k[n] = A[n]c_k[n] \tag{7}$$

where $A[n]$ is a global amplitude, $c[n]$ is a normalized distribution over harmonics that determines spectral variations, which satisfies $\sum_{k=0}^{K} c_k[n] = 1$ and $c_k[n] \geq 0$. In our implementation, the first output of the linear layer will be the global amplitude and the rest are for each harmonic sinusoid. All amplitudes will be rescaled to positive values using a modified sigmoid function in Engel et al. (2020):

$$a_{\text{rescaled}} = 2.0 \cdot \text{sigmoid}(a)^{\log 10} + 10^{-7}. \tag{8}$$

The instantaneous frequency is obtained from the "pitch" feature, and all sinusoids with a frequency above the Nyquist frequency ($f_s/2$, i.e., half of the sampling frequency) will be removed by setting its amplitude to 0[6]. Since these parameters are in a lower control rate (62.5 Hz) compared to the sampling rate (16000 Hz), we upsample these parameters from 250 to 64,000 samples using linear interpolation for applying the Eq.(1) to synthesize the tonal component.

[6]For numerical stability, we set the value to 10^{-4} instead of 0.

To synthesize the noise component, we first apply the same scale function in Eq.(8) to scale the sampled amplitude for each frequency bin, then convert it to an impulse response based on the frequency sampling method in Engel et al. (2020). First, the noise feature tensor will be stacked (by `torch.stack`) with a tensor with all 0 components of the same size of the noise feature tensor (by `torch.zeros_like`) along the last dimension. This is to specify the real part (all values in the noise feature tensor) and imaginary part (all 0) of the noise feature in order to make it complex in the next step. Then, `torch.view_as_complex` is used to convert the stacked tensor into a complex tensor. We will employ an inverse real fast Fourier transform (IRFFT) to the complex tensor using `torch.fft.irfft` function to get the impulse response. Next, we apply a Hann window (`torch.hann_window`) of size 257 to the impulse response for reducing the spectral leakage. The impulse response is shifted to zero-phase form (symmetric) before applying the window, and we convert it back to causal form before applying the filter, using `torch.roll` and `nn.functional.pad` functions. The noise component is constructed as the convolution of the filter and white noise in Eq.(4). The final output is the superposition of tonal ($\mathbf{y}_{\text{tonal}}$) and noise ($\mathbf{y}_{\text{noise}}$) components. The structure of the decoder is illustrated in Figure 4.

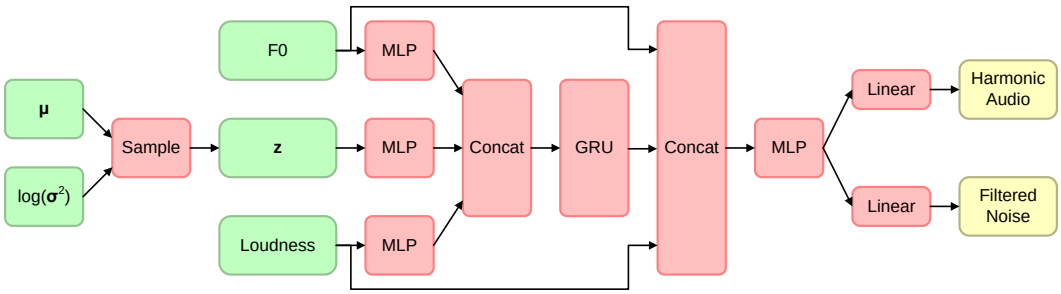

Figure 4: Structure of the proposed decoder component.

Loss Function Since VAE generates a distribution instead of discrete points, we need to change the loss function to the following, proposed by Caillon and Esling (2021):

$$S(x,y) = \sum_{n \in \mathcal{N}} \left[\frac{\|STFT_n(x) - STFT_n(y)\|_F}{\|STFT_n(x)\|_F} + \log(\|STFT_n(x) - STFT_n(y)\|_1) \right] \tag{9}$$

$$\mathcal{L}_{\text{vae}}(x) = \mathbb{E}_{x \sim p(x|z)}[S(x,\tilde{x})] + \beta \times D_{KL}[q_\phi(z|x) \| p(z)] \tag{10}$$

where $S(x,y)$ is a spectral distance that estimates the distance between real and synthesized waveforms (Caillon and Esling, 2021). The function STFT in $S(x,y)$ refers to the multiscale Short-Time Fourier Transform and n represents the scale (window size and hop size) for calculating the STFT. In our implementation, we use 6 window sizes (128, 256, 512, 1024, 2048, and 4096) and set the overlap to 75%, which means the hop sizes are 32, 64, 128, 256, 512, and 1024, respectively. The loss function $\mathcal{L}_{\text{vae}}(x)$ consists of 2 parts: the first part tries to minimize the likelihood of the data given $S(x,y)$; and the second part is a KL divergence that tries to match the predefined prior $p(z)$ and posterior distribution $q_\phi(z|x)$, multiplied by a coefficient β (Caillon and Esling, 2021). We set β to be 0.2 in our code.

Training First, we create the function "spectral_distance(x,y,scales)" to calculate $S(x,y)$ between the synthetic signal and the actual signal. The inputs "x" and "y" are the 2 signals and "scales" are the different window sizes that are described in the previous section. The function uses "torch.stft" to calculate the STFT of both signals; then we express Eq.(9), using "torch.norm()", and "torch.log()", to calculate the scale distance value. In the training loop, we first calculate $S(x,y)$; then the KL divergence is calculated using Eq.(10), and the loss is the sum of those 2 terms.

The actual training process is straightforward. The loss function is fully differentiable, therefore the standard back-propagation can be used to update the loss. In our code, we use "Torch.Tensor.backward()" to update the loss and train our model.

2.3.2 Proposed model 2: DDSP model with VAE & timbre

To enhance the controllability of the model, we add "timbre" to the proposed method. The "timbre" feature includes 3 timbre descriptors: spectral centroid, spectral flatness, and temporal centroid (Peeters et al., 2011). All 3 timbre descriptors are calculated and concatenated during the pre-processing stage. The spectral centroid (SC) represents the spectral center of gravity, which is related to auditory brightness. It is defined as:

$$\mathrm{SC}[m] = \sum_{k=1}^{K} \left(f_k[m] \cdot \frac{a_k[m]}{\sum_{l=1}^{K} a_l[m]} \right) \tag{11}$$

where K is the total number of bins of the one-sided spectrum, f_k is the frequency (in Hz) of the k-th bin, and a_k is the amplitude of the k-th bin.

The spectral flatness (SF) measures the randomness of the spectrum, which is related to the harshness of a sound. It is defined as:

$$\mathrm{SF}[m] = \frac{(\prod_{k=1}^{K} a_k)^{1/K}}{\frac{1}{K}\sum_{l=1}^{K} a_l}. \tag{12}$$

In order to make the spectral centroid and spectral flatness differentiable, the whole calculation process should be done in Pytorch. Therefore, we use the `spectrogram` function in torchaudio[7] to calculate the STFT of the signal. We calculate the time-variant version using STFT with a rectangle window. The window size and hop size are both set to 256 samples.

The temporal centroid (TC) represents the center of gravity of the energy envelope, which is related to a spectrotemporal dimension in timbre spaces (Kazazis et al., 2021). It is defined as:

$$\mathrm{TC} = \frac{\sum_{n=1}^{N} n \cdot e[n]}{f_s \cdot \sum_{m=1}^{N} e[n]} \tag{13}$$

where $e[n]$ is the energy envelope of a signal, N is the total samples of the energy envelope $e[n]$, and f_s is the sampling frequency.

In order to make the temporal centroid differentiable, we use the 1D max-pool (`nn.functional.max_pool1d` in Pytorch) to calculate the energy envelope of a signal. The input of the max-pool is the absolute value of the signal, with a kernel size of 256, a stride size of 128, and a padding size of 64. Notice that the temporal centroid is a scalar for a given signal, which means it's a time-invariant feature.

Pre-processing The pre-processing operations are the same as the procedure described in Section 2.3.1.

Encoder The implemented encoder is exactly the same as the proposed VAE in Section 2.3.1.

Decoder The decoder is implemented the same as the method described in Section 2.3.1. The only difference is that timbre descriptors will be concatenated with the output of the GRU, F0, and loudness. Since the temporal centroid is a time-invariant feature while the spectral centroid and spectral flatness are time-variant, it needs to be repeated 250 times so that it has the same size as the rest of the features. The structure of this decoder is shown in Figure 5.

Loss Function We incorporate an extra timbre loss $\mathcal{L}_{\mathrm{timbre}}$ to the VAE loss $\mathcal{L}_{\mathrm{vae}}$ in Eq. (10), so that the loss function is:

$$\mathcal{L} = \mathcal{L}_{\mathrm{vae}} + \alpha \mathcal{L}_{\mathrm{timbre}} \tag{14}$$

where α is a factor for balancing the back-propagation between the reconstruction quality of the audio signal and timbre descriptors' distance. We set $\alpha = 1$ in our training process.

The timbre loss is defined as the sum of three timbre descriptors' ℓ_1 distance, which can be formulated as:

$$\mathcal{L}_{\mathrm{timbre}} = \|\mathrm{ERB}(\mathrm{SC}(\hat{\mathbf{y}})) - \mathrm{ERB}(\mathrm{SC}(\mathbf{y}))\|_1 + \|\mathrm{SF}(\hat{\mathbf{y}}) - \mathrm{SF}(\mathbf{y})\|_1 + \|\mathrm{TC}(\hat{\mathbf{y}}) - \mathrm{TC}(\mathbf{y})\|_1 \tag{15}$$

[7] https://pytorch.org/audio/

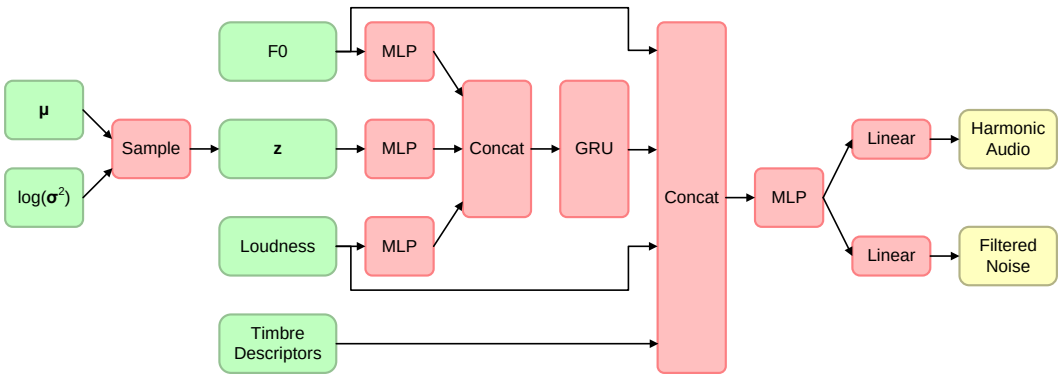

Figure 5: Structure of the proposed VAE decoder with timbre descriptors.

where $\mathbf{y}$ is the input signal, $\hat{\mathbf{y}}$ is the resynthesized signal, $\mathrm{SC}(\mathbf{y})$ is the time-variant spectral centroid of signal $\mathbf{y}$, $\mathrm{SF}(\mathbf{y})$ is the time-variant spectral flatness of signal $\mathbf{y}$, $\mathrm{TC}(\mathbf{y})$ is the temporal centroid of signal $\mathbf{y}$, and $\mathrm{ERB}(f) = \frac{1000}{24.7 \times 4.37} \log(\frac{4.37f}{1000} + 1)$ is the equivalent rectangular bandwidth (ERB) scale of frequency f, which estimates the auditory scale in human hearing (Glasberg and Moore, 1990).

Training The training is implemented exactly the same as the method described in Section 2.3.1.

2.4 Baseline Methods

2.4.1 Baseline model 1: DDSP method by Engel et al. (2020)

Encoder The detailed structure of the encoder is shown in Section 1.2. When the target audio is fed into the encoder, the MFCCs will be loaded from the pre-processing stage and normalized using the `nn.LayerNorm()` method. The size of the MFCCs vector is (16,250,30). Then it will be passed into a GRU using the `nn.GRU()` method. The input size of GRU is 30 and the hidden layer size is set to 512. Finally, the output of the GRU will be densed, which is implemented by a `nn.Linear()` layer in the code, to obtain the latent embedding $\mathbf{z}$. The output size of the linear layer is set to 16. The "pitch" F0 is obtained using the Python library CREPE. The loudness of the signal will be extracted using the Python library librosa. Those 3 features will be fed into the decoder for the reconstruction of the signal.

Decoder The decoder's structure is shown in Section 1.2. The implementation is almost the same as that in Section 2.3.1. Since the baseline model implements the traditional autoencoder, the latent space $\mathbf{z}$ can be directly obtained by feeding in the MFCCs into the encoder instead of using the sample function.

Loss Function The loss function consists of 2 parts: linear loss and log loss. First, the function `multiscale_fft()` defined in `core.py` is used to calculate the STFT of the synthetic signal and the original signal with a set of scales defined in `config.yaml`. Then, we will loop through the pairs of STFT of original and synthetic signals and calculate the absolute linear distance and log distance between them. The sum of those 2 distances will be the loss of our model.

Training The training process is straightforward. The loss function is fully differentiable, therefore the standard back-propagation can be used to update the loss. In our code, we use `Torch.Tensor.backward()` to update the loss and train our model.

2.4.2 Baseline model 2: DDSP model with timbre

In addition to the original DDSP model, as mentioned in section 2.3.2, we add timbre descriptors to improve the controllability of the whole framework. Therefore, we also consider the DDSP model with timbre descriptors as an alternative baseline. The decoder structure is illustrated in Figure 6. The detailed calculations for the features related to timbre are introduced in section 2.3.2.

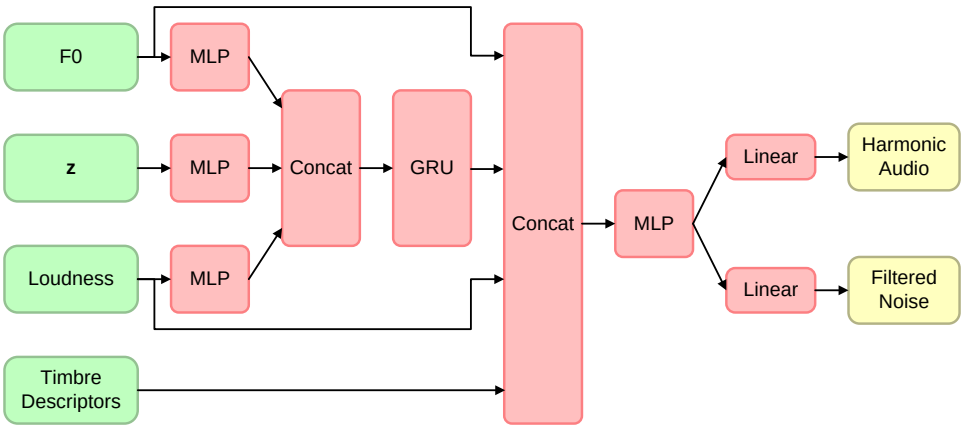

Figure 6: Structure of the proposed decoder with timbre descriptors.

2.5 Evaluation

2.5.1 Evaluation Approach

In keeping up with the aims of our project, we wanted to check if the reconstructed signals were believable enough to originate from the instrument with the associated sound quality specified in the timbre parameters. We designed our approach to check the suitability of the outputs: Qualitatively and Quantitatively.

- Qualitative Evaluation: The resulting signals from our model needed to be of high quality and closely related to the perception of the sound for that specific timbre attribute. Project team members assessed how natural the generated samples sounded to the human ear. For each generated sample, if more than 75% of team members agreed that the sound was natural and associated with the designated timbre qualities, we proceeded with the quantitative evaluation for that sample. It is to be noted that we did not find any case where the members did not agree on the suitability of the generated sounds from the flute samples. (however, for piano sounds, we did come across cases which did not seem believable enough upon hearing which is outside the scope of this project)
- Quantitative Evaluation: Our intention was to check the suitability of the newly proposed method to generate samples sounding like they are from different instruments but based on the existing flute samples processed as input. We examined if the generated samples retained the characteristics of their original source samples. Since, we had made timbre a controllable parameter which is why, we made special consideration for timbre specific features in the evaluation metrics in addition to the sound characteristics. These metrics are described in the following section.
- Splits: We had a total of 6572 samples. They were split as 80% training set, 5% validation set, and 15% test set.

2.5.2 Performance Metrics

We wanted to measure the difference in the sound characteristics of the output samples compared to the source samples. This was done by determining the L_1 distance of the following attributes:

- Fundamental Frequency (“Pitch”): is one of the defining characteristics of sound to the human ear. It is determined by the fundamental frequency. It defines the identity of a musical note and is a fundamental element in music. It is perceived as the highness or lowness of a sound. It is important because the pitch can decide the emotional impact and overall quality of a music sample. The accuracy of pitch affects the melody and harmony of the sound sample. We measure the L_1 distance of pitch to check how good of a match the generated sample is to the original sound in terms of its musical note accuracy.

- Loudness: defines how strong or weak a sound sample may be. It is the energy contained in the sound wave and is defined by the sound's amplitude. The generated sample should have a similar loudness as the source sample so that the dynamics of the audio remain the same. This will ensure that the listener does not feel that the generated sample is too loud or too quiet. We need the difference in loudness to quantify how different the synthesized sound is from the source samples in terms of the intensity of the sound.
- Spectral Centroid (Brightness): This characteristic is influenced by the timbral identity of the sample. It indicates where the center of mass of the spectrum is located for the sample. For a sample, it can affect the perception of brightness or dullness of the sound. For two samples, a higher spectral centroid means a brighter sound.
- Spectral Flatness (Noiseness): This characteristic is influenced by the timbral identity of the sample. It indicates how noise-like a sound is. This means that the higher the value of spectral flatness, the more the sound may seem like white noise as opposed to being more tone-like. A more tonal sound will have more peaks in the spectrum while a sound with more noise will have a flatter spectrum. For our purposes, we use it to measure how closely the output reproduces the tonal or noise-like qualities of the input.
- Temporal Centroid (Energy Distribution): This characteristic is influenced by the timbral identity of the sample. It indicates how the energy of a sound sample is distributed over time. It will influence the dynamics and rhythmic feel of the sample. This is used to compare the timing aspects of the sample.

3 Results

The characteristics mentioned in the section above help to capture different aspects of the sound sample. We examine them separately instead of combining them into a single weighted average because each characteristic highlights a different component of the auditory experience and depending on the context, each may be assigned a different importance so combining them into a single weighted average did not seem appropriate to quantify our measurements. The individualistic comparison of the factors helps us to pinpoint the components for which our approach is performing well.

Model	F0 (ℓ_1)	Loudness (ℓ_1)	SC (ℓ_1)	SF (ℓ_1)	TC (ℓ_1)
Original Autoencoder	5.086	**1.078**	**16.915**	0.191	**0.126**
Proposed VAE	**2.131**	1.440	17.142	**0.168**	0.291
Original Autoencoder with Timbre	**0.925**	**1.230**	**13.852**	**0.133**	**0.154**
Proposed VAE with Timbre	2.167	1.790	14.176	0.134	0.272

Table 1: *Comparison of resynthesis accuracies between the baseline models and the proposed models. F0, SC, SF, and TC represent the fundamental frequency ("pitch"), spectral centroid, spectral flatness, and temporal centroid, respectively. The unit of ℓ_1 distance of "pitch" and the spectral centroid is the musical note, and the unit of ℓ_1 distance of the temporal centroid is second.*

Description of results: First, we will analyze the performance of the VAE model without timbre adjustment in comparison to the baseline approach. The ℓ_1 distance for loudness and spectral flatness is much less with the VAE approach, while other factors are marginally higher. This suggests that the reconstructed sample is much closer to the original sample in terms of pitch and tonal qualities than the baseline approach; however, it is a mixed result because the loudness, brightness, and energy distribution are marginally higher than the original approach. What this means is that the reconstructed sample, though sounds like as produced from a flute but the sample will have slightly more variation in the loudness and intensity of the sound from the original sample belonging to the source instrument. We argue that a lower ℓ_1 distance for the pitch and spectral flatness characteristics are important because they have a greater role in showing that the reconstructed sample is closer to the input sample in terms of the distinct tonal qualities and musical notes identity. We believe this is more important than the differences in the loudness or intensity distribution.

Secondly, with the added timbre parameter to our VAE approach, we find that all the characteristics are marginally greater than the baseline approach with our timbre modification added. It is to be noted that we trained all the models with the same number of steps and computational resources.

Since the VAE with timbre model has the most complicated structure, it may not converge as well as the others.

Applicability of alternative solutions in achieving in our constraints: The original model did not have any timbre-related control thereby not satisfying constraints two and three. Except for constraint one, the baseline approach did not satisfy other constraints.

Description of results and applicability of our approach for the constraints: Only after the reconstructed samples had passed the qualitative evaluation as described were they considered for quantitative evaluation. Both our proposed models: VAE without and with timbre control, generated audio samples that clearly depicted flute sounds as expected, thereby satisfying the first constraint of obtaining outputs of high perceived quality. In terms of the preservation of sound characteristics, the VAE approach without timbre control led to better results as described above. However, in fulfillment of constraints two and three: we were able to create models with controllable features to manipulate the nature of sound for specific timbre control in the latent space with continuous control. The VAE model with our timbre control approach showed a greater difference in the characteristics of the output sample in comparison to the baseline approach model with our timbre control approach added. For both models, our timbre control approach allowed us to add controllable features in the latent space of the samples to manipulate the reconstructed samples which was shown by the timbre-specific evaluation metrics above. An interesting observation is that even though we manipulated the timbre parameters in the latent space, we found that the difference in the timbre-related sound characteristics in the output samples is lesser for both our timbre control-based models in comparison to the non-timbre control-based models.

4 Conclusion

In this project, we developed a deep learning model using Variational Autoencoder (VAE) architecture based on the differentiable digital signal processing (DDSP) framework with controllable features in the latent space for timbre control. Our VAE approach had lower L_1 distance measures in comparison to the original model for pitch and timbre descriptors of the reconstructed audio samples. This means the reconstructed flute samples were closer to the source samples in terms of the sound quality and tonality attributes. As described in the previous section, our approach was able to satisfy the constraints (all three of them) defined for the problem formulation. Our approach was able to incorporate control of timbre attributes which was not present in the baseline or other alternative models in DDSP paper (Engel et al., 2020), thereby, satisfying one of the major goals of this project. However, the performance of the VAE model with the timbre control modality added is not better than the baseline with our timbre modality added. This limitation is likely due to the complex structure of the VAE model. Training with more diverse data, introducing data augmentation, including attention mechanism to condition the VAE for further improve reconstruction quality and control accuracy could be possible directions for future works.

Furthermore, within the scope of the course and the corresponding time and computational constraints, we focused on synthesizing flute samples. We only considered flute samples from the NSynth dataset (Engel et al., 2017) for training our models albeit we included all three types of sources (acoustic, electronic and synthesized). Future works may include multiple types of instruments for building their model with our timbre control based approach. An interesting application of our approach will be to combine it with domain adaptation should the use case arise for handling unlabelled samples of sounds. In our case, we had the labelled samples available as input but in cases where random sound samples may be available without any generator source identified explicitly in the dataset, domain adaptation can be used and our model with timbre control applied to convert the designated output sample.

References

Caillon, A. and P. Esling (2021). RAVE: A variational autoencoder for fast and high-quality neural audio synthesis.

Défossez, A., N. Zeghidour, N. Usunier, L. Bottou, and F. Bach (2018). SING: Symbol-to-instrument neural generator.

Engel, J., K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue, and A. Roberts (2019). GANSynth: Adversarial neural audio synthesis. In *International Conference on Learning Representations*.

Engel, J., L. Hantrakul, C. Gu, and A. Roberts (2020). DDSP: Differentiable digital signal processing. In *2020 International Conference on Learning Representations (ICLR)*.

Engel, J., C. Resnick, A. Roberts, S. Dieleman, D. Eck, K. Simonyan, and M. Norouzi (2017). Neural audio synthesis of musical notes with WaveNet autoencoders.

Glasberg, B. R. and B. C. Moore (1990). Derivation of auditory filter shapes from notched-noise data. *Hearing Research 47*(1-2), 103–138.

Hantrakul, L., J. Engel, A. Roberts, and C. Gu (2019). Fast and flexible neural audio synthesis. In *Proceedings of the 20th International Society for Music Information Retrieval Conference (ISMIR)*, Delft, Netherlands, pp. 524–530. [object Object].

Hayes, B., J. Shier, G. Fazekas, A. McPherson, and C. Saitis (2024). A review of differentiable digital signal processing for music and speech synthesis. *Frontiers in Signal Processing 3*, 1284100.

Kazazis, S., P. Depalle, and S. McAdams (2021). Ordinal scaling of temporal audio descriptors and perceptual significance of attack temporal centroid in timbre spaces. *The Journal of the Acoustical Society of America 150*(5), 3461–3473.

Kim, J. W., J. Salamon, P. Li, and J. P. Bello (2018). CREPE: A convolutional representation for pitch estimation. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Calgary, AB, pp. 161–165. IEEE.

Masuda, N. and D. Saito (2023). Improving semi-supervised differentiable synthesizer sound matching for practical applications. *IEEE/ACM Transactions on Audio, Speech, and Language Processing 31*, 863–875.

McFee, B., M. McVicar, D. Faronbi, I. Roman, M. Gover, S. Balke, S. Seyfarth, A. Malek, C. Raffel, V. Lostanlen, B. Van Niekirk, D. Lee, F. Cwitkowitz, F. Zalkow, O. Nieto, D. Ellis, J. Mason, Kyungyun Lee, B. Steers, E. Halvachs, C. Thomé, F. Robert-Stöter, R. Bittner, Ziyao Wei, A. Weiss, E. Battenberg, Keunwoo Choi, R. Yamamoto, CJ Carr, A. Metsai, S. Sullivan, P. Friesch, Asmitha Krishnakumar, S. Hidaka, S. Kowalik, F. Keller, D. Mazur, A. Chabot-Leclerc, C. Hawthorne, Chandrashekhar Ramaprasad, Myungchul Keum, J. Gomez, W. Monroe, V. A. Morozov, K. Eliasi, Nullmightybofo, P. Biberstein, N. Dorukhan Sergin, R. Hennequin, R. Naktinis, Beantowel, T. Kim, J. P. Åsen, J. Lim, A. Malins, D. Hereñú, S. Van Der Struijk, L. Nickel, J. Wu, Z. Wang, T. Gates, M. Vollrath, A. Sarroff, Xiao-Ming, A. Porter, S. Kranzler, Voodoohop, M. Di Gangi, H. Jinoz, C. Guerrero, Abduttayyeb Mazhar, Toddrme2178, Z. Baratz, A. Kostin, Xinlu Zhuang, Cash TingHin Lo, P. Campr, E. Semeniuc, M. Biswal, Shayenne Moura, P. Brossier, Hojin Lee, and W. Pimenta (2023). Librosa/librosa: 0.10.1.

Peeters, G., B. L. Giordano, P. Susini, N. Misdariis, and S. McAdams (2011). The Timbre Toolbox: Extracting audio descriptors from musical signals. *The Journal of the Acoustical Society of America 130*(5), 2902–2916.

Renault, L., R. Mignot, and A. Roebel (2022). Differentiable piano model for midi-to-audio performance synthesis. In *Proceedings of the 25th International Conference on Digital Audio Effects (DAFx20in22)*, Vienna, Austria, pp. 232–239.

Serra, X. and J. Smith (1990). Spectral modeling synthesis: A sound analysis/synthesis system based on a deterministic plus stochastic decomposition. *Computer Music Journal 14*(4), 12–24.

Wu, Y., E. Manilow, Y. Deng, R. Swavely, K. Kastner, T. Cooijmans, A. Courville, C.-Z. A. Huang, and J. Engel (2022). MIDI-DDSP: Detailed control of musical performance via hierarchical modeling. In *International Conference on Learning Representations*.