Université de Montréal

Controllable Music Performance Synthesis via Hierarchical Modelling

par

Yusong Wu

Département de mathématiques et de statistique
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Informatique

August 25, 2022

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

Controllable Music Performance Synthesis via Hierarchical Modelling

présenté par

Yusong Wu

a été évalué par un jury composé des personnes suivantes :

Ioannis Mitliagkas
(président-rapporteur)

Aaron Courville
(directeur de recherche)

Cheng-Zhi Anna Huang
(codirecteur)

Aishwarya Agrawal
(membre du jury)

Résumé

L'expression musicale requiert le contrôle sur quelles notes sont jouées ainsi que comment elles se jouent. Les synthétiseurs audios conventionnels offrent des contrôles expressifs détaillés, cependant au détriment du réalisme.

La synthèse neuronale en boîte noire des audios et les échantillonneurs concaténatifs sont capables de produire un son réaliste, pourtant, nous avons peu de mécanismes de contrôle.

Dans ce travail, nous introduisons MIDI-DDSP, un modèle hiérarchique des instruments musicaux qui permet tant la synthèse neuronale réaliste des audios que le contrôle sophistiqué de la part des utilisateurs.

À partir des paramètres interprétables de synthèse provenant du traitement différentiable des signaux numériques (Differentiable Digital Signal Processing, DDSP), nous inférons les notes musicales et la propriété de haut niveau de leur performance expressive (telles que le timbre, le vibrato, l'intensité et l'articulation).

Ceci donne naissance à une hiérarchie de trois niveaux (notes, performance, synthèse) qui laisse aux individus la possibilité d'intervenir à chaque niveau, ou d'utiliser la distribution préalable entraînée (notes étant donné performance, synthèse étant donné performance) pour une assistance créative. À l'aide des expériences quantitatives et des tests d'écoute, nous démontrons que cette hiérarchie permet de reconstruire des audios de haute fidélité, de prédire avec précision les attributs de performance d'une séquence de notes, mais aussi de manipuler indépendamment les attributs étant donné la performance. Comme il s'agit d'un système complet, la hiérarchie peut aussi générer des audios réalistes à partir d'une nouvelle séquence de notes.

En utilisant une hiérarchie interprétable avec de multiples niveaux de granularité, MIDI-DDSP ouvre la porte aux outils auxiliaires qui renforce la capacité des individus à travers une grande variété d'expérience musicale.

mots-clés: Synthèse Audio, Modèles Génératif, Hiérarchique, DDSP, Musique, Audio, Modèles Structurés

Abstract

Musical expression requires control of both *what* notes are played, and *how* they are performed. Conventional audio synthesizers provide detailed expressive controls, but at the cost of realism. Black-box neural audio synthesis and concatenative samplers can produce realistic audio, but have few mechanisms for control. In this work, we introduce MIDI-DDSP a hierarchical model of musical instruments that enables both realistic neural audio synthesis and detailed user control. Starting from interpretable Differentiable Digital Signal Processing (DDSP) synthesis parameters, we infer musical notes and high-level properties of their expressive performance (such as timbre, vibrato, dynamics, and articulation). This creates a 3-level hierarchy (notes, performance, synthesis) that affords individuals the option to intervene at each level, or utilize trained priors (performance given notes, synthesis given performance) for creative assistance. Through quantitative experiments and listening tests, we demonstrate that this hierarchy can reconstruct high-fidelity audio, accurately predict performance attributes for a note sequence, independently manipulate the attributes of a given performance, and as a complete system, generate realistic audio from a novel note sequence. By utilizing an interpretable hierarchy, with multiple levels of granularity, MIDI-DDSP opens the door to assistive tools to empower individuals across a diverse range of musical experience.

Keywords: Audio Synthesis, Generative Models, Hierarchical, DDSP, Music, Audio, Structured Models

Contents

List of tables

List of figures

List of acronyms and abbreviations

CNN	Convolutional Neural Network
DDSP	Differentiable Digital Signal Processing
GAN	Generative Adversarial Networks
GRU	Gated Recurrent Unit
LSTM	Long short-term memory
MIDI	Musical Instrument Digital Interface
MLP	Multilayer Perceptron
MSE	Mean Square Error
RMSE	Root Mean Square Error
RNN	Recurrent neural network
STFT	Short-time Fourier transform

Acknowledgement

I am extremely grateful for the amazing set of friends, colleagues and advisors who taught me a lot through my master study and help me come to what I achieved. First, I would like to thank Prof. Aaron Courville and Prof. Cheng-Zhi Anna Huang. I am extremely grateful for the chance you enabled. Thank you for the important lessons you taught me which gave me valuable insights and shaped my research career thus far. I want to especially thank the guidance Prof. Cheng-Zhi Anna Huang in my research career after undergraduate and introduce me the great opportunity of studying at Mila.

I wish to thank all my coauthors. They gave me various ways of support and I learned a ton from them. The paper presented in this thesis can not be done without all the support and help from my coauthors: Ethan Manilow, Yi Deng, Rigel Swavely, Kyle Kastner, Tim Cooijmans, Aaron Courville, Cheng-Zhi Anna Huang, Jesse Engel. Among those and beside my advisors, I would like to especially thank Jesse Engel who kindly offered help and guidance on the research work presented in this thesis as well as skills and tricks for presenting the research work.

I would also like to thank the friends at Mila and beyond for the conversations and discussions in which I often learn things that would not appear on research papers and textbooks. Those I would like to thank include the above as well as but not limited to (alphabetically): Chence Shi, Curtis (Fjord) Hawthorne, Daphne Lafleur, Diganta Misra, Dinghuai Zhang, Evgenii Nikishin, Hugo Sonnery, Ian Simon, Jae Hyun Lim, Josh Gardner, Ke Chen, Mattie Tesfaldet, Max Schwarzer, Minkai Xu, Shengchen Li, Samuel Lavoie, Shujun Han, Tianyu Zhang, Yi Ren, Zhixuan Lin.

I am also extremely grateful for the financial and computational support I received. I would like to thank the financial support from Calcul Quebec, Compute Canada, the Canada Research Chairs, CIFAR and other institutions that provided support that I am not aware of. I want to thank the computational resources provided at Mila, through the Mila cluster and Compute Canada (Calcul Quebec).

Last, I would like to thank my family and friends who supported my research study and research career. I wish to thank my parents, grandparents and other relatives who raised me and introduce this wonderful world to me. I would like to thank my dear partner Yi Deng for her accompaniment and for her support in the most difficult moments in my research and study life. Finally, I want to express my sincere thanks to my late grandfather, who taught me to get interested in the world and science and guided me to research life. I believe he would be proud of me if he knew what I accomplished.

Chapter 1

Introduction

Music is one of the human's oldest art forms and is one that is closely related to technological advances: the manufacturing and electronic technology that creates new music instruments such as electronic guitar; the electronic and computer technology that enables recording and playing equipment which deeply shapes the music industry; the electronic, computer and software improvements that give the world analog and digital synthesizers, effect unit, auto-tune, and digital audio workstation (DAW), etc.

As a quickly advancing technology, deep-learning based artificial intelligence (AI) models are transforming the way people create music. Deep-learning based models are powerful, on account of their capability to extract knowledge and pattern from an ample amount of data. Nowadays, deep-learning based models have exceeded human-level performance in certain tasks and are able to generate realistic data such as text, image, or speech. In the music domain, deep-learning based models show their potential to help people compose music with machine creativity, creating new types of sound and timbre, or synthesizing realistic music audio.

However, current deep-learning based models are awkward for music creation. Most of the models are designed to be "end-to-end": meaning the models directly generate output from input, lacking the interpretability and controllability of the generative process. Just as conductors give detailed suggestions in rehearsal, the music creation process requires controlling the generation process with iterative editing. For example, as presented in Figure 1.1, a holistic process of music creation is often not end-to-end. It is separated into different stages, each with its workflow. For example (third column in Figure 1.1), when composing the song, one would create and edit the chord progression, arrangement, and melody in consecutive steps. An end-to-end generative model would generate a full song in one shot, leaving no room for controlling and adjusting the workflow. A more useful model would generate different steps in the workflow such as chord progression, arrangement, and melody separately while allowing detailed control for each step.

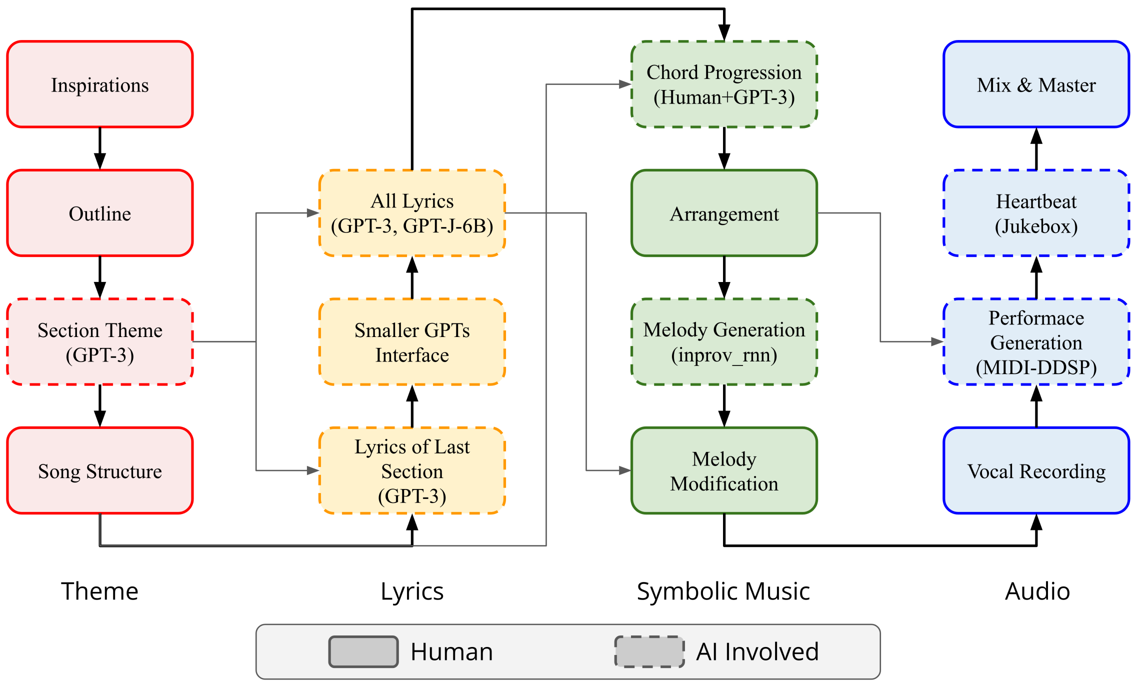


Fig. 1.1. Flowchart of the creation process in the AI Song Contest 2022 entry "A to I" (`https://www.aisongcontest.com/participants-2022/3i`). This flowchart shows an example of how music creation is compose of different stage (present in different color) where each stage has its own workflows (present in each column).

1.1. AI Song Contest: Workflow of Music Creators

The AI Song Contest[1] [59] is an international music competition starting in 2020 for songs that have been composed using artificial intelligence (AI). Teams participating in the AI Song Contest will create a song using AI as part of their songwriting process. AI Song Contest is a good place to observe how people create music, what are their workflow, and how they use the current deep learning-based AI models. The author of this thesis teamed up with two other team members, Yuxuan Wu and Yi Deng, to participate in the AI Song Contest 2022. The entry of their team, "A to I", won 3rd place in the contest[2]. The song "A to I" is about the first-person perspective of an AI model born from chaos, growing up and learning to greet the world, doubts its existence but finally embraces the original intention of humans: to develop more intelligent machines that make humans better lives. In the song, they explore using AI models not only as tools but also as collaborators, song and lyrics writers, performers, storytellers, and even the mentor and first-person narrator. In this section, we will present the creation process of the song "A to I" as a typical music creation workflow to show why controllable approaches are more helpful.

[1]`https://www.aisongcontest.com/`
[2]`https://www.aisongcontest.com/participants-2022/3i`

The division of work in the team follows their complementary background: Yuxuan Wu, with rich experience in music composition and songwriting, compose the song and contributes the solo; Yi Deng, with a background in music and sound engineering, is responsible for using and controlling the models to generate samples, recording the vocal as well as mixing and mastering; the role of Yusong Wu, the author of this thesis is to search and develop tools and models for the other two team members to use, as well as generate samples from those models for the other two team members.

Figure 1.1 shows a flowchart of the creation process. The creation process consists of four stages (theme, lyrics, symbolic music, audio), where each stage has its own detailed workflow. We first work on the outline and theme of the song, then generate all the lyrics using AI models. After that, we start composing the chords, making arrangements (accompaniments), and writing the melody line. Last, we record vocal and generate instrumental accompaniment before applying the final mixing and mastering.

We choose the creation process to be modular as we want to have total control of each step in the song creation, thus ensuring the song is of high quality musically. Inside each step, we also want the model we use to enable detailed control over the generative process. With control, we can accurately realize our creative ideas while saving time in selecting and editing results output by models without control.

To conclude, an ideal generative model for music creators would fit in the creative workflow (modular) and be controllable.

1.2. Problem Definition: Controllable Music Performance Synthesis

In this thesis, we aim to make generative models controllable under the scope of music performance synthesis. Music performance synthesis is a process that commonly generates audio of music instrument playing (or ensemble of music instruments) given music score. The music performance synthesis models are often also referred to as score-to-audio generation models.

A music performance synthesis model, similar to a human player, needs to learn both what to play and how to play. The music score written by the composer, in its symbolic and discrete form, in most cases only indicates notes which consist of what pitch to play, when to play, and how long the note should be playing. Besides rendering the timbre, pitch, and timing of the instrument playing, performance synthesis models need to also infer and render the expressive performance. To perform the score, a human player would interpret these notes through expressive performance: a myriad of nuanced, sub-second choices about articulation, dynamics, and expression. In audio synthesis, those expressive gestures will be realized as short-time pitch and timbre changes of the physical vibration of the instrument.

Thus, the challenge of music performance synthesis is to synthesize audio waveform with realistic timbre, pitch, timing, and expressive performance. Further, the challenge of controlling the music performance synthesis process lies in how to enable control and what to select for control.

1.3. MIDI-DDSP: Controllable Music Performance Synthesis via Hierarchical Modelling

In this thesis, we present our core contribution, a controllable music performance synthesis model named MIDI-DDSP. MIDI-DDSP is a hierarchical generative model of musical performance to provide both realism and control. MIDI-DDSP is built on a similar 3-level hierarchy (notes, performance, synthesis) with interpretable representations at each level (Figure 1.2, left). Compared to conventional methods and previous neural network models, MIDI-DDSP enables both realistic neural audio synthesis and detailed user control (Figure 1.2, right).

Formally, our contributions is as follows:

- We propose MIDI-DDSP, a 3-level hierarchical generative model of music (notes, performance, synthesis), and train a single model capable of realistic audio synthesis for 13 different instruments.
- *Expression Attributes:* We introduce heuristics to extract mid-level per-note expression attributes from low-level synthesis parameters.
- *User Control:* Quantitative studies confirm that manipulating the expression attributes creates a corresponding effect in the synthesizer parameters, and we qualitatively demonstrate the detailed control that is available to users manipulating all three levels of the hierarchy.
- *Assistive Generation:* Reconstruction experiments show that MIDI-DDSP can make assistive predictions at each level of the hierarchy, accurately resynthesizing audio, predicting synthesis parameters from note-wise expression attributes, and autoregressively predicting note-wise expression attributes from a note sequence.
- *Realistic Note Synthesis:* An extensive listening study finds that MIDI-DDSP can synthesize audio from new note sequences (not seen during training) with higher realism than both comparable neural approaches and professional concatenative sampler software.
- *Automatic Music Generation:* We demonstrate that pairing MIDI-DDSP with a pretrained note generation model enables full-stack automatic music generation. As an example, we use Coconet [57] to generate and synthesize novel 4-part Bach chorales for a variety of instruments.

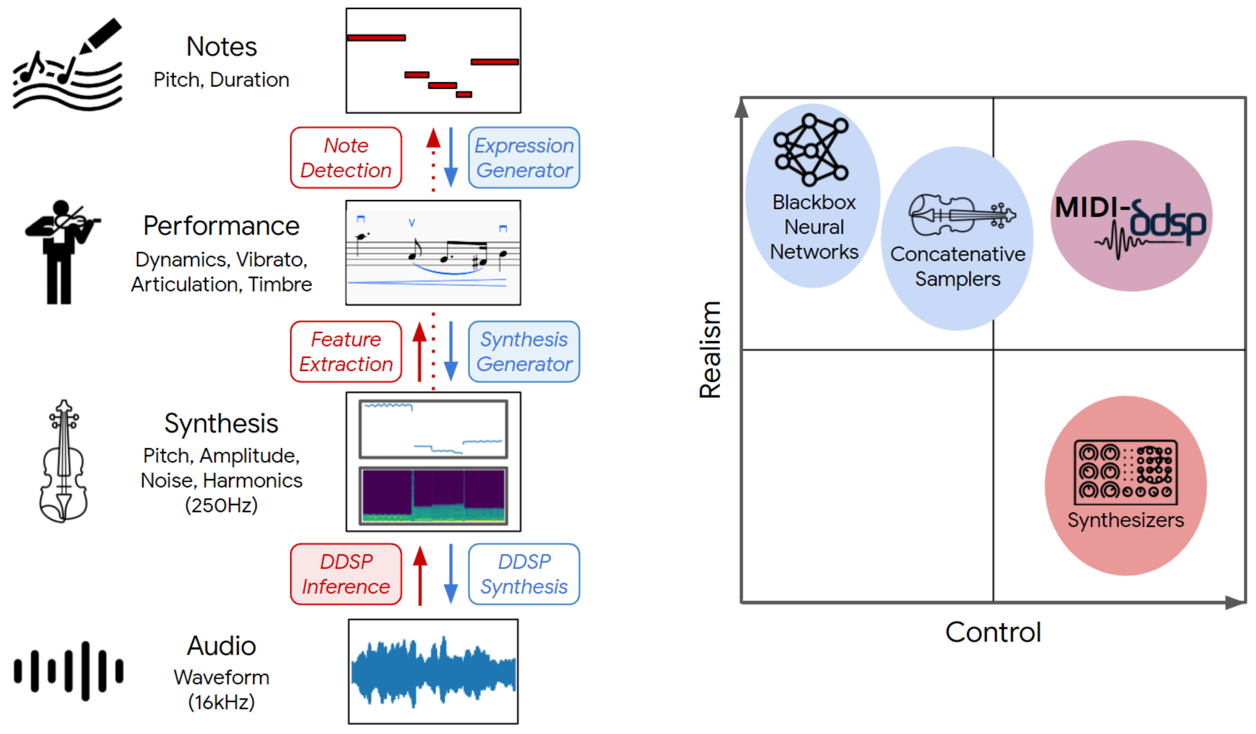


Fig. 1.2. (Left) The MIDI-DDSP architecture. MIDI-DDSP extracts interpretable features at the performance and synthesis levels, building a modeling hierarchy by learning feature generation at each level. Red and blue components indicate encoding and decoding respectively. Shaded boxes represent modules with learned parameters. Both expression features and notes are extracted directly from synthesis parameters. (Right) Synthesizers have wide range of control, but struggle to convey realism. Neural audio synthesis and concatenative samplers can produce realistic audio, but have limited control. MIDI-DDSP enables both realistic neural audio synthesis and detailed user control.

MIDI-DDSP has been shown to be an effective tool for generating and controlling music performance. In our AI Song Contest entry, MIDI-DDSP is used as a tool for rendering accompaniment performance in the later stage of creation (Figure 1.1, third row in the right-most column).

1.4. Resources and Supplementary Materials

- AI Song Contest 2022 entry "A to I": `https://www.aisongcontest.com/participants-2022/3i`
- Audio examples of MIDI-DDSP: `https://midi-ddsp.github.io/`
- Code of MIDI-DDSP: `https://github.com/magenta/midi-ddsp`
- Colab notebook demo of using and controlling MIDI-DDSP: `https://colab.research.google.com/github/magenta/midi-ddsp/blob/main/midi_ddsp/colab/MIDI_DDSP_Demo.ipynb`

1.5. Thesis Layout

Chapter 2 introduces core concepts and backgrounds to understand the paper presented in this thesis, including generative models, music theory, and music generation models. Chapter 3 and 4 present MIDI-DDSP, our contributions toward enabling detailed control in music synthesis models. Chapter 5 includes conclusions drawn from this thesis as well as proposals for future works.

Chapter 2

Background

This chapter will introduce core concepts and basic background to understand the topic of this thesis. Our work lies at the intersection of the domain generative modelling and music generation. In this chapter, we first introduce common types of generative models, and then we introduce basic music theory. Last, we include a section introducing works of music generation using deep learning. We assume the readers have basic knowledge of deep learning, including the basic techniques and terminology such as stochastic gradient descent, regularization, optimization, feed-forward, convolutional, and recurrent neural networks. For readers unfamiliar with deep learning, we recommend reading the book *Deep Learning* [42].

2.1. Deep Generative Models

Generative models are one of the most important families of models in modern deep learning whose goal is to realistically generates data distribution. Formally, given a dataset D from which data $\mathbf{x}$ and optionally target y are sampled. Different from discriminative models which aims to model $p(y \mid \mathbf{x})$, the target of generative models with parameter θ often are modelling $p_\theta(\mathbf{x})$ or $p_\theta(\mathbf{x}, y)$. Modelling directly $p_\theta(\mathbf{x})$ in most cases is more challenging than model $p(y \mid \mathbf{x})$. However, in addition to enabling authentic data generation, learning a good generative model could also capture salient features to help other tasks such as classification, representation learning, semi-supervised learning, denoising, impainting, etc. As generative models aim to learn $p_\theta(\mathbf{x})$, it relieves the need for target y, which is often manually annotated and could effectively leverage a large amount of noisy data without labels. For example, recent advances in generative training (pre-training generative models on a large amount of data) have proven to be an effective technique for models to learn a good representation which benefits down-stream tasks [5, 6, 11, 17, 26, 50].

With recent research advances, deep neural network-based generative models can convincingly generate data such as text, image, speech, music, video, and more. In this section, we introduce the types of generative models commonly used in deep generative models and are essential for understanding the model presented in this thesis. Specifically, we introduce Auto-regressive Models, Variational Autoencoders, Generative Adversarial Networks, Diffusion and Score-based Models. This section is by no mean a complete survey of generative models: we only introduce some of the most popular family of models while not including other such as Flow-based Models [30, 31], Energy Based Models [77, 115] (Restricted Boltzmann Machines [53], Deep Boltzmann Machines [105]), Vector Quantised-Variational AutoEncoder (VQ-VAE) [121] or State-space Models [43, 44]. For each section, we present only a few key papers while omitting other equally important research works. We refer readers to the reference of each method for further understanding.

2.1.1. Auto-regressive Models

Auto-regressive models are a type of directed probabilistic model that estimates the joint distribution by factorization into a chain of conditionals.

$$p_\theta(\mathbf{x}) = \prod_{i=1}^{n} p_\theta(x_i \mid x_{<i}) \tag{2.1.1}$$

where we define $x_{<1} = \emptyset$. For example, consider data as high-dimensional random variable $\mathbf{x} \in \mathbb{R}^d$, an auto-regressive model for such data can learn $p(\mathbf{x})$ as $\prod_{i=1}^{d} p_\theta(x_i \mid x_{<i})$.

Auto-regressive models make no conditional independence of the factors and can be viewed as fully visible Bayes Networks. Unlike other models, generative models can explicitly estimate the likelihood of the sample. In auto-regressive models, usually, the same network and same set of parameters are used to model all the conditional probabilities, such that auto-regressive models have the benefit of saving the parameters to model the joint distribution and enable weight sharing. In some problems, modelling probability condition on infinite past $\prod_{i=1}^{n} p_\theta(x_i \mid x_{<i})$ is computational expensive, whereas a common modification is to condition on fix-length context $\prod_{i=1}^{n} p_\theta(x_i \mid x_{i-1:i-n})$ where n is the context length.

The directed factorization in auto-regressive models is natural for sequential data. Thus auto-regressive models are most common in learning sequential data such as natural language, time series, audio, or music. Although for some data such as image, defining a conditional chain is not salient and will cause a very long chain (e.g., a chain of $64 \times 64 \times 3$ for an RGB image of resolution 64×64), recent studies show that one can learn a good generative model on tasks traditionally hard for auto-regressive models such as image generation [17, 40, 97, 107, 120] or reinforcement learning [16, 98] if the problem can be defined as a conditional chain in a specific order. Some works also propose learning an order-less conditional chain [57, 75].

In dealing with sequential data, Recurrent Neural Networks (RNN) and Transformers are often used to model the conditional probability for their nature to model sequential data. In other cases, Convolutional Neural Networks (CNN) are also used as the back-bone network [107, 119, 120].

The training of auto-regressive models usually uses maximum likelihood estimation on each conditional probability as it is explicit. In inference time, auto-regressive models often sample each factor one at a time according to the order in the conditional chain. Because of sequential sampling, auto-regressive models are often slow to sample from. This is especially the case for auto-regressive models for audio waveform, as waveform consists of tens of thousands of points per second. Some researchers propose various techniques for improving sampling speed in auto-regressive models via parallel sampling [93], output decomposition [131], or more efficient model for faster single-step inference [62].

2.1.2. Variational Autoencoders

The variational autoencoders (VAE) proposed by Kingma and Welling [69], Rezende et al. [102] is a stochastic variational model for efficient approximate inference of the data. Readers are encouraged to read [70] for in-depth introduction of VAE.

VAEs use a latent variable $\mathbf{z}$ to factorize the distribution of data:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) d\mathbf{z}. \tag{2.1.2}$$

Variational autoencoder consists of an encoder (approximate posterior) $q_\phi(\mathbf{z} \mid \mathbf{x})$, a prior $p(\mathbf{z})$ and a decoder (generator) $p_\theta(\mathbf{x} \mid \mathbf{z})$. The aim of the VAE is to maximize the evidence lower bound (ELBO):

$$\log p_\theta(\mathbf{x}) \geq E_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x} \mid \mathbf{z}) - D_{KL}\left[q_\phi(\mathbf{z} \mid \mathbf{x}) \| p_\theta(\mathbf{z})\right], \tag{2.1.3}$$

where the first term on the RHS of the Equation 2.1.3 can be viewed as reconstruction loss, while the second term is the Kullback-Leibler divergence between encoder's output $q_\phi(\mathbf{z} \mid \mathbf{x})$ and the prior distribution $p(\mathbf{z})$.

ϕ and θ are parameters of the encoder network and decoder network, and they are trained with reparameterization tricks using back-propagation. The prior distribution $p(\mathbf{z})$ is often chosen as isotropic Gaussian. Inference in VAEs is very simple: first sample from prior distribution $p(\mathbf{z})$ and then run through decoder $p_\theta(\mathbf{x} \mid \mathbf{z})$ to get data.

VAEs can be used to generate image [18, 69], speech [101], symbolic music [104, 126]. One of the advantages of VAEs is they often learn a good manifold of latent representation through training. Thus VAEs are often used for representation learning. For example, there are works that suggest that variants of VAEs could learn disentangled representation of latent factors in data [52, 88].

Traditionally one of the major shortcomings of VAEs is considered to be generating smoothed data points, such as blurry images. Recent advance in VAE [18] shows that sufficiently deep networks can generate sharp images and outperforms their auto-regressive counterparts.

2.1.3. Generative Adversarial Networks

Generative Adversarial Networks (GAN) [41] is a type of implicit generative model that is based on differentiable generators. GAN consists of a generator network $G(\mathbf{z}, \theta_g)$ that maps a noise prior $p(\mathbf{z})$ to the data space $\mathbf{X}$, and a discriminator network $D(\mathbf{x}, \theta_d)$ that is trained to distinguish samples from the generator distribution $p_{model}(x)$ from real data. GANs aim to use the discriminator to "compete" with the generator and let the generator eventually be able to "fool" the discriminator, thus generating realistic data. Goodfellow et al. [41] formalize this into a two-player zero-sum minimax game with value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]. \tag{2.1.4}$$

In training, the generator and discriminator are updated iteratively. When the minimax game converges and such equilibrium is achieved, we can consider the generator generates samples that cannot be distinguished from the true data distribution. By that time, the discriminator can be discarded, and we can sample data from the generator by inputting prior distribution, which is usually a uniform noise.

Since the GAN was proposed, it has been widely used and has become one type of the most successful generative model algorithms. GANs can generate realistic image [40, 64, 65], audio [74], speech [78], music [33, 37, 128] and video [23, 118].

GANs do not need approximate inference or variational inference and have very little constraint on the architecture of the neural network, thus making them very flexible to implement. Different from auto-regressive models or diffusion models, GANs do not have constraints on modelling in inference time (such as conditional chain and iterative denoising). Thus they often can generate data in parallel, resulting in fast inference speed. This is especially useful when generating sequential data such as audio or music.

Although GANs have various advantage, the original GAN suffers from bad training stability and convergence. To improve convergence of GAN, many works propose new training objective such as Least Squares Generative Adversarial Networks [86] that use least square objective, or Wasserstein Generative Adversarial Networks (WGAN) [2] that use Wasserstein loss and constrain the range of model weights. Also, feature matching [106] is proposed that adds the additional training objective of matching the feature vector of the discriminator between real data and generated data $\left\|\mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} \mathbf{f}(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})} \mathbf{f}(G(\boldsymbol{z}))\right\|_2^2$.

Others propose various network architectures for improving GAN training and generation quality. Those improvements includes using a CNN as the generator network [96], multi-scale generation and discrimination [63], proposing a style-based generator [64].

2.1.4. Diffusion and Score-based Models

Denoising Diffusion Probabilistic Models (DDPM) [54], often referred to as diffusion models, are a family of models which generate data by reversing a Gaussian diffusion process that turns data $\mathbf{x}$ to noise $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ by incrementally adding noise. The forward diffusion process is defined as a Markov chain:

$$q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right)=\mathcal{N}\left(\mathbf{x}_t ; \sqrt{1-\beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I}\right) \quad q\left(\mathbf{x}_{1: T} \mid \mathbf{x}_0\right)=\prod_{t=1}^T q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right), \tag{2.1.5}$$

where $\mathbf{x_0} \sim p(\mathbf{x})$ is the data distribution we want to model. The forward diffusion process is defined such that $T \rightarrow \infty$, $\mathbf{x_T}$ is equivalent to an isotropic Gaussian distribution.

The reverse diffusion process is defined as:

$$p_\theta\left(\mathbf{x}_{0: T}\right)=p\left(\mathbf{x}_T\right) \prod_{t=1}^T p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right) \quad p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)=\mathcal{N}\left(\mathbf{x}_{t-1} ; \boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right), \boldsymbol{\Sigma}_\theta\left(\mathbf{x}_t, t\right)\right). \tag{2.1.6}$$

A model p_θ is learned to estimate the conditional probabilities on the chain to reverse the process. Studies proposed different objectives to train the model, however the common used objective is predicting the added noise given noisy data in certain step on the Markov chain by minimizing an L1 objective:

$$\underset{\mathbf{x}, \mathbf{c}, \epsilon, t}{\mathbb{E}} w_t\left\|\epsilon_\theta\left(\mathbf{x_t}, t\right)-\epsilon\right\|_1^1, \tag{2.1.7}$$

where w_t is a weighting term for loss on different timesteps in the diffusion process. Training in diffusion models consists of first sampling a random timestep from the diffusion process, then adding noise to the original data according to the noise schedule defined in Equation 2.1.6, in the end, run through the model and update network parameters according to the objective. To sample from a diffusion model, one starts from an isotropic Gaussian noise, then run through the backward diffusion process that uses model to gradually denoise and generate the output.

Score-based models [113] are a type of models that are similar to diffusion models. The training objective in score-based models is to match the score function of the distribution $\mathbf{s}_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$. Similar to diffusion models, in inference time, score-based models use Langevin dynamics to draw samples from score function. Ho et al. [54] shows the objective for diffusion models are equivalent to the weighted combination of the objectives used in score-based models.

Despite the need for performing iterative denoising during sampling, diffusion models and score-based models are successful in generating images [27, 91, 114], speech [73], and music [48, 82] with on par or even better generation quality than GANs. The way the modelling and sampling process formed in diffusion and score-based models facilitates inverse problem solving such as inpainting, image colorization, or image super-resolution. It also makes them suitable for guiding by external models who could provide gradients to the output. For example, in text-to-image generation, one could use an image-text contrastive learning model to provide extra guidance to the generation process [91].

2.2. Music

Music is the art of arranging sound [92] and plays a key role in human activities and entertainment. In this section, we introduce the basics of music that are fundamental to understanding the contributions in this thesis. We first introduce basic music theory and notation to represent music. Then, we present the hierarchy present in music. In the next section, we describe common music representation used in computer music and music generation research.

This section is by no means a detailed introduction to music, and we will only cover the basics needed to understand this thesis. For readers interested in this subject, we recommend the book *The Cambridge history of Western music theory* [21].

2.2.1. Music Theory and Notation

Different systems and theories of music exist across different cultures and throughout history. Here, we present the basic theory based on *diatonic scale* and the commonly used music system today in most western music. One of the basic elements in music is the note. The note is a basic unit to describe a sound in music. The basic elements of notes are pitch which denotes the frequency of the sound, and duration, which describes how long the sound lasts. In some cases, a note will also have velocity, which defines how loud the note is played. We will introduce the concepts related to pitch and duration below.

The perceived pitch of a note typically corresponds to the fundamental frequency f_0. In the music system used in this thesis and commonly used today, pitch in music is discrete in frequency space, meaning there is a limited number of music pitches that exist in continuous frequency space. The way in which the pitch is arranged in frequency space is called *tuning system*. The tuning system commonly used is *12 tone equal temperament* which divides the octave (the interval between two notes such that the higher note has double the frequency than the lower note) into 12 parts, all of which are equal on a logarithmic scale. The interval between each part among 12 parts is called a semitone. The *concert pitch* is the reference pitch A4 in 440 Hz. Given the concert pitch, the frequency of the pitch that is i semitones apart from the concert pitch is $f_i = 440 \cdot 2^{i/12}$. The *scale* defines a set of pitches among the 12 semitones in an octave. The scale used in the thesis is *diatonic scale* often written as “C–D–E–F–G–A–B", where the semitone between pitch in scale are arranged as “2–2–1–2–2–2–1". Figure 2.1 shows a C major scale.

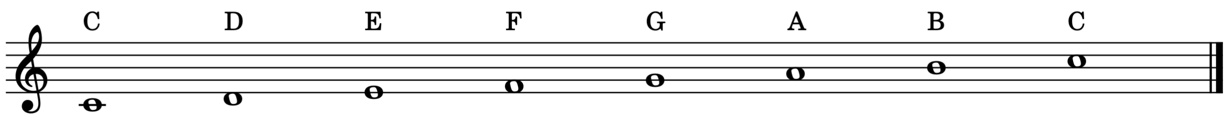


Fig. 2.1. The C major scale.

The length of time for a note to play is determined according to the relative duration of the note represented in note type and the tempo denoted in *Beats Per Minute (BPM)*. The longest note is called the whole note, and the common note types are half note, quarter note, eighth note, and sixteenth note, each occupies $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}$ duration of the whole note. Figure 2.2 shows the common note duration types.

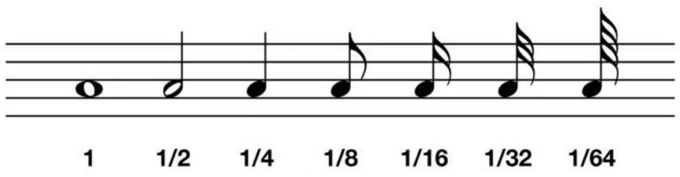


Fig. 2.2. Common note duration types and their staff notation.

The music is often written in a score using staff notation. The staff consists of a set of five horizontal lines and four spaces in between that each represents a different musical pitch. The vertical bars segmenting the staff are called measures, which denote a fixed duration defined by the Time signature. The staff is written to be read in the same order in the western text. Figure 2.3 shows a music score in staff notation.

2.2.2. Music Hierarchy

Just like the hierarchical syntax graph in natural language or the hierarchy of object, substructure, and texture in an image, hierarchy exists in music. This hierarchy is also a consequence of the music creation process: the composer first composes a music piece in the form of a score, and the performers perform their instruments according to the score. The kinetic action of the performer will cause the musical instrument to sound according to its acoustic characteristic, which results in a series of vibrations perceived by humans. The hierarchy in music starts with the piece of music and ends with the vibration in the audio signal that is eventually perceived by humans. A way of dividing music hierarchy can be described as: genre, phrase (a segment of music), motif (short patterns), chord (combination of pitch set), note, expressive performance (the way the music is performed), fundamental frequency and timbre, and audio signal.

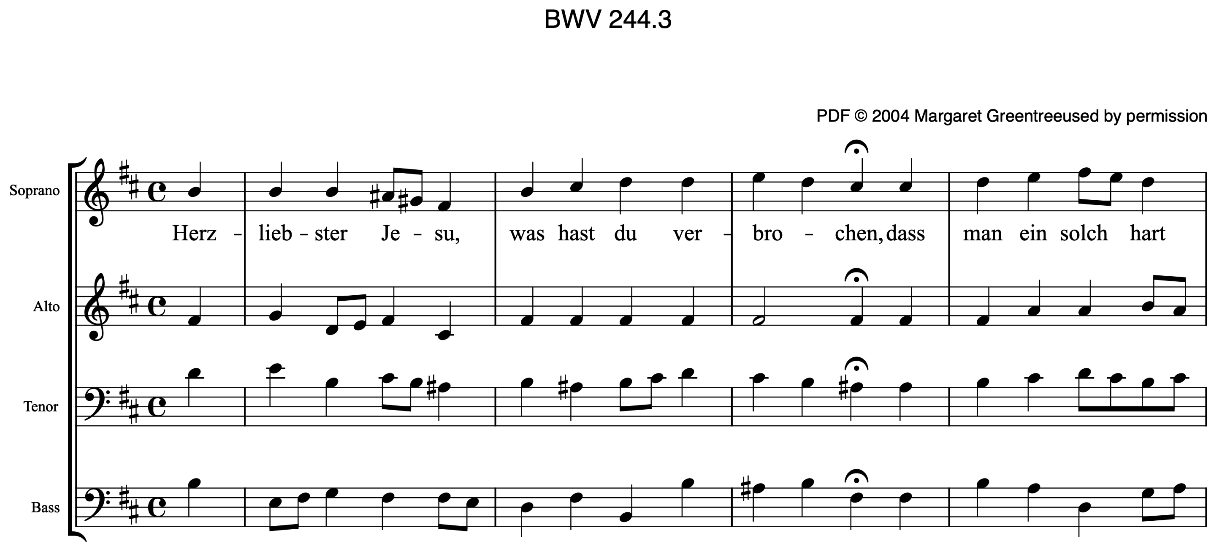

BWV 244.3
PDF © 2004 Margaret Greentreeused by permission
Soprano
Alto
Tenor
Bass
Herz - lieb - ster Je - su, was hast du ver - bro - chen, dass man ein solch hart

Fig. 2.3. An example music score in staff notation.

Figure 2.4 shows a piece of music and the zoom-in for different levels of hierarchy.

The higher level, more abstract part of the music hierarchy, such as genre, phrase, chord, or note, is usually referred to as symbolic music for its discrete nature. The lower level part of the music hierarchy that more close to sound, such as fundamental frequency, timbre, or audio, is often called the acoustic aspect of music.

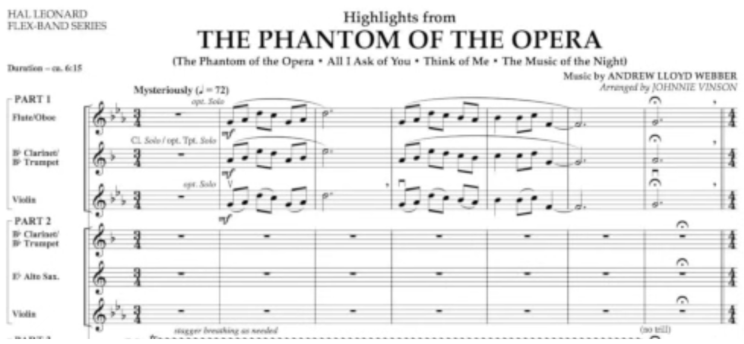
As we introduced above, music has a deep hierarchy containing both symbolic and acoustic essence. Thus, modelling music, especially directly from audio form, is very challenging.

2.3. Music Generation using Deep Learning

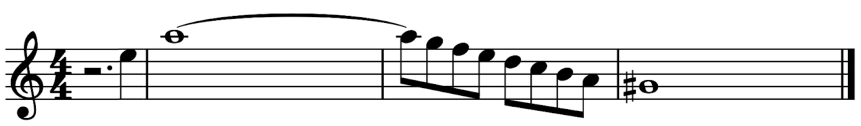
This section will introduce the basics of music generation using deep learning models. We will first describe common music representation used in deep learning models and then introduce music generation works for symbolic and acoustic music. Again, this section is by no means a detailed survey of music generation. Readers are encouraged to [51] and [117] for a more comprehensive survey.

2.3.1. Symbolic Music Representation

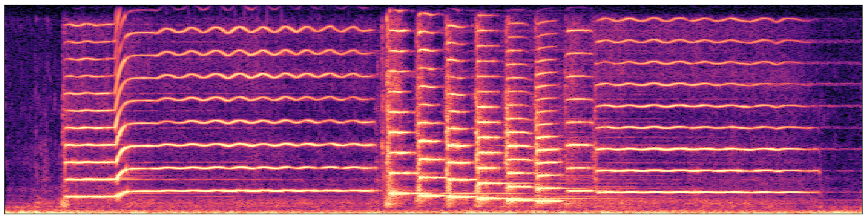
Symbolic music is a sequence where a note is its basic element. Thus, the natural way of representing symbolic music is a sequence of notes where each note is a tuple containing pitch and duration. However, this representation only works in monophonic music, where at most one note is played at a time. In polyphonic music, where multiple notes will play simultaneously and overlap, it is hard to define the order of note sequence tuple.



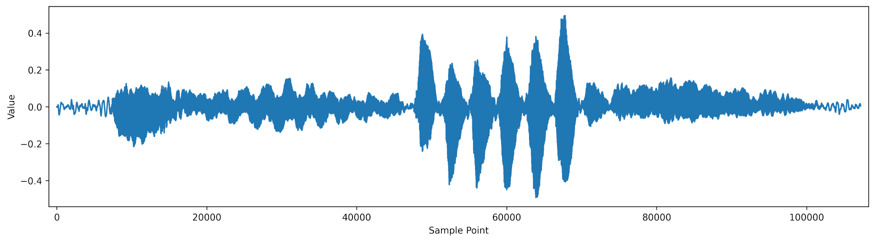
(a)



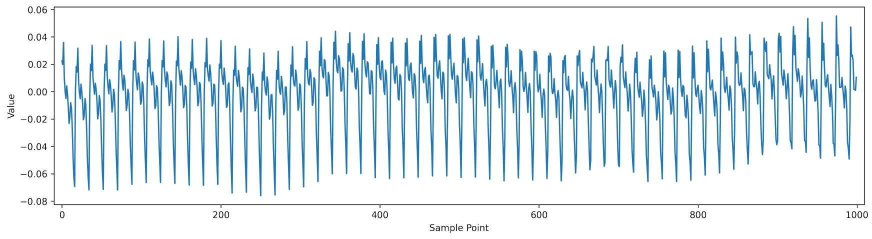
(b)



(c)



(d)



(e)

Fig. 2.4. The music hierarchy of the same music piece "phantom of the opera". From top to bottom: (a) the full score of the piece for symphony orchestra (excerpt from `https://www.youtube.com/watch?v=G4kk88oSv7I`), (b) the score of part of the core melody, (c) the melody of (b) played in violin, visualized as spectrogram, (d) the melody of (b) played in violin, visualized as audio waveform, (e) the zoom-in of sub-structure appears in audio waveform of (d).

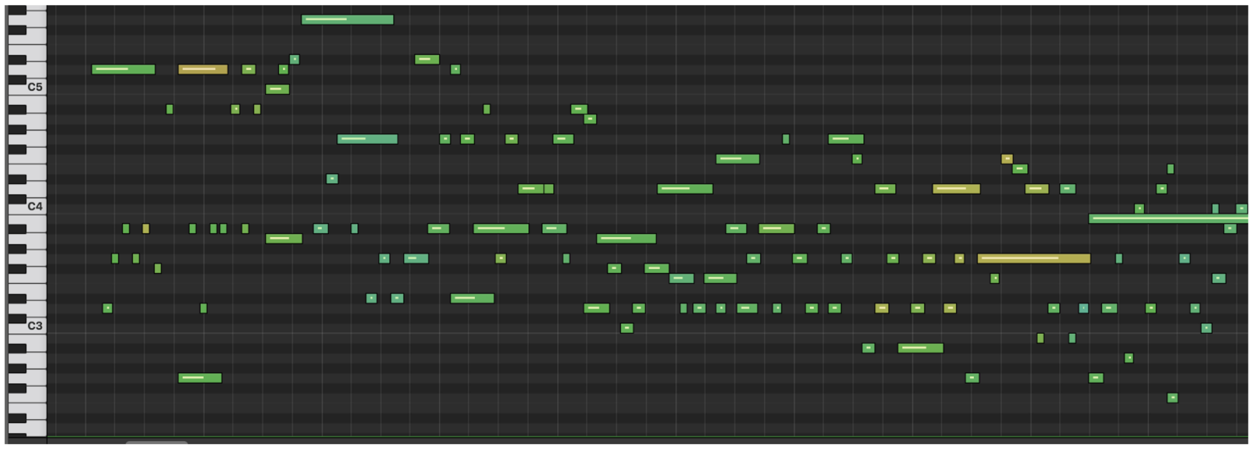


Fig. 2.5. An example music score in pianoroll representation. Figure from `https://magenta.tensorflow.org/performance-rnn`.

In such a case, a natural way of representing symbolic polyphonic music is the pianoroll. Pianoroll gets its name from the roll of paper that is fed into a self-playing piano or music box. In its simplest form can be seen as a binary matrix $\mathbf{x} \in \mathbb{R}^{n \times t}$ where n is the total number of pitches, and t is the number of total timesteps, often quantized to the smallest unit. The element with 1 in pianoroll denotes pitch present in the current timestep and vice versa. Figure 2.5 shows the pianoroll notation.

Although pianoroll is a very intuitive representation, it is not always best for deep learning models to model. The quantization of timesteps often makes the resulting time dimension unnecessarily long, resulting from a small quantization unit. Also, the pianoroll representation does not explicitly distinguish between the onset of the note and the sustain of the note, making it difficult to separate between two consecutive notes in the same pitch and one long note. To mitigate such shortcomings, people turn to use event-based representation [58] which is inspired by the Musical Instrument Digital Interface (MIDI) that represent music as a sequence of event. Event-based representation use `NOTE_ON`, `NOTE_OFF`, `NOTE_SUSTAIN` to denote the boundary of the note as event. The event-based can effectively represent polyphonic music regardless of the total duration of the note, and the resulting discrete sequence is easy for language models to model as a sequence of tokens. However, event-based representation failed to incorporate the vertical relationship between notes (such as notes played simultaneously). Recently, other modifications of event-based representation has been proposed [99]. Figure 2.6 shows the event-based representation.

2.3.2. Acoustic Music Representation

Representations for acoustic music are audio-related representations. Audio is most commonly saved digitally as a signal consisting of a sequence of sample points with a certain sample rate and often is called waveform. In waveform, each sample point represents the

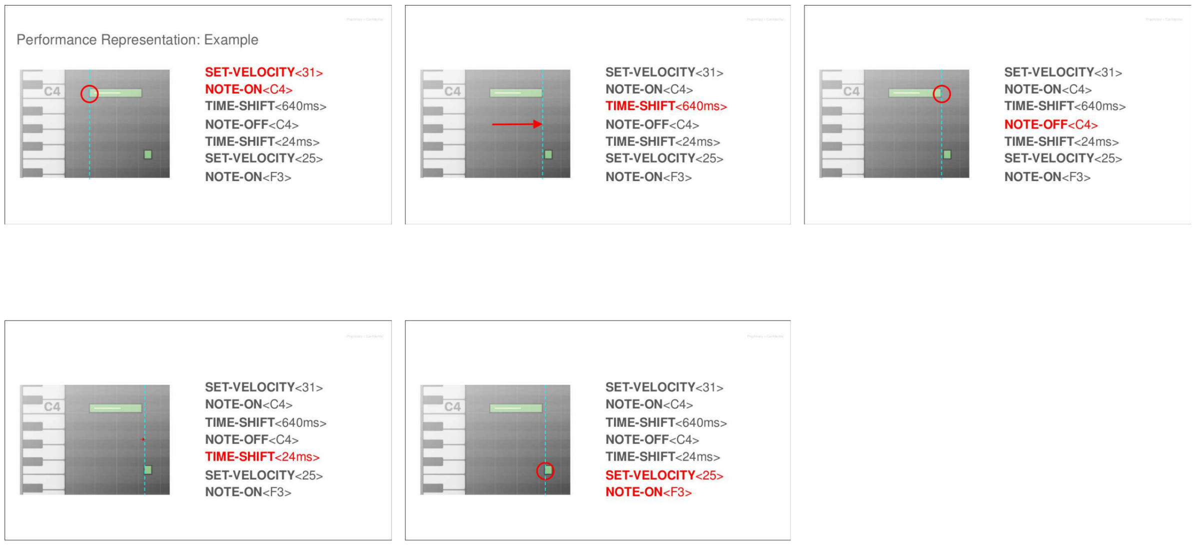


Fig. 2.6. An example music score in event-based representation. Figure from [94].

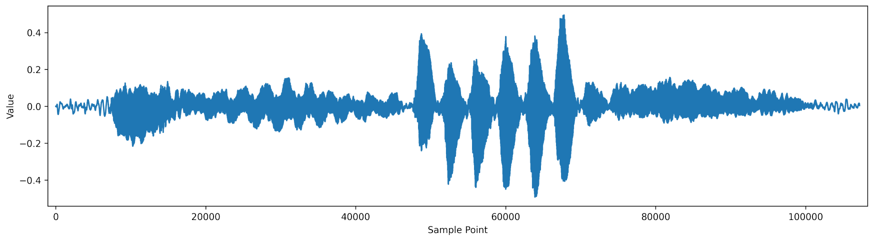
electrical voltage of that time. The commonly used sample rates for recording and processing music audio are 48kHz, 44.1kHz, 24kHz, and 16kHz.

In addition to the waveform, a common way of representing music audio is using spectral-based representation such as Short-time Fourier transform (STFT) or Mel-spectrogram (transformation of frequency in STFT using Mel-scale [116]). Due to the harmonic nature of the music signal, chromagram [36] and constant-Q transform (CQT) are also often used as feature in music information retrieval (MIR) research. Chromagram projects the entire spectrum into 12 bins representing 12 pitch classes, and constant-Q transform (CQT) [108] uses a constant ratio of frequency to resolution, resulting in logarithmically spaced center frequency for each bin and increasing resolution for higher frequencies. Figure 2.7 shows a performance in audio waveform, STFT, Mel-spectrogram, Chromagram, and CQT representations.

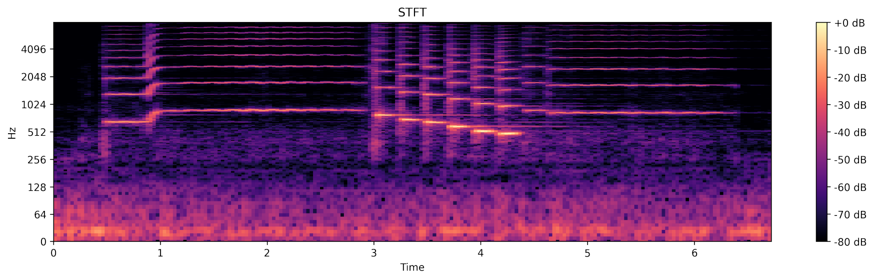
2.3.3. Symbolic Music Generation

A natural way of training symbolic music generation models is to treat symbolic music as a sequence of discrete tokens and apply language models [10, 32, 35, 58, 95]. Others might treat symbolic generation as a conditional generation or conversion task, enabling control of the generative process [12, 19, 22, 83, 104, 126].

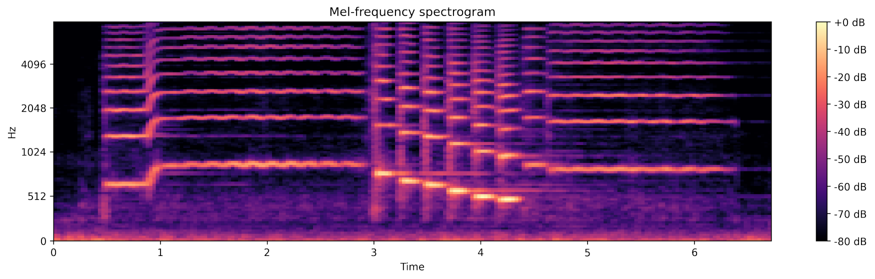
Most symbolic music generation models use sequential models such as RNN or Transformer. In some works, non-sequential models are used, such as GAN with CNN [33] and CNN-based orderless Neural Autoregressive Distribution Estimator (NADE) [57, 75]. Most symbolic generation works use maximum likelihood training, while some approaches propose to train with reinforcement learning [60].



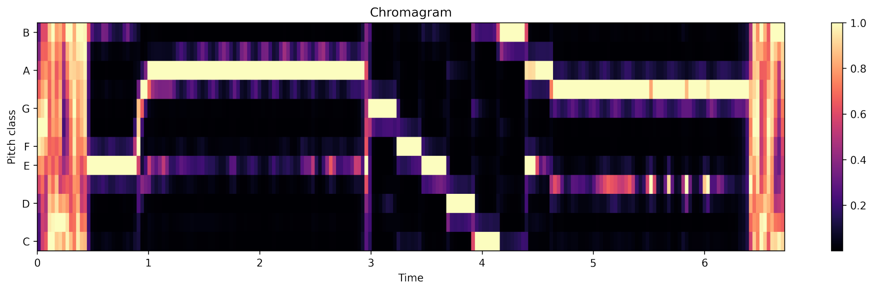
(a)



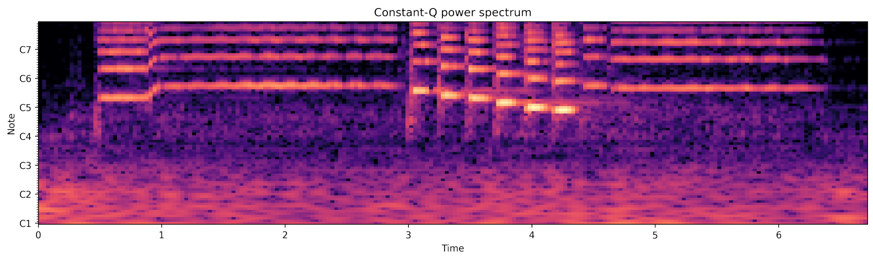
(b)



(c)



(d)



(e)

Fig. 2.7. The acoustic representation commonly used in music generation: (a) waveform, (b) STFT, (c) Mel-spectrogram, (d) Chromagram, (e) CQT.

2.3.4. Acoustic Music Generation

The ultimate output of acoustic music generation is the audio waveform. A type of research is to directly model time dependencies of waveform sample points directly using auto-regressive models and hierarchical models [28, 29, 119]. However, such models often can only generate in an unconditional manner, making them less useful in the music creation process.

Alternatively, score-to-audio generation is an approach to make acoustic music generation controllable via leveraging discrete score or note labels alongside the music recording. Many score-to-audio generation models follow the paradigm of text-to-speech models that first generates a spectrogram and then use another model to inverse spectrogram back to the audio waveform. Various types of generative model are used in spectrogram generation models and spectrogram inversion models, including auto-regressive models [34, 46, 62, 119, 122], GANs [71, 74], or diffusion models [73, 82].

Recently, a separate line of works has been proposed aiming to add in the explicit inductive bias of digital signal processing (DSP). The most representative among those works are Differentiable Digital Signal Processing (DDSP) which proposes to use of additive synthesis, subtractive synthesis, and a reverb module to synthesize audio signal from synthesis parameters. The audio synthesis module in DDSP is made to be fully differentiable, such that neural networks can be trained end-to-end with gradient descent to generate audio. In a similar principle, differentiable wavetable synthesis [111] or waveshaping synthesis [49] can also use as differentiable DSP-based audio synthesizers.

DSP-based audio synthesis benefits from the interpretable DSP parameters and audio synthesis process, which makes it easy to control the audio synthesis process. However, most of those works are proposed to use frame-wise fundamental frequency and loudness contour as input which prohibits them from synthesizing audio from a music score. Several works [14, 61] propose to train an additional network to predict the fundamental frequency and loudness contour and use the pre-trained DDSP model to synthesize audio. Such works leveraging the DDSP model for audio generation inherit the controllability and interpretability from DDSP, enabling them to control the generation to a certain extent. However, the control from DDSP is too fine-grained for the users to have reasonable control on the note-level, and the generation quality is often limited by the pre-trained DDSP model. Wu et al. [128] proposed a hierarchy model that adds an extra performance level modelling that enables detailed control on both note-level and synthesis levels while generating realistic music performance.

Chapter 3

Prologue to Paper

3.1. Paper Details

MIDI-DDSP: Detailed Control of Musical Performance via Hierarchical Modeling, by **Yusong Wu**, Ethan Manilow, Yi Deng, Rigel Swavely, Kyle Kastner, Tim Cooijmans, Aaron Courville, Cheng-Zhi Anna Huang, Jesse Engel. This paper was submitted and accepted for oral presentation at the 2022 *International Conference on Learning Representations* and the 2021 Neurips Workshop *Controllable Generative Modeling in Language and Vision*.

3.2. Contributions

The idea was initially conceptualized by Yusong Wu and Cheng-Zhi Anna Huang. Yusong Wu made the most significant contribution to this paper. In this paper, Yusong Wu is advised by Aaron Courville, Cheng-Zhi (Anna) Huang, Jesse Engel, while helped by other authors by giving advice and helping with experiments. Specifically, the codebase used in this work adapted some of the code Ethan Manilow wrote for his prior work. Ethan Manilow helped extensively by providing helpful advice and helped with paper writing. Yi Deng helped on assessing the user experience of the model and used the model to generate most of the demo samples. Yusong Wu wrote most of the code, designed and conducted most of the experiments, and wrote the paper together with other authors.

3.3. Affiliation

- Yusong Wu, Mila, Université de Montréal
- Ethan Manilow, Northwestern University, Google Brain
- Yi Deng, New York University
- Rigel Swavely, Google Brain
- Kyle Kastner, Mila, Université de Montréal

- Tim Cooijmans, Mila, Université de Montréal
- Aaron Courville, Mila, Université de Montréal
- Cheng-Zhi Anna Huang, Mila, Université de Montréal, Google Brain
- Jesse Engel, Google Brain

Chapter 4

MIDI-DDSP: Detailed Control of Musical Performance via Hierarchical Modeling

4.1. Introduction

[1]Generative models are most useful to creators if they can generate realistic outputs, afford many avenues for control, and easily fit into existing creative workflows [59]. Deep generative models are expressive function approximators, capable of generating realistic samples in many domains [11, 97, 119], but often at the cost of interactivity, restricting users to rigid black-box input-output pairings without interpretable access to the internals of the network. In contrast, structured models chain several stages of interpretable intermediate representations with expressive networks, while still allowing users to interact throughout the hierarchy. For example, these techniques have been especially effective in computer vision and speech, where systems are optimized for both realism and control [15, 79, 90, 100, 123, 132].

For music generation, despite recent progress, current tools still fall short of this ideal (Figure 4.1, right). Deep networks can either generate realistic full-band audio [28] or provide detailed controls of attributes such as pitch, dynamics, and timbre [24, 37, 38, 46, 122] but not both. Many existing workflows use the MIDI specification [3] to Conventional DSP synthesizers [20, 103] provide extensive control but make it difficult to generate realistic instrument timbre, while concatenative samplers [109] play back high-fidelity recordings of isolated musical notes, but require manually stitching together performances with limited control over expression and continuity.

[1]Online resources:
Code: `https://github.com/magenta/midi-ddsp`
Audio Examples: `https://midi-ddsp.github.io/`
Colab Demo: `https://colab.research.google.com/github/magenta/midi-ddsp/blob/main/midi_ddsp/colab/MIDI_DDSP_Demo.ipynb`

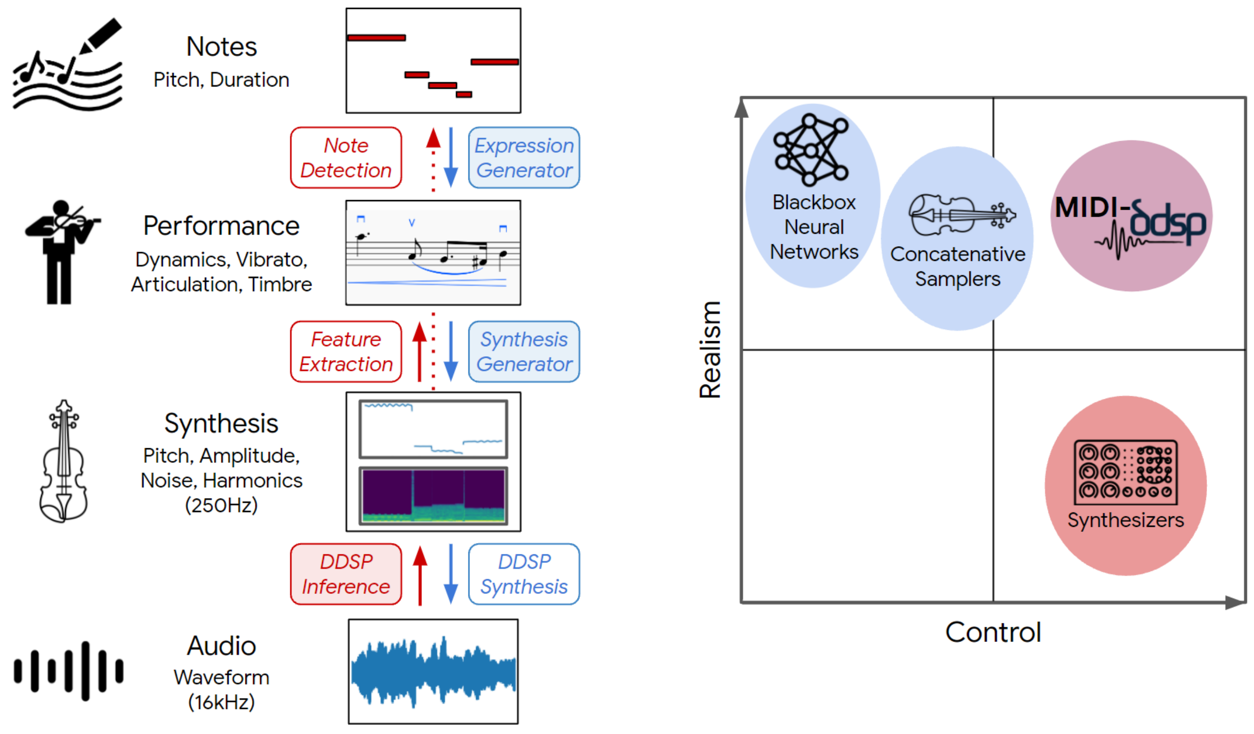


Fig. 4.1. (Left) The MIDI-DDSP architecture. MIDI-DDSP extracts interpretable features at the performance and synthesis levels, building a modeling hierarchy by learning feature generation at each level. Red and blue components indicate encoding and decoding respectively. Shaded boxes represent modules with learned parameters. Both expression features and notes are extracted directly from synthesis parameters. (Right) Synthesizers have wide range of control, but struggle to convey realism. Neural audio synthesis and concatenative samplers can produce realistic audio, but have limited control. MIDI-DDSP enables both realistic neural audio synthesis and detailed user control.

In this paper, we propose MIDI-DDSP, a hierarchical generative model of musical performance to provide both realism and control (Figure 4.1, left). Similar to conventional synthesizers and samplers that use the MIDI standard [3], MIDI-DDSP converts note timing, pitch, and expression information into fine-grained parameter control of DDSP synthesizer modules.

We take inspiration from the hierarchical structure underlying the process of creating music. A composer writes a piece as a series of notes. A performer interprets these notes through a myriad of nuanced, sub-second choices about articulation, dynamics, and expression. These expressive gestures are realized as audio through the short-time pitch and timbre changes of the physical vibration of the instrument. MIDI-DDSP is built on a similar 3-level hierarchy (notes, performance, synthesis) with interpretable representations at each level.

While the efficient DDSP synthesis representation (low-level) allows for high-fidelity audio synthesis [38], users can also control the notes to be played (high-level), and the expression with which they are performed (mid-level). A qualitative example of this is shown in

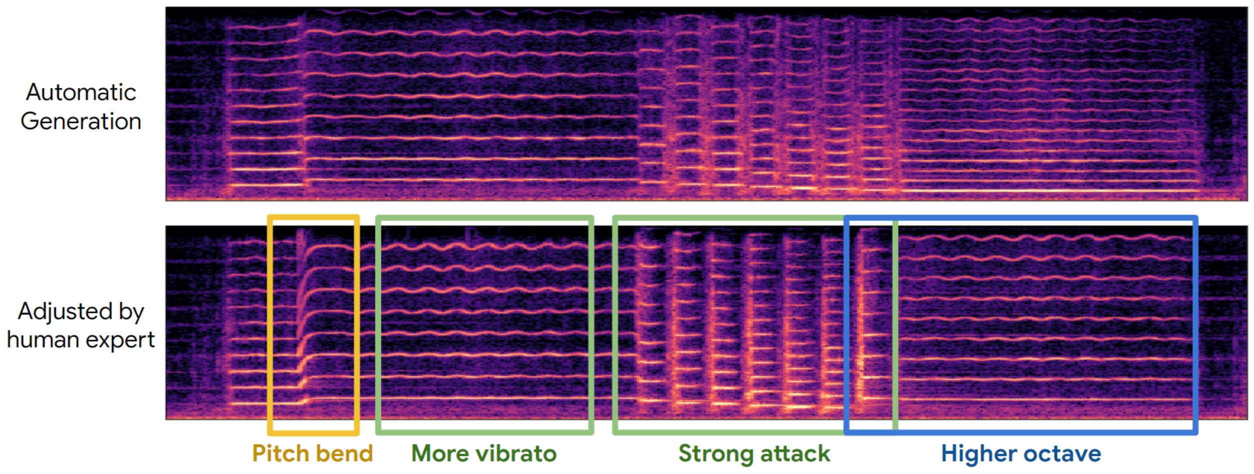


Fig. 4.2. An example of detailed user control. Given an initial generation from the full MIDI-DDSP model (top), an expert musician can adjust notes (blue), performance attributes (green), and low-level synthesis parameters (yellow) to craft a personalized expression of a musical piece (bottom).

Figure 4.2, where a given performance on violin is manipulated at all three levels (notes, expression, synthesis parameters) to create a new realistic yet personalized performance.

As seen in Figure 4.1 (left), MIDI-DDSP can be viewed similarly to a multi-level autoencoder. The hierarchy has three separately trainable modules (*DDSP Inference*, *Synthesis Generator*, *Expression Generator*) and three fixed functions/heuristics (*DDSP Synthesis*, *Feature Extraction*, *Note Detection*). These modules enable MIDI-DDSP to conditionally generate at any level of the hierarchy, providing creative assistance by filling in the details of a performance, synthesizing audio for new note sequences, or even fully automating music generation when paired with a separate note generating model.

It is important to note that the system relies on pitch detection and note detection, so is currently limited to training on recordings of single monophonic instruments. This approach has the potential to be extend to polyphonic recordings via multi-instrument transcription [7, 39, 47] and multi-pitch tracking, which is an exciting avenue to explore for future work. Finally, we also show that each stage can be made conditional on instrument identity, training on 13 separate instruments with a single model.

For clarity, we summarize the core contributions of this work:

- We propose MIDI-DDSP, a 3-level hierarchical generative model of music (notes, performance, synthesis), and train a single model capable of realistic audio synthesis for 13 different instruments. (Section 4.3)
- *Expression Attributes:* We introduce heuristics to extract mid-level per-note expression attributes from low-level synthesis parameters. (Figure 4.5)

- *User Control:* Quantitative studies confirm that manipulating the expression attributes creates a corresponding effect in the synthesizer parameters, and we qualitatively demonstrate the detailed control that is available to users manipulating all three levels of the hierarchy. (Table 4.5 and Figure 4.2)
- *Assistive Generation:* Reconstruction experiments show that MIDI-DDSP can make assistive predictions at each level of the hierarchy, accurately resynthesizing audio, predicting synthesis parameters from note-wise expression attributes, and autoregressively predicting note-wise expression attributes from a note sequence. (Tables 4.4a, 4.4b, 4.4c)
- *Realistic Note Synthesis:* An extensive listening study finds that MIDI-DDSP can synthesize audio from new note sequences (not seen during training) with higher realism than both comparable neural approaches and professional concatenative sampler software. (Figure 4.9)
- *Automatic Music Generation:* We demonstrate that pairing MIDI-DDSP with a pretrained note generation model enables full-stack automatic music generation. As an example, we use Coconet [57] to generate and synthesize novel 4-part Bach chorales for a variety of instruments. (Figure 4.10)

4.2. Related Work

Note Synthesis. Existing neural synthesis models allow either high-level manipulation of note pitch, velocity, and timing [46, 67, 85, 122], or low-level synthesis parameters [9, 14, 61]. MIDI-DDSP connects these two approaches by enabling both high-level note controls and low-level synthesis manipulation in a single system.

Most related to this work is MIDI2Params [14], a hierarchical model that autoregressively predicts frame-wise pitch and loudness contours to drive the original DDSP autoencoder [38]. MIDI-DDSP builds on this work by adding an additional level of hierarchy for the note expression, training a new more accurate DDSP base model, and explicitly modeling the synthesizer coefficients output by that model, rather than the pitch and loudness inputs to the model. We extensively compare to our reimplementation of MIDI2Params as a baseline throughout the paper.

Hierarchical Audio Modelling. Audio waveforms have dependencies over timescales spanning several orders of magnitude, lending themselves to hierarchical modeling. For example, Dieleman et al. [29] and Dhariwal et al. [28] both choose to encode audio as discrete latent codes at different time resolutions, and apply autoregressive models as priors over those codes. MIDI-DDSP applies a similar approach in spirit, but constructs a hierarchy based on semantic musical structure (note, performance, synthesis), allowing interpretable manipulation by users.

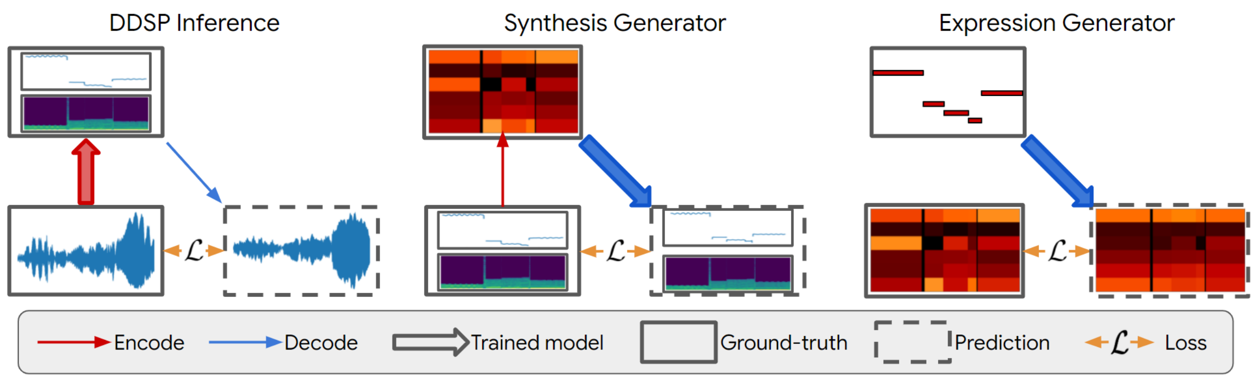


Fig. 4.3. Separate training procedures for the three modules in MIDI-DDSP. (Left) The DDSP Inference module predicts synthesis parameters from audio and is trained via an audio reconstruction loss on the resynthesized audio. (Middle) The Synthesis Generator module predicts synthesis parameters from notes and their expression attributes (shown as a 6-dimensional color map) and is trained via a reconstruction loss and an adversarial loss. (Right) The Expression Generator module autoregressively predicts note expression given a note sequence and is trained with teacher forcing. Encoding processes are shown in red, and decoding processes are shown in blue and loss calculations are shown in yellow. Thicker arrows indicate the process that is being trained in each level. Ground-truth data are shown in solid frames, while model predictions are shown in dashed frames.

Expressive Performance Analysis and Synthesis. Many prior systems pair analysis and synthesis functions to capture expressive performance characteristics [13, 112, 130]. Such methods often use heuristic functions to generate parameters for driving synthesizers or selecting and modifying sample units. MIDI-DDSP similarly uses feature extraction, but each level is paired with a differentiable neural network function that directly learns the mapping to expression and synthesis controls for more realistic audio synthesis.

4.3. Model Architecture

4.3.1. DDSP Synthesis

Differentiable Digital Signal Processing (DDSP) [38] enables differentiable audio synthesis by using a harmonic plus noise model [110]. Briefly, an oscillator bank synthesizes a harmonic signal from a fundamental frequency $f_0(t)$, a base amplitude $a(t)$, and a distribution over harmonic amplitudes $\boldsymbol{h}(t)$, where the dimensionality of $\boldsymbol{h}$ is the number of harmonics. The noise signal is generated by filtering uniform noise with linearly spaced filter banks, where $\boldsymbol{\eta}(t)$ represents the magnitude of noise output from each filter in time. In this study, we use 60 harmonics and 65 noise filter banks, giving 127 total synthesis parameters each time frame ($\boldsymbol{s}(t) = (f_0(t), a(t), \boldsymbol{h}(t), \boldsymbol{\eta}(t))$). The final audio is the addition of harmonic and noise signals.

The harmonic signal is synthesized using a bank of K_h sinusoidal oscillators parameterized by fundamental frequency $f_0(t)$, harmonic amplitudes $a(t)$, and harmonic distribution $\boldsymbol{h}(t)$. The noise signal is synthesized by a filtered noise synthesizer parameterized by noise magnitudes $\boldsymbol{\eta}(t)$. Due to the strong inductive bias introduced by DDSP, the synthesis parameters are highly interpretable, i.e., opposed to some high-dimensional learned latent vector producing audio, with DDSP models the network's output is input to the harmonic plus noise model, whose parameters are interpretable by definition. For the DDSP synthesis used in this work, $\boldsymbol{s}(t) = (f_0(t), a(t), \boldsymbol{h}(t), \boldsymbol{\eta}(t))$, where $f_0(t) \in \mathbb{R}^{1\times t}$, $a(t) \in \mathbb{R}^{1\times t}$, $\boldsymbol{h}(t) \in \mathbb{R}^{60\times t}$, $\boldsymbol{\eta}(t) \in \mathbb{R}^{65\times t}$.

Same as the original DDSP synthesizer, the noise signal is generated by filtering uniform noise given noise magnitudes $\boldsymbol{\eta}(t)$ in K_n frequency bins. An exponential sigmoid nonlinearity is applied to the raw neural network output to generate the $a(t)$, $\boldsymbol{h}(t)$ and $\boldsymbol{\eta}(t)$: exp-sigmoid$(x) = 2.0 \cdot \text{sigmoid}(x)^{\log 10} + \epsilon$, where $\epsilon = 10^{-7}$. The harmonic distribution $\boldsymbol{h}(t)$ is further normalized to constrain amplitudes of each harmonic component: $\sum_{k=0}^{K} h^k(t) = 1$.

In this paper, we use 60 harmonic bins to synthesize harmonic signal, and 65 magnitude bins to synthesize filtered noise. The frame size is set to 64 samples per frame, and the sample rate of the audio synthesis is set to 16000 Hz. After $x(t)$ is synthesized, a reverb module is applied to $x(t)$ to capture essential reverberation in the environment and instrument. The reverb module is implemented as a frequency domain convolution with a learnable impulse response parameter. This paper uses different reverb parameters for different instruments, and the reverb parameters are learned with back-propagation. The learnable impulse response of the reverb is set to have a length of 48000 sample points. In experiments, we found the latter part of the impulse response, which has minimal impact on the timbre and environmental reverb, would cause a very long lingering sound. Thus, in inference time, we add an exponential decay to the impulse response after 16000 samples to constraint the lingering effect:

$$\left.\begin{array}{ll} \mathrm{IR}'(t) = \mathrm{IR}(t), & 0 \leq t \leq 16000 \\ \mathrm{IR}'(t) = \mathrm{IR}(t) \cdot \exp\left(-4(t-16000)\right), & 16000 < t \leq 48000 \end{array}\right\} \qquad (4.3.1)$$

where $\mathrm{IR}(t)$ is the original impulse response, and $\mathrm{IR}'(t)$ is the impulse response after decay.

4.3.2. DDSP Inference

Since the synthesis process is differentiable, Engel et al. [38] demonstrate that it is possible to train a neural network to predict the other synthesis parameters given $f_0(t)$ and the loudness of the audio, and optimize a multi-scale spectral loss [38, 125] of the resynthesized

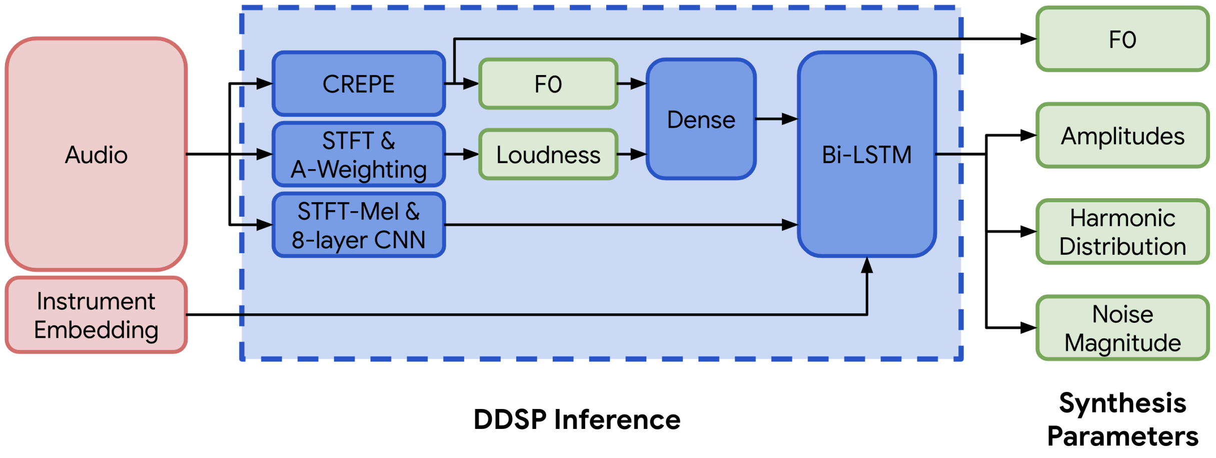


Fig. 4.4. The architecture of the DDSP Inference. The DDSP Inference module extract f_0 and loudness from audio, and use an 8-layer CNN to extract features from Mel-spectrogram. A bi-directional LSTM takes in all features from audio and predict synthesis parameters.

audio (Figure 4.3 left). $f_0(t)$ is extracted by a pre-trained CREPE model [66], and the loudness is extracted via an A-weighting of the power spectrum [45, 89].

We extend this work for our DDSP Inference module, by providing an additional input features extracted by a CNN [76] from log-scale Mel-spectrogram of the audio, that produces higher quality resynthesis (Table 4.4a).

In work proposed by [38], an autoencoder is built to reconstruct audio by predicting synthesis parameters from audio features. We refer to this prior work as the "DDSP autoencoder." Given an audio, fundamental frequency is estimated by CREPE [66] model, and the loudness is extracted via an A-weighting of the power spectrum [45]. Then, the fundamental frequency and loudness are input to a multi-layer perceptron (MLP) respectively. The output of the MLPs are concatenated and passed to a uni-directional GRU. Finally, an MLP is used to predict synthesis parameters from GRU output.

Our DDSP Inference module differs from the above DDSP autoencoder by enabling more information extracted from audio input. The architecture of the DDSP Inference is shown in Figure 4.4. In addition to fundamental frequency estimated by CREPE and loundess extracted via A-weighting of the power spectrum, an 8-layer convolutional neural network (CNN) [76] is used to extract features from log-scale Mel-spectrogram, and a bi-directional long-short term memory network (LSTM) [55] is then applied to extract contextual features. The use of CNN applied on log-scale Mel-spectrogram help model to extract more information from audio input, thus enabling more accurate synthesis parameter estimation.

In our DDSP Inference module, a fully-connected network is applied on fundamental frequency and loudness. The output is concatenated with the output of CNN, send to the bi-directional LSTM to extract contextual features. Another fully connected layer are used to map the features to synthesis parameters. The CNN network in the DDSP Inference module

ConvBlock	Kernel Size	Stride	Filter Size
Conv2d	(3,3)	(1,1)	$K_{Filters}$
Batch norm + ReLU	-	-	-
Conv2d	(3,3)	(1,1)	$K_{Filters}$
Batch norm + ReLU	-	-	-
Average pooling	(1,2)	-	-

Mel-CNN	Output Size	Filter Size	Dropout Rate
LogMelSpectrogram	(1000, 64, 1)	-	-
ConvBlock	(1000, 32, 64)	64	-
Dropout	(1000, 32, 64)	-	0.2
ConvBlock	(1000, 16, 128)	128	-
Dropout	(1000, 16, 128)	-	0.2
ConvBlock	(1000, 8, 256)	256	-
Dropout	(1000, 8, 256)	-	0.2
ConvBlock	(1000, 4, 512)	512	-
Reshape	(1000, 2048)	-	-
Dense	(1000, 256)	256	-

Table 4.1. The architecture of the Mel-CNN (bottom) used to extract features from log-scale Mel-spectrogram in the DDSP Inference module. The Mel-CNN uses convolutinal blocks defined in the first table below.

is similar to the one used by [72] for computational efficiency. The detailed architecture of the CNN is shown in Table 4.1.

The DDSP Inference Module is optimized via Adam optimizer in a batch size of 16 and a learning rate of $3e-4$. We choose a log-scale Mel-spectrogram with 64 frequency dimensions in this work to extract features by CNN. The features extracted by CNN are mapped to 256 dimensions, and concatenated to the features extracted from the fundamental frequency and loudness. When trained in the multi-instrument setting, a 64 dimensional instrument embedding is also concatenated to the aforementioned feature vectors. The bi-directional LSTM has a hidden dimension of 256.

4.3.3. Expression Controls

We aim to model aspects of expressive performance with a continuous variable. For example, this enables a performer to choose *how loud* the note should be performed, or *how much* vibrato to apply. We define a 6-dimensional vector, $\boldsymbol{e}_i$, for each note, $\boldsymbol{n}_i$, where each dimension corresponds to one of the six expression controls, scaled within $[0, 1]$. These are extracted from synthesis parameters $\boldsymbol{s}(t)$ and applied within the ith note, $\boldsymbol{n}_i$, in a note sequence.

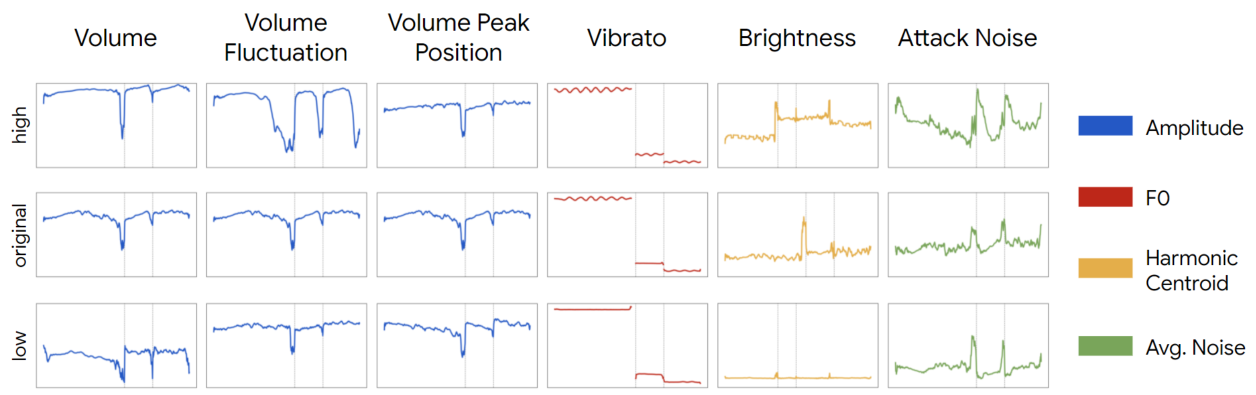


Fig. 4.5. In MIDI-DDSP, manipulating note-level expression can effectively change the synthesis-level quantities. We show by taking a test-set sample (middle row) and adjusting each expression control value to lowest (bottom row) and highest (upper row), how each synthesis quantities (rightmost legend) would change. The dashed gray line in each plot indicates the note boundary.

Specifically, given frame-wise extracted synthesis parameters $\boldsymbol{s}(t)$ for the ith note, $\boldsymbol{n}_i$, in a note sequence, we say that the ith note starts at frame T_{on} and ends at frame T_{off}. We define $\tau \in [T_{\text{on}}, T_{\text{off}}]$ to be every frame that a note is active and the total frame duration of the note is $T_n = T_{\text{off}} - T_{\text{on}}$. The synthesis parameters for a the whole duration of the note are defined by a fundamental frequency $f_0(\tau)$ contour, an amplitude contour $a(\tau)$, a harmonic distribution $\boldsymbol{h}(\tau)$, and a set of noise magnitudes $\boldsymbol{\eta}(\tau)$.

The amplitude contour, $a(\tau)$, and noise magnitudes $\boldsymbol{\eta}(\tau)$ are transformed into dB scale:

$$a'(\tau) = 20 \log_{10} a(\tau), \quad \boldsymbol{\eta}'(\tau) = 20 \log_{10} \boldsymbol{\eta}(\tau) \tag{4.3.2}$$

The note expression controls are extracted as follows:

Volume: Controls the volume of a note, extracted by taking average amplitude over a note. Volume is the sum of the amplitude contour (in dB) normalized over the length of the note, i.e., the mean amplitude over the note,

$$\frac{1}{T_n} \sum_{i=1}^{T_n} a'(i). \tag{4.3.3}$$

Volume fluctuation: Determines the magnitude of a volume change across a note. Used with the volume peak position described below, this can make a note crescendo or decrescendo. This is extracted by calculating the standard deviation of the amplitude over a note (in dB):

$$\sqrt{\frac{1}{T_n} \sum_{i=1}^{T_n} (a'(i) - \overline{a'}(\tau))^2}. \tag{4.3.4}$$

where $\overline{a'}(\tau)$ is the mean amplitude over the whole note.

Volume peak position: Controls where, over the duration of a note, the peak volume occurs. Zero value corresponds to decrescendo notes, whereas one corresponds to crescendo notes. The volume peak position is extracted by calculating the relative position of maximum amplitude in the note:

$$\frac{1}{T_n} \arg\max_i a'(i) \quad \forall i \in [1, T_n]. \tag{4.3.5}$$

Vibrato: Controls the extent of the vibrato of a note. Vibrato is a musical technique defined by pulsating the pitch of a note. Vibrato is extracted by applying Discrete Fourier Transform (DFT) on the fundamental frequency $f_0(t)$ in a note and selecting the peak amplitude. Vibrato is inspired by previous works on expressive performance analysis [81, 87, 130]. The vibrato is calculated by applying Discrete Fourier Transform (DFT) to the fundamental frequency sequence:

$$\max_i \mathcal{F}\{f_0(t)\}_i, \tag{4.3.6}$$

where $\mathcal{F}\{\cdot\}$ defines the DFT function. Only notes with a vibrato rate between 3 to 9 Hz and longer than 200ms are recorded. Otherwise, the vibrato of the note is set to 0. In calculating the DFT function, the fundamental frequency, $f_0(t)$, is zero-padded to 1000 frames.

Brightness: Controls the timbre of a note where higher values correspond to louder high-frequency harmonics. Brightness is determined by calculating the average harmonic centroid (in bin numbers) of a the harmonic distribution in a note:

$$\frac{1}{T_n} \sum_{i}^{T_n} \sum_{k=1}^{|\boldsymbol{h}|} k \cdot \boldsymbol{h}^k(i), \tag{4.3.7}$$

where $\boldsymbol{h}^k(i)$ represents the k-th bin of the harmonic distribution, $\boldsymbol{h}(\tau)$ in the i-th time-step, and we use $|\boldsymbol{h}|$ to refer to the number of bins in the harmonic distribution used by the DDSP module (we use $|\boldsymbol{h}| = 60$, see Section 4.3.2).

Attack Noise: Controls how much noise occurs at the start of the note (the attack), e.g., the fluctuation of string and bow. Attack noise can determine whether two notes sound consecutively or separately. Attack noise is extracted by taking a note's average noise magnitude in the first ten frames (40ms). Many instruments have a high amount of noise in the first few milliseconds of a note (e.g., a bow scraping across a violin string), before the harmonic components are heard. We determine attack noise like

$$\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{|\boldsymbol{\eta}|} \boldsymbol{\eta}'^k(i), \tag{4.3.8}$$

where N determines how many of the first few frames we look at to determine the attack noise (we set $N = 10$, corresponding to 40ms), $\boldsymbol{\eta}'^k(i)$ represents the k-th bin of the dB-scaled noise magnitudes in the i-th time-step, and $|\boldsymbol{\eta}|$ is the number of noise magnitude bins in the DDSP module (set to $|\boldsymbol{\eta}| = 65$, see Section 4.3.2).

Recall that all expression controls are normalized to be in the interval $[0.0, 1.0]$, concatenated to a 6-dimensional vector, and repeated for the full frame duration of the note, T_n, before we use them with the MIDI-DDSP networks. See Sections 4.3.3 and 4.3.4 for additional information. We note that these are not the only ways to determine the "expression" of a note, nor are our definitions definitive. Expression does not even need to be hand designed, and could perhaps be learned in an unsupervised way by neural networks. However, we designed our expression controls because they are all inherently interpretable (compared to black box neural net features). We leave the exploration of other ways to define expression for future work.

For constructing a conditioning sequence from these note expressions and an input note sequence, the frame-wise note expressions and note pitch is expanded to a note-length sequence by repeating these parameters for the number of frames the note occupies. Then, these note expression and pitch parameters are concatenated with note onsets and offsets (a binary flag at the start and end of a note, respectively), and a scalar note positioning code to provide additional information about note boundaries.

4.3.4. Synthesis Generator

Given the output of the per-note Expression Controls, $\boldsymbol{e}_i$ for $i = 1, ..., I$ notes, and a corresponding note sequence, $\boldsymbol{n}_i$, the Synthesis Generator predicts the frame-level synthesis parameters that, in turn, generate audio. Note expression controls are unpooled (repeated) over the duration of the corresponding note to make a conditioning sequence, $\boldsymbol{c}(t) = [(\boldsymbol{e}_1, \boldsymbol{n}_1), ..., (\boldsymbol{e}_I, \boldsymbol{n}_I)]$, with the same length as the fundamental frequency curve, $f_0(t)$.

The Synthesis Generator, g_θ, is an autoregressive recurrent neural net (RNN) is used to predict a fundamental frequency, $\hat{f}_0(t)$ given conditioning sequence, and a convolutional generative adversarial network (GAN), g_ϕ, is used to predict the other synthesis parameters given conditioning sequence and generated fundamental frequency:

$$\hat{f}_0(t) = g_\theta(\boldsymbol{c}(t)), \quad \hat{a}(t), \hat{\boldsymbol{h}}(t), \hat{\boldsymbol{\eta}}(t) = g_\phi(\boldsymbol{c}(t), \hat{f}_0(t)), \tag{4.3.9}$$

where θ denotes trainable parameters in the autoregressive RNN, and ϕ indicates trainable parameters in the GAN. The architecture of the Synthesis Generator is shown in Figure 4.6.

f_0 Generation using the Autoregressive RNN

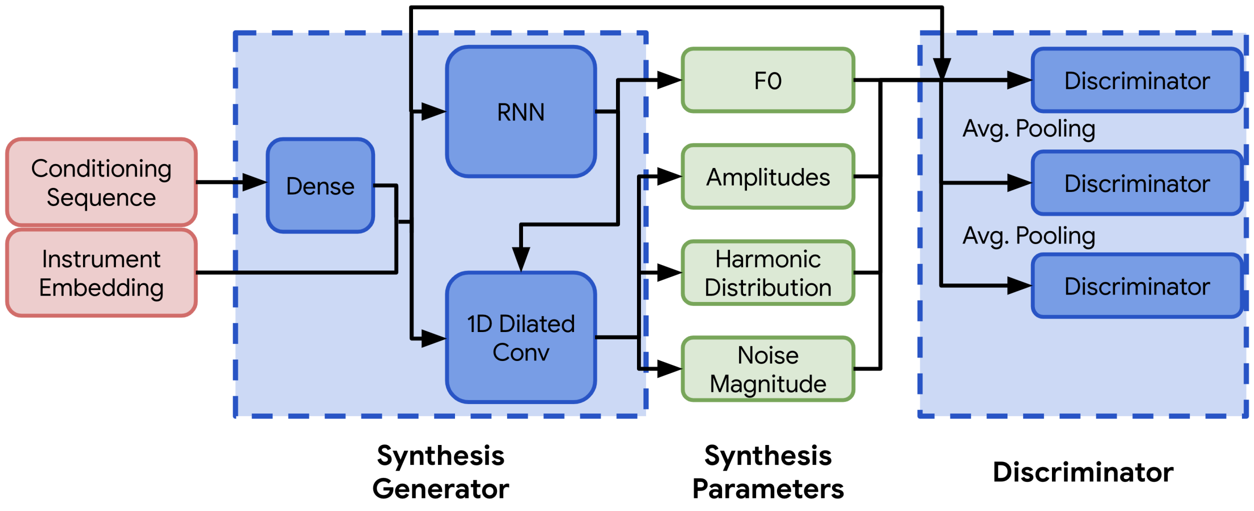


Fig. 4.6. The architecture of the Synthesis Generator. The Synthesis Generator is a GAN whose generator (left) takes in per-note Expression Controls and instrument embedding as a conditioning sequence (red box, left) and produces DDSP synthesis parameters, i.e., f_0, Amplitudes, Harmonic Distribution, and Noise Magnitudes (green boxes, middle). These synthesis parameters get turned into audio by the DDSP modules.

The autoregressive RNN generates an f_0 curve based on a note sequence (e.g., MIDI) and the corresponding Note Expression Controls (described in Section 4.3.3). The autoregressive RNN for f_0 generation consists of a single-layer Bi-LSTM and a 2-layer GRU that autoregressively samples the note's exact pitch, by predicting f_0 frequency offset (in units of semitones) with respect to a known f_0 MIDI note number for the current MIDI note. For example, if at some frame the note specified by the note sequence is A4, which has a MIDI number[2] of $f_0^{\mathrm{A4}} = 69$, and the ground truth f_0 in the audio is $f_0^{\mathrm{GT}} = 69.2$, the autoregressive RNN is expected to predict $f_0^{\mathrm{GT}} - f_0^{\mathrm{A4}} = 0.2$, indicating that the f_0 at the current frame should be 0.2 semitones above the f_0 determined by the current MIDI note, $f_0^{\mathrm{A4}} = 69$. The autoregressive RNN outputs a categorical distribution of pitch offsets in the range of $[-1.00, 1.00]$ semitones, quantized to 201 bins, where each bin represents 0.01 semitone. The final frequency, f_0, input to DDSP synthesizer is converted from semitone units to Hz using $f(n) = 440 \cdot 2^{(n-69)/12}$, where f is the frequency in Hz, and n is the integer MIDI note number. Both the single-layer Bi-LSTM and a 2-layer GRU in the Synthesis Generator have a hidden dimension of 256. The autoregressive RNN is trained using cross-entropy loss $\mathcal{L}_{ce}$. At inference time, the pitch offset is sampled using nucleus sampling [56] with $p = 0.95$ to avoid sudden unrealistic change in fundamental frequency contour. Similar approaches for using autoregressive RNNs to sample f_0 curves has been proposed for speech synthesis and conversion [90, 124].

[2] `https://www.inspiredacoustics.com/en/MIDI_note_numbers_and_center_frequencies`

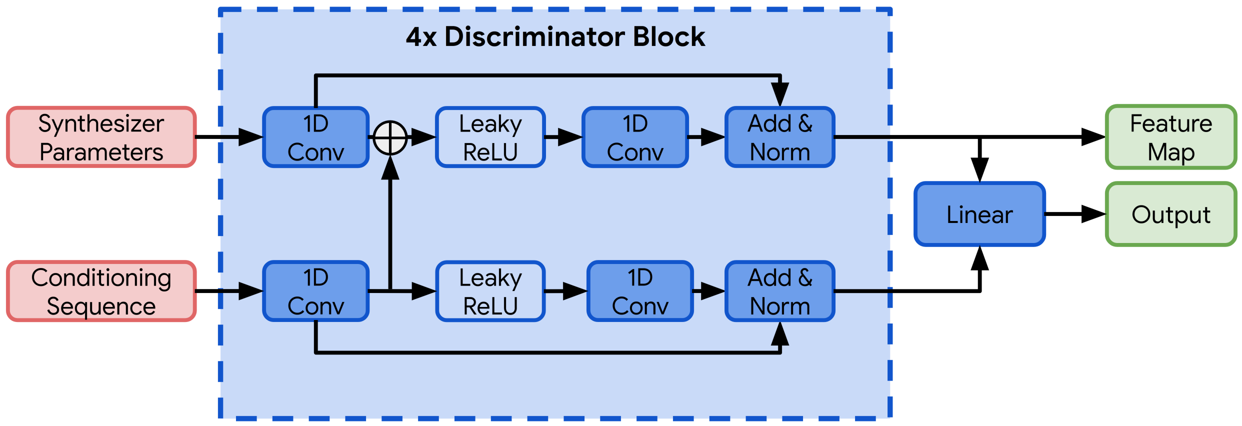


Fig. 4.7. The architecture of the discriminator used in the Synthesis Generator.

Generating the Rest of the Synthesis Parameters

The stack of dilated 1-D convolution layers, *bi* WaveNet [119], is used to generate the rest of synthesis parameters, i.e., the base amplitude for the note, $a(t)$, the harmonic amplitudes, $\boldsymbol{h}(t)$, and the noise magnitudes $\boldsymbol{\eta}(t)$. This network uses the predicted $f_0(t)$, and concatenated expression control vector, $\boldsymbol{c}(t)$, as conditioning. This network consists of 4 convolutional stacks, with each stack having five layers of 1-D convolution with exponentially increasing dilation rate followed by ReLU activation [84] and layer normalization [4]. For clarity, we refer to this network as the "Dilated CNN."

The conditioning sequence input is mapped to a 256-dimensional vector by a linear layer before being sent into an autoregressive RNN and dilated convolution network. If trained on the multi-instrument dataset, a 64-dimension instrument embedding is concatenated after the linear layer. A dropout of rate 0.5 is applied to all GRU units, and during training the input is teacher-forced to avoid overfitting and exposure bias.

The full details of the dilated convolutional network architecture are shown in Table 4.2.

Discriminator for the Synthesis Generator

The architecture of the discriminator used with the Synthesis Generator is shown in Figure 4.7, and the detailed architecture of each discriminator block is shown in Table 4.3. The discriminator is motivated by the multi-scale discriminator network in previous works generating waveforms and spectrograms [74, 78]. It consists of 3 discriminator networks with same architecture that take input signals with different downsample rate in an effort to learn the features of synthesis parameters at different time resolutions. Each discriminator network consists of 4 blocks at each scale, and each block extracts features from predicted synthesis parameters and the conditioning sequence. For each feature stream in a block, two 1-D convolutional layers are used with Leaky-ReLU activation function [129], with skip connections and layer normalization [4].

Dilated Stack	Kernel Size	Dilation Rate	Stride	Filter Size
Conv1d	3	1	1	$K_{Filters}$
ReLU + layer norm	-	-	-	-
Add residual	-	-	-	-
Conv1d	3	2	1	$K_{Filters}$
ReLU + layer norm	-	-	-	-
Add residual	-	-	-	-
Conv1d	3	4	1	$K_{Filters}$
ReLU + layer norm	-	-	-	-
Add residual	-	-	-	-
Conv1d	3	8	1	$K_{Filters}$
ReLU + layer norm	-	-	-	-
Add residual	-	-	-	-
Conv1d	3	16	1	$K_{Filters}$
ReLU + layer norm	-	-	-	-
Add residual	-	-	-	-

Dilated CNN	Output Size	Kernel Size	Dilation Rate	Stride	Filter Size
Conditioning Sequence	(1000, 384)	-	-	-	-
Conv1d	(1000, 128)	3	1	1	128
Dilated Stack	(1000, 128)	3	-	1	128
Dilated Stack	(1000, 128)	3	-	1	128
Dilated Stack	(1000, 128)	3	-	1	128
Dilated Stack	(1000, 128)	3	-	1	128
Layer norm	(1000, 128)	-	-	-	-
Dense	(1000, 126)	-	-	-	126

Table 4.2. The architecture of dilated convolution network (bottom) used in the Synthesis Generator to generate amplitude, harmonic distribution and noise magnitude. The dilated convolution network uses dialted stack layers defined in the first table below.

Synthesis Generator Losses

The Synthesis Generator is trained by minimizing both reconstruction loss and adversarial loss:

$$\mathcal{L} = \mathcal{L}_{recon} + \alpha\mathcal{L}_{adv} = (\mathcal{L}_{CE(f_0)} + \mathcal{L}_{spec}) + \alpha(\mathcal{L}_{lsgan} + \gamma\mathcal{L}_{fm}), \tag{4.3.10}$$

where $\mathcal{L}_{recon}$ is the reconstruction loss, $\mathcal{L}_{adv}$ is the adversarial loss, α and γ are settable hyperparameters that control the overall GAN loss and feature matching loss, respectively. The reconstruction loss, $\mathcal{L}_{recon}$, consists of two pieces: a cross-entropy loss, $\mathcal{L}_{CE(f_0)}$, on the f_0 to train the autoregressive RNN, and a multi-scale spectral loss, $\mathcal{L}_{spec}$ to train the Dilated CNN. The adversarial objective consisting of a least-squares GAN (LSGAN) $\mathcal{L}_{lsgan}$ [86] loss and a feature matching loss $\mathcal{L}_{fm}$, Eqs. 4.3.14-4.3.17) [74].

Discriminator Block - conditioning sequence	Output Size	Kernel Size	Stride	Filter Size
Conditioning Sequence	$(T_{in},\ D_c)$	-	-	-
Conv1d	$(T_{in}/2,\ 256)$	3	2	256
Add output from Discriminator Block of synthesizer parameters	$(T_{in}/2,\ 256)$	-	-	-
Leaky ReLU	$(T_{in}/2,\ 256)$	-	-	-
Add residual and layer norm	$(T_{in}/2,\ 256)$	-	-	-

Discriminator Block - synthesis parameters	Output Size	Kernel Size	Stride	Filter Size
synthesis parameters	$(T_{in},\ D_s)$	-	-	-
Conv1d	$(T_{in}/2,\ 256)$	3	2	256
Leaky ReLU	$(T_{in}/2,\ 256)$	-	-	-
Add residual and layer norm	$(T_{in}/2,\ 256)$	-	-	-

Table 4.3. Details of the discriminator blocks used in the Synthesis Generator.

The cross-entropy loss for the autoregressive RNN is defined as

$$\mathcal{L}_{CE(f_0)} = -\sum_i f_{0i} \log \hat{f}_{0i}, \tag{4.3.11}$$

where i is the length of the entire sequence in frames, f_0 is the ground truth fundamental frequency curve, and $\hat{f}_0$ is the estimated fundamental frequency curve.

The multi-scale spectral loss, $\mathcal{L}_{spec}$, is used for the reconstruction loss. This loss is used for reconstruction in the original DDSP paper [38]. The multi-scale spectral loss computes the L1 difference between the magnitude spectrogram of the predicted and target audio by comparing the computed spectrograms at a number of different FFT sizes. Given the magnitude spectrogram of the predicted audio $\hat{S}_i$ and that of the target audio S_i with FFT size i, the multi-scale spectral loss computes the $L1$ difference between $\hat{S}_i$ and S_i as well as $\log \hat{S}_i$ and $\log S_i$:

$$\mathcal{L}^{(i)}_{spec} = ||S_i - \hat{S}_i||_1 + \beta || \log S_i - \log \hat{S}_i||_1, \tag{4.3.12}$$

$$\mathcal{L}_{spec} = \sum_i \mathcal{L}^{(i)}_{spec} \quad \forall i \in \{2048, 1024, 512, 256, 128, 64\}. \tag{4.3.13}$$

The adversarial loss, $\mathcal{L}_{adv}$, used to train the dilated CNN in the Synthesis Generator. This loss combines a least-squares GAN (LSGAN) [86] objective and a feature matching objective objective [74]:

$$\mathcal{L}_{adv} = \mathcal{L}_{lsgan} + \gamma \mathcal{L}_{fm}. \tag{4.3.14}$$

Given discriminator network D_k in k-th scale, the LSGAN objective to train the Synthesis Generator can be written as

$$\mathcal{L}_{lsgan} = \mathbb{E}_{\boldsymbol{c}} \left[\sum_k ||D_k(\hat{\boldsymbol{s}}, \boldsymbol{c}) - 1||_2 \right], \tag{4.3.15}$$

and the LSGAN objective for training the discriminator is given by

$$\min_{D_k} \mathbb{E} \left[||D_k(\boldsymbol{s}, \boldsymbol{c}) - 1||_2 + ||D_k(\hat{\boldsymbol{s}}, \boldsymbol{c})||_2 \right], \forall k. \tag{4.3.16}$$

In this work, we use 3 scales, so $k = [1, 2, 3]$. Given the output of i-th feature map from one of the 4 layers of the k-th discriminator D_k, the feature map matching objective is calculated as L1 difference between corresponding feature map:

$$\mathcal{L}_{fm} = \mathbb{E}_{\boldsymbol{s},\boldsymbol{c}} \left[\sum_{i=1}^{4} \frac{1}{N_i} ||D_k^{(i)}(\boldsymbol{s}, \boldsymbol{c}) - D_k^{(i)}(\hat{\boldsymbol{s}}, \boldsymbol{c})||_1 \right], \tag{4.3.17}$$

where N_i is the number of units in i-th layer of the feature map.

In training, we use a stop gradient between the adversarial loss and the autoregressive RNN. That is, the RNN is trained by the cross-entropy loss only. The Synthesis Generator is trained using 4 seconds of audio with a frame length of 4ms, resulting in a sequence length of 1000. The Synthesis Generator is optimized via Adam optimizer in a batch size of 16 and a learning rate of 0.0003, with an exponential learning rate decay at a rate of 0.99 per 1000 steps. The discriminator is optimized using Adam optimizer in a batch size of 16 and a learning rate of 0.0001. $\alpha = 1$, $\beta = 1$, and $\gamma = 10$ are used for loss coefficients.

4.3.5. Expression Generator

The Expression Generator uses an autoregressive RNN to predict note expression controls from note sequence. A single-layer bidirectional GRU extracts context information from input, and a two-layer autoregressive GRU generates note expression. The Expression Generator is trained by mean square error (MSE) loss between ground-truth note expression and teacher-forced prediction (Figure 4.3, right). The note sequence used to train the Expression Generator can either be extracted or comes from human labels. To show the full potential of MIDI-DDSP, we use the ground-truth note boundary label from dataset in all experiments for best accuracy. However, in future work note transcription models can be used to provide the note labels.

The Expression Generator creates a set of Expression Controls (see Section 4.3.3) given an input note sequence. The architecture of the Expression Generator is shown in Figure 4.8. The Expression Generator consists of two parts: a single-layer bidirectional GRU

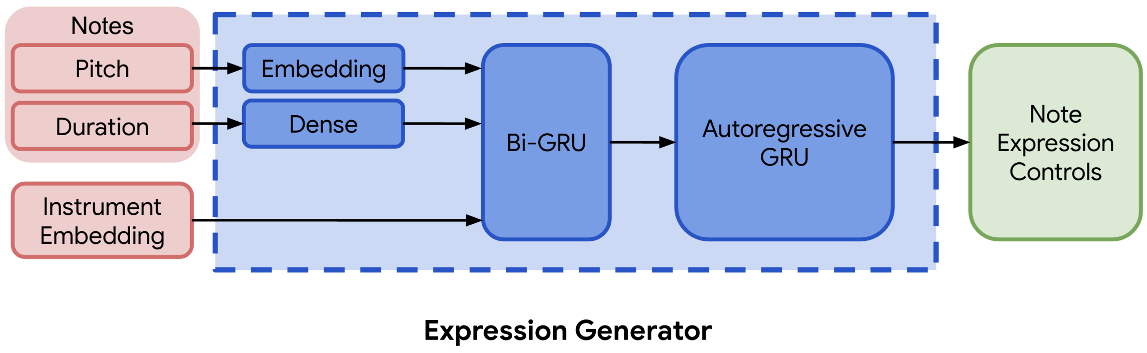


Fig. 4.8. The architecture of the Expression Generator. The Expression Generator consists of two parts: a single-layer bidirectional GRU extracts context information from input, and a two-layer autoregressive GRU generates note expression. The input to the bi-directional GRU is a concatenation of pitch embedding vector, duration feature vector and instrument embedding.

extracts context information from input, and a two-layer autoregressive GRU generates note expression.

In the input to the Expression Generator, the discrete pitch is mapped into a feature vector of 64 dimensions via an embedding layer, and the scalar duration is mapped into a feature vector of 64 dimensions via a fully-connected layer. The instrument embedding input is a 64 dimension feature vector. The input to the bi-directional GRU is a concatenation of pitch embedding vector, duration feature vector and instrument embedding. The bi-directional GRU and auto-regressive GRU all use a hidden size of 128 and a dropout rate of 0.5. Input dropout with a rate of 0.5 is applied to the teacher-forced input in training time to avoid over-fitting and exposure bias. The output layer of the Expression Generator is a two-layer Multi-layer Perceptron (MLP), which consists of a fully connected layer with a layer normalization before the ReLU nonlinearity used in [38].

Data augmentation is applied in the training of the Expression Generator. For the input note sequence, we randomly transpose $\{-3, -2, -1, 0, 1, 2, 3\}$ semitone(s) and randomly stretch the duration by a factor of $\{0.9, 0.95, 1, 1.05, 1.1\}$.

The Expression Generator is trained on a sequence length of 64 notes and a batch size of 256. Adam optimizer [68] is used in training with a learning rate of 0.0001. For training the Expression Generator, we use the mean-square loss (MSE) between the estimated Expression Controls and the ground truth Expression Controls (see Section 4.3.3).

4.4. Experiments

The structured hierarchy and explicit latent representations used in MIDI-DDSP benefit music control as well as music modeling. We design a set of experiments to answer the following questions: First, does the system generate realistic audio, and if so, how does each

Table 4.4. Each module in MIDI-DDSP produces a high-quality reconstruction and accurate prediction. We show reconstruction accuracy of each MIDI-DDSP module against a comparable method.

Model	Spectral Loss
DDSP Inference	**4.28**
Engel et al. [38]	5.00

(a)

Model	RMSE
Synthesis Generator	**0.19**
MIDI2Params	0.26

(b)

Models	RMSE
Expression Generator	**0.14**
MIDI2Params	0.23

(c)

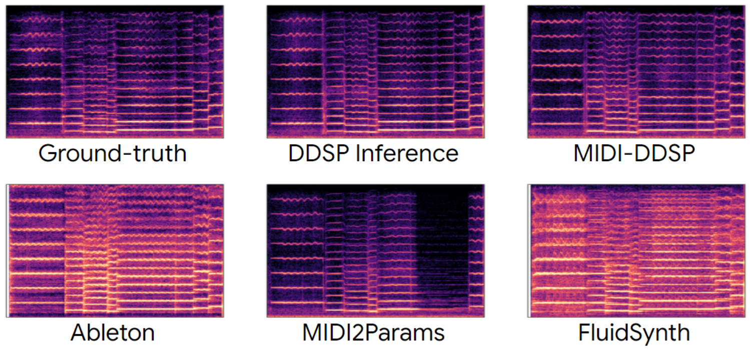
module contribute? How does this compare to existing systems? And, second, is the system capable of enabling note-level, performance-level, and synthesis-level control? How effective are these controls?

4.4.1. Dataset

To demonstrate modeling a variety of instruments, we use the URMP dataset [80], a publicly-available audio dataset containing monophonic solo performances of a variety of instruments. URMP is widely used in music synthesis research [8, 39, 49, 133]. The URMP dataset contains solo performance recordings and ground truth note boundaries from 13 string and wind instruments, which allows us to test MIDI-DDSP on many different instruments. The recordings in the URMP dataset are played by students, and the performance quality is substantially lower compared to virtuoso datasets used in other work [46]. The URMP dataset contains 3.75 hours of 117 unique solo recordings, where 85 recordings in 3 hours are used as the training set, and 35 recordings in 0.75 hours are used as the hold-out test set.

The 13 instruments in the URMP dataset are violin, viola, cello, double bass, flute, oboe, clarinet, saxophone, bassoon, trumpet, horn, trombone, tuba. In this paper, we regard both the soprano saxophone and tenor saxophone in URMP dataset as saxophone. The recordings in the dataset have a sample rate of 48kHz but we downsampled them to 16kHz to match the sample rate of the DDSP synthesis module. To train the Synthesis Generator and DDSP Inference, we segmented the recordings into 4 seconds clips with 50% overlap.

In the URMP dataset, the solo recordings are part of ensemble pieces. Splitting the same piece played by different instruments into training and test sets can cause data leakage. Thus, we split the dataset based on a random shuffle of the recordings, and then moved pieces post-hoc such that the same piece does not appear in both training and test set.



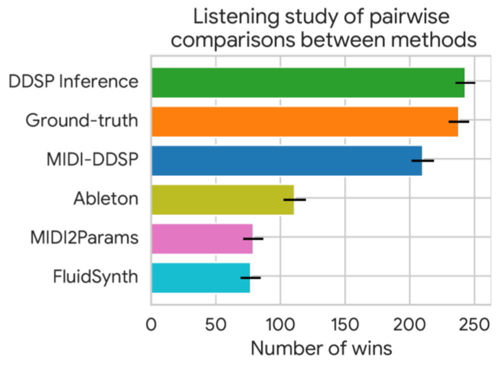


Fig. 4.9. (left) Comparing the log-scale Mel spectrograms of synthesis results from test-set note sequences, MIDI-DDSP synthesizes more realistic audio (more similar to ground-truth and DDSP Inference) than prior score-to-audio work MIDI2Params (enlarged in Figure 4.11). The synthesis quality is also reflected in the listening study (right), where the MIDI-DDSP synthesis is perceived as more realistic than the professional concatenative sampler Ableton and the freely available FluidSynth.

4.4.2. Model Accuracy

Modules in MIDI-DDSP can accurately reconstruct output at multiple levels of the hierarchy (Figure 4.3). We evaluate the reconstruction quality of MIDI-DDSP by evaluating each module, and report the average value across all test set samples in Table 4.4.

DDSP Inference. We measure the difference between reconstruction and ground-truth in the audio spectral loss for our DDSP Inference module and compared it with the original DDSP autoencoder. As shown in Table 4.4a, with an additional CNN to extract features, the DDSP Inference module can reconstruct audio more accurately than the original DDSP Autoencoder.

Synthesis Generator. We predict synthesis parameters from ground-truth note expression and then extract note expression back from the generated synthesis parameters. We measure the root mean square error (RMSE) between note expressions. The prior approach MIDI2Params directly generates synthesis parameters from notes and does not have access to note expressions. However, we can extract note expressions from the generated synthesis parameters and compare them to ground truth. As shown in Table 4.4b, the Synthesis Generator can faithfully reconstruct the input note expression, whereas without access to note expression, MIDI2Params generates larger error.

Expression Generator. We take ground-truth MIDI and evaluate the likelihood of the ground-truth note expressions under the model. As the Expression Generator is autoregressive, we use teacher-forcing to sequentially accumulate the squared error note by note. The total error thus computed can be interpreted as a log-likelihood. We again compare to MIDI2Params, where we autoregressively condition its own output within and on ground-truth across notes to obtain a note-wise metric. That is, MIDI2Params sees the ground truth of past notes, but sees its own output for the current note. As shown in Table 4.4c, the Expression Generator can accurately predict the note expression. In comparison, MIDI2Params without performance-level modeling suffers from predicting the note expression with a higher error when compared to our frame-wise sequence models.

	Volume	Vol. Fluc.	Vol. Peak Pos.	Vibrato	Brightness	Attack Noise
violin	**.99**	**.84**	**.80**	**.86**	**.96**	**.97**
viola	**.98**	**.74**	**.70**	**.82**	**.98**	**.97**
cello	**.97**	.64	.54	**.74**	**.98**	**.94**
double bass	**.98**	**.85**	.34	**.84**	**.99**	**.95**
flute	**.99**	**.87**	.48	.63	**.90**	**.97**
oboe	**.97**	**.79**	**.72**	**.91**	**.87**	**.97**
clarinet	**.98**	**.88**	.63	**.72**	**.88**	**.97**
saxophone	**.97**	**.71**	.43	**.80**	**.94**	**.96**
bassoon	**.99**	**.90**	.56	**.91**	**.99**	**.96**
trumpet	**.97**	**.88**	.55	**.73**	**.92**	**.92**
horn	**.97**	**.88**	.41	**.64**	**.94**	**.95**
trombone	**.97**	**.93**	.59	.52	**.99**	**.96**
tuba	**.98**	**.91**	.15	.22	**.98**	**.93**

Table 4.5. The Pearson correlation result of all instruments described in Table 4.5. The Pearson correlation r-values are shown in the table, while p-values are omitted as in all entries $p < 0.0001$. We consider Pearson r-values larger than 0.7 (marked on **bold**) to mean that our input controls are strongly correlated with the output. For Volume, Brightness, and Attack Noise, all instruments show a strong correlation; for Volume Fluctuation, and Vibrato, the majority of instruments show a strong correlation.

4.4.3. Audio Quality Evaluation by Human Listeners

We evaluate the audio quality of MIDI-DDSP via a listening test. We compare ground truth audio from the URMP dataset to MIDI-DDSP and four other sources: a stripped down version of our system, containing just our DDSP Inference module (Section 4.3.1), MIDI2Params [14], and two concatenative samplers: FluidSynth and Ableton (detailed in Appendix 4.6.2.1). DDSP Inference infers synthesis parameters from the ground truth audio; it serves as an upper bound on what is attainable with MIDI-DDSP, which has to predict expression and synthesis parameters from MIDI. MIDI2Params is prior work that synthesizes audio from MIDI by predicting frame-wise loudness and pitch contour, which is fed as input to a DDSP autoencoder.

Participants in the listening test were presented with two 8-second clips, and asked which clip sounded more like a person playing on a real violin, on a 5-point Likert scale. We collected 960 ratings, with each source involved in 320 pair-wise comparisons. Figure 4.9 shows the number of comparisons in which each source was selected as more realistic. According to a post-hoc analysis using the Wilcoxon signed-rank test with Bonferroni correction (with $p < 0.01/15$), the orderings shown in Figure 4.9 (right) are all statistically significant with the exception of ground truth versus DDSP Inference and MIDI2Params versus FluidSynth (Table 4.6). In particular, MIDI-DDSP was significantly preferred over MIDI2Params, Ableton, and FluidSynth.

The difference among the sources can also be seen from visual inspection of the spectrograms (Figure 4.9, left). While the DDSP Inference module faithfully re-synthesizes the ground-truth audio, MIDI-DDSP generates a coherent performance from a series notes, and has rich, varying expressions across the notes. MIDI2Params failed to generate coherent expressions within a note, generating unrealistic pitch and loudness contours. Also, MIDI2Params stopped the note in the middle when generating the fifth note, suggesting that modelling expressive performance only in synthesis level is limited in long-term coherence even inside a single note. On the contrary, the note expression modeling in MIDI-DDSP allows it to model dependencies at the granularity of the note sequence and use synthesis parameters to model the frame-wise parameter changing inside a single note. The two concatenative synthesizers Ableton and FluidSynth generate the same note expression with identical vibrato and volume for all notes. Although the expression is coherent inside a single note, they fail to generate expression dependencies between different notes automatically.

4.4.4. Effects of Note Expression Controls

To evaluate the behavior of the note expression controls, we measure how well each control correlates with itself after a roundtrip through synthesis. That is, for each sample in the test set, we interpolate the control from lowest (0) to highest (1) in an interval of 0.1 and generate synthesis parameters. Then we extract the note expressions from these synthesis parameters. Table 4.5 reports the correlation between the value we put in and the value observed after synthesis. All controls exhibit strong correlation as desired, except for volume peak position. A low correlation may stem from characteristics of the instrument, or imbalances of those performance techniques in the dataset.

Figure 4.5 illustrates how each note expression affects properties of the sound. For example, as we increase vibrato, we see stronger fluctuations in pitch. Similarly, changing the volume peak position changes the shape of the amplitude curve.

4.4.5. Fine Grained Control or Full End-to-End Generation

The structured modelling approach of MIDI-DDSP enables end users to have as much or as little control over the output as they want. A user can add manipulations at certain levels of the hierarchy or let the model guide the synthesis.

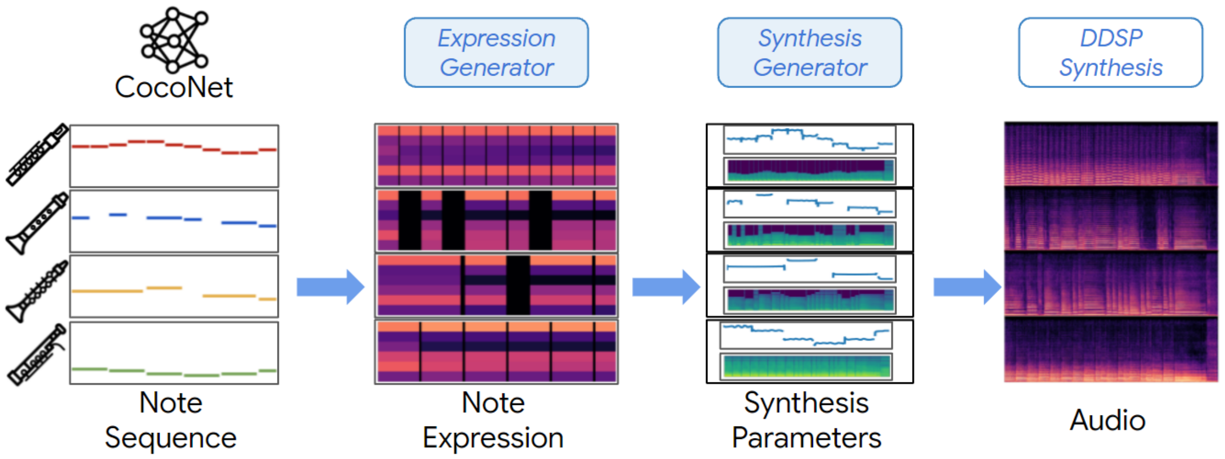


Fig. 4.10. MIDI-DDSP can take input from different sources (human or other models) by designing explicit latent representations at each level. A full hierarchical generative model for music can be constructed by connecting MIDI-DDSP with an automatic composition model. Here, we show MIDI-DDSP taking note input from a score level Bach composition model and automatically synthesizing a Bach quartet by generating explicit latent for each level in the hierarchy.

On one end of this spectrum, Figure 4.2 shows the results of an end user manipulating each level of MIDI-DDSP. Because different levels of the MIDI-DDSP hierarchy correspond with different musical attributes, a user can make manipulations at the note-level to change the attack noise and volume to create staccato notes (second green box in Figure 4.2) or a user could make adjustments to the synthesis-level to control the pitch contour for making a "pitch bend" (yellow box in Figure 4.2).

On the other end of the spectrum, MIDI-DDSP can be paired with generative symbolic music models to make fully generated, realistic end-to-end performances. As shown in Figure 4.10, MIDI-DDSP can be combined with a composition Bach chorales model CoCoNet [57], to form a fully generated musical quartet that sounds like real instruments performance. Readers are encouraged to listen to both the hand-tuned and end-to-end performances on our accompanying website.

4.5. Conclusion

We proposed MIDI-DDSP, a hierarchical music modeling system that factorizes the generation of audio to note, performance, and synthesis levels. By proposing explicit representations for each level alongside modeling note expression, MIDI-DDSP enables effective manipulation and realistic automatic generation of music. We show, experimentally, that the input controls for MIDI-DDSP are correlated with desired performance characteristics (e.g., vibrato, volume, etc). We also show that listeners preferred MIDI-DDSP over existing systems, while enabling fine-grained control of these characteristics. MIDI-DDSP can also connect to other models to construct a full audio generation model, where beginners can obtain realistic novel music from scratch, while expert users can manipulate results based on model prediction to realize unique musical design.

4.6. Appendix

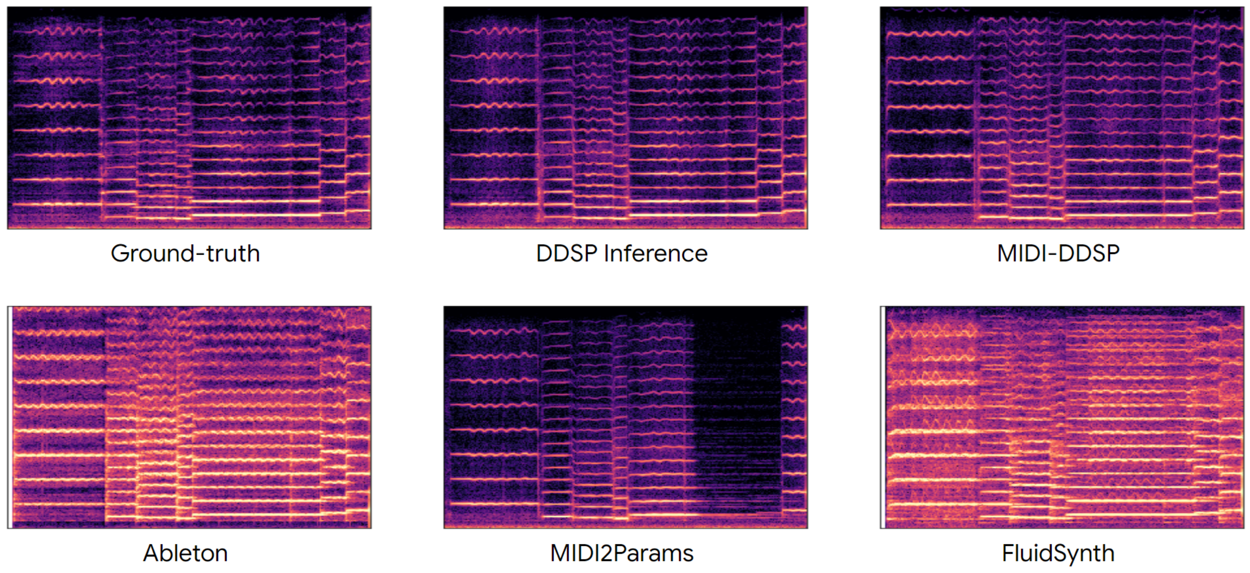


Fig. 4.11. The enlarged log-scale Mel spectrograms of synthesis results in Figure 4.9

4.6.1. Other Training Details

The Expression Generator, Synthesis Generator, and DDSP Inference Module are trained separately. The Expression Generator is trained for 5000 steps, the Synthesis Generator is trained for 40000 steps, and the DDSP Inference Module is trained for 10000 steps.

4.6.2. Experiment Details

4.6.2.1. Details of Baseline Methods. We use the orchestral strings pack in Ableton[3] without any manual adjustment to synthesize the audio from Ableton. For FluidSynth, we used the `FluidR3_GM.sf2`[4] sound font. We reimplemented the MIDI2Param system proposed by [14] by following the official implementation on GitHub[5]. The MIDI2Params system is only trained in the single instrument setting for the listening test experiments as is done in the original paper. However, as a comparison in our multi-instrument scenario, we trained a multi-instrument MIDI2Params model that had an additional instrument embedding concatenated to the model input.

4.6.2.2. Details of the Listening Test. In total, we gathered 960 ratings based on 360 pairwise comparisons from 14 participants for our listening test. Participants were presented with two unlabeled audio clips and asked "Which one of the following recordings sound most

[3] `https://www.ableton.com/en/packs/orchestral-strings/`
[4] `https://member.keymusician.com/Member/FluidR3_GM/index.html`
[5] `https://github.com/rodrigo-castellon/midi2params`

Pairs		wins	ties	losses	p value
Ableton	MIDI2Params	39	1	24	7.61e-5
Ableton	DDSP Inference	11	0	53	2.98e-27
Ableton	FluidSynth	42	0	22	0.0002
Ableton	Ground-truth	14	1	49	6.62e-26
Ableton	MIDI-DDSP	5	0	59	5.34e-16
MIDI2Params	DDSP Inference	8	0	56	5.73e-42
MIDI2Params	FluidSynth	31	0	33	0.027*
MIDI2Params	Ground-truth	8	0	56	2.43e-40
MIDI2Params	MIDI-DDSP	8	0	56	7.16e-28
DDSP Inference	FluidSynth	55	0	9	2.97e-39
DDSP Inference	Ground-truth	35	0	29	0.024*
DDSP Inference	MIDI-DDSP	44	0	20	6.02e-05
FluidSynth	Ground-truth	4	0	60	1.00e-37
FluidSynth	MIDI-DDSP	9	0	55	7.25e-26
Ground-truth	MIDI-DDSP	44	0	20	0.00018

Table 4.6. A post-hoc comparison of each pair on their pairwise comparisons with each other, using the Wilcoxon signed-rank test for matched samples ("win" means the first item in the pair is selected) . p value less than $0.01/15 = 6.67\times10^{-4}$ yields a statistically significant difference. Only two pairs are not significantly different (DDSP Inference vs. Ground-truth, MIDI2Params vs. FluidSynth), and are marked with an asterisk (*).

like a person playing it on a real violin?" Participants were asked to wear headphones, and were able to listen to the audio clips as many times as they pleased before submitting their answers. The participants chose their preference using a 5-point Likert-like scale, with the first option being "Strong preference for Audio Clip 1", the middle option indicating "No preference", and the final option being "Strong preference for Audio Clip 2."

The pairs of unlabeled clips were drawn from the set of [Ground-truth, DDSP Inference, MIDI-DDSP, Ableton, MIDI2Params, Fluidsynth]. Participants were recruited through a Google internal data labeling platform, and were selected based on a pre-screening pilot study to ensure that the participant was able to provide reasonable evaluations of audio recordings. In this pilot study, we filtered out individuals who rated FluidSynth examples as sounding more like a real violin recording than the Ground-truth violin recording. Participants were not screened based on their musical background.

A Kruskal-Wallis H test of the ratings showed that there is at least one statistically significant difference between the models: $\chi^2(2) = 395.35$, $p < 0.01$. The number of wins for each pair comparison and a Wilcoxon signed-rank test for each pair is shown in Table 4.6.

4.6.3. Acknowledgements and Open-Source Image Attributions

We would like to thank Yi Ren for the discussion on Generative Adversarial Nets for audio generation. We would like to thank Shujun Han for providing advice on making and arranging figures in the paper. We would also like to thank Mark Cartwright for his feedback on the paper and fruitful discussions about it.

4.7. Open-Source Image Attribution

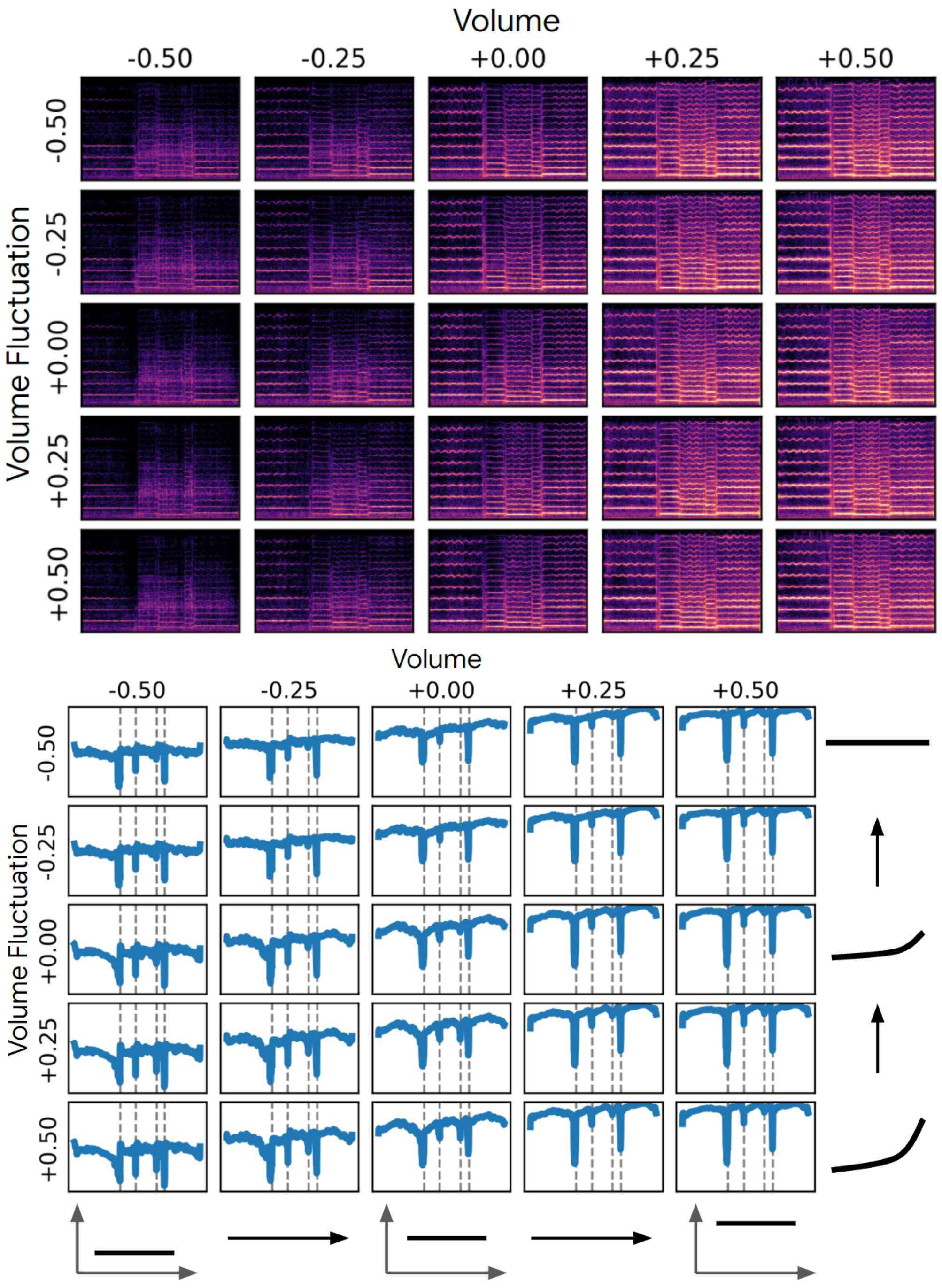


Fig. 4.12. The effect of modifying the 'Volume' and 'Volume Fluctuation' note expression for a sample. Each row shows the amplitudes of the notes when fixing 'Volume Fluctuation' and changing 'Volume', while each column shows the amplitudes of the notes when fixing 'Volume' and changing 'Volume Fluctuation'. The number indicates the amount of modification to the note expression control where (+0, +0) is the original sample. Upper figure shows the spectrogram of generations, and the bottom figure shows the amplitude of the generations. The cartoon on the side indicates how the modified feature would change the synthesis parameters. The gray dash line in the bottom figure indicates the note boundaries.

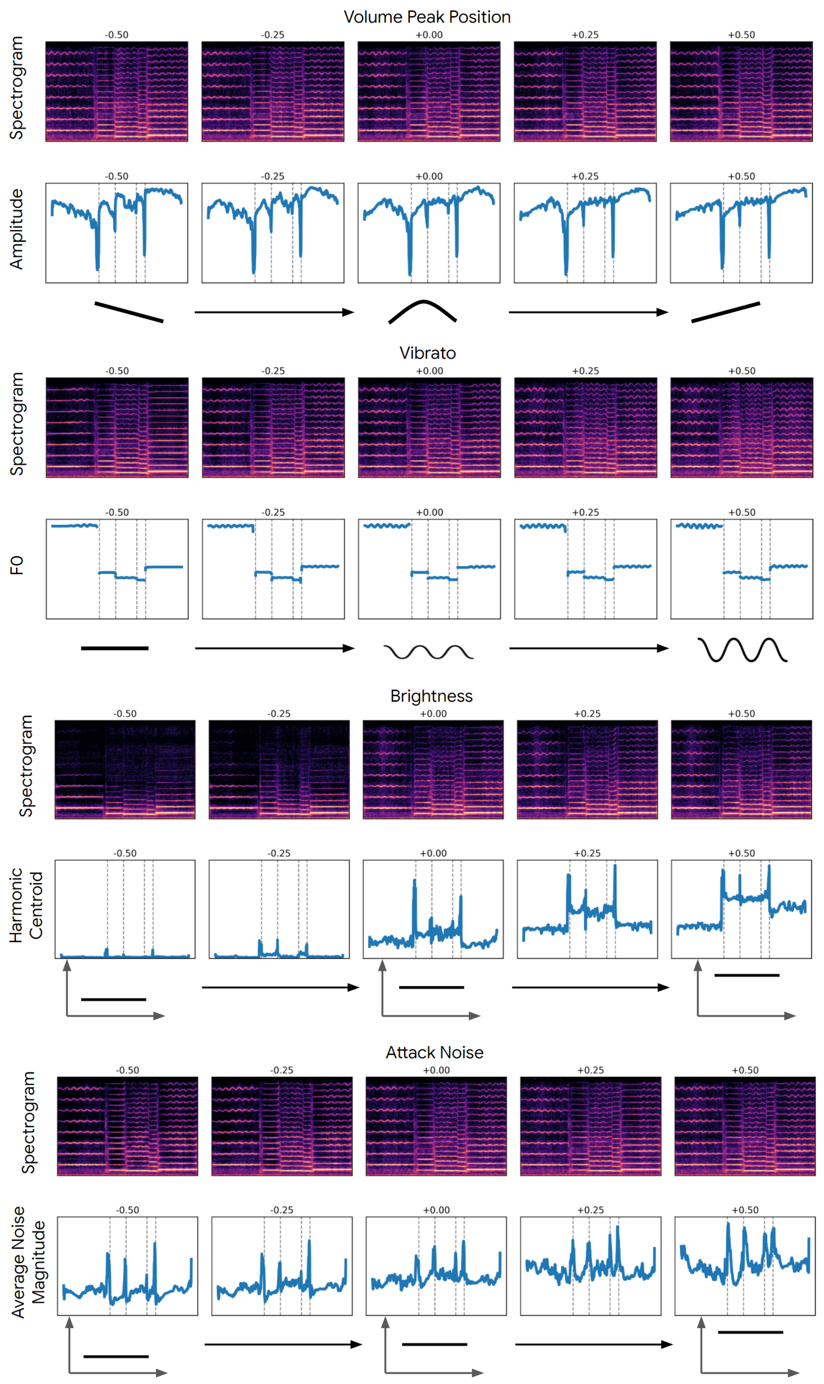


Fig. 4.13. The effect of modifying different note expression parameters on an existing sample. The number indicates the amount of modification to the note expression control where (+0) is the original sample. Upper figure shows the spectrogram of generations, and the bottom figure shows the quantity of the generations affected by the control. The cartoon on the side indicates how the modified feature would change the synthesis parameters. The gray dash line in the bottom figure indicates the note boundaries.

Chapter 5

Conclusion

In this thesis, we proposed MIDI-DDSP, which leverages a structured hierarchy to achieve controllable music performance synthesis. The proposed model is an instance of controllable generative models and is an effective tool for generating and controlling music performance.

The proposed MIDI-DDSP gives adaptable insights for enabling control in any generative models: Controls can be enabled in generative models by extracting features in intermediate levels and factorizing the generation pipeline according to domain workflow. In MIDI-DDSP, the intermediate-level representations are extracted from the data where no additional labels are needed, following a "analysis and synthesis" paradigm. This "analysis and synthesis" paradigm can be applied in future works to control generative models in other domains such as symbolic music generation or speech synthesis. On the other hand, one could replace the extracted feature-based intermediate representation with the learned prior or couple it with linguistic semantics that enables control via natural language.

The structured hierarchy and explicit feature extraction also make MIDI-DDSP a good dataset generator. The intermediate output of MIDI-DDSP will naturally serve as paired data for supervised models to train on, and it is easy to sample a large dataset from MIDI-DDSP. In our recent work [127], we proposed a dataset generation pipeline connecting Coconet [57] that generates MIDI and MIDI-DDSP, which renders the MIDI into audio. We use the pipeline to create a large-scale dataset we named CocoChorales dataset, which consists of 240,000 samples. We show that the CocoChorales dataset can improve the performance of the music transcription model on the low-resource dataset and enables source separation task that has too little data to train a model.

MIDI-DDSP can be further improved by developing the methods it relies on. Future advances in differentiable polyphonic synthesis and note detection could enable MIDI-DDSP to generate polyphonic instruments such as the piano. Currently, MIDI-DDSP replies on clean recordings for training data. Further development in unsupervised learning could enable transfer learning of expressive performance from noisy data.

Last, more efforts are needed beyond machine learning to make good creative tools powered by generative models. Although we demonstrated to enable controls in a music performance synthesis model suitable for music creation, there is still a gap between a controllable generative model and a tool music creators will prefer for their creation and production. As one of the most important directions to consider beyond machine learning, studies are needed from the angle of Human-Computer Interaction (HCI) to determine which form of interactions and visual representations are better for such creation tool [1, 25]. Those findings will in turn affect the design of the control in generative models, proposing more challenges from the machine learning side. Model development is another direction to consider beyond machine learning. To release a creation tool, sometimes one needs to make sure the generative model runs in real-time on a personal or portable device. This often involves redesigning the generative model, distilling the model, and optimizing model computation on hardware-specific platforms.

References

[1] Saleema Amershi, Dan Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi Iqbal, Paul N Bennett, Kori Inkpen, et al. Guidelines for human-ai interaction. In *Proceedings of the 2019 chi conference on human factors in computing systems*, pages 1–13, 2019.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[3] MIDI Manufacturers Association et al. The complete midi 1.0 detailed specification. *Los Angeles, CA, The MIDI Manufacturers Association*, 1996.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[5] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in Neural Information Processing Systems*, 33:12449–12460, 2020.

[6] Alexei Baevski, Wei-Ning Hsu, Qiantong Xu, Arun Babu, Jiatao Gu, and Michael Auli. Data2vec: A general framework for self-supervised learning in speech, vision and language. *arXiv preprint arXiv:2202.03555*, 2022.

[7] Rachel M Bittner, Brian McFee, Justin Salamon, Peter Li, and Juan Pablo Bello. Deep salience representations for f0 estimation in polyphonic music. In *ISMIR*, pages 63–70, 2017.

[8] Adrien Bitton, Philippe Esling, and Tatsuya Harada. Vector-quantized timbre representation. *arXiv preprint arXiv:2007.06349*, 2020.

[9] Merlijn Blaauw and Jordi Bonada. A Neural Parametric Singing Synthesizer Modeling Timbre and Expression from Natural Songs. *Applied Sciences*, 7(12):1313, dec 2017. ISSN 2076-3417. doi: 10.3390/app7121313.

[10] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: application to polyphonic music generation and transcription. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, pages 1881–1888, 2012.

[11] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[12] Gino Brunner, Andres Konrad, Yuyi Wang, and Roger Wattenhofer. MiDI-VAE: Modeling dynamics and instrumentation of music with applications to style transfer. Technical report, 2018. URL `https://junyanz.github.io/CycleGAN/`.

[13] Sergio Canazza, Giovanni De Poli, Carlo Drioli, Antonio Roda, and Alvise Vidolin. Modeling and control of expressiveness in music performance. *Proceedings of the IEEE*, 92(4):686–701, 2004.

[14] Rodrigo Castellon, Chris Donahue, and Percy Liang. Towards realistic midi instrument synthesizers. In *NeurIPS Workshop on Machine Learning for Creativity and Design (2020)*, 2020.

[15] Caroline Chan, Shiry Ginosar, Tinghui Zhou, and Alexei A Efros. Everybody dance now. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5933–5942, 2019.

[16] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[17] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *International conference on machine learning*, pages 1691–1703. PMLR, 2020.

[18] Rewon Child. Very deep vaes generalize autoregressive models and can outperform them on images. *arXiv preprint arXiv:2011.10650*, 2020.

[19] Kristy Choi, Curtis Hawthorne, Ian Simon, Monica Dinculescu, and Jesse Engel. Encoding musical style with transformer autoencoders. In *International Conference on Machine Learning*, pages 1899–1908. PMLR, 2020.

[20] John M Chowning. The synthesis of complex audio spectra by means of frequency modulation. *Journal of the audio engineering society*, 21(7):526–534, 1973.

[21] Thomas Christensen. *The Cambridge history of Western music theory.* Cambridge University Press, 2006.

[22] 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.

[23] Aidan Clark, Jeff Donahue, and Karen Simonyan. Adversarial video generation on complex datasets. *arXiv preprint arXiv:1907.06571*, 2019.

[24] Alexandre Défossez, Neil Zeghidour, Nicolas Usunier, Léon Bottou, and Francis Bach. Sing: Symbol-to-instrument neural generator. *arXiv preprint arXiv:1810.09785*, 2018.

[25] Dominik Dellermann, Adrian Calma, Nikolaus Lipusch, Thorsten Weber, Sascha Weigel, and Philipp Ebel. The future of human-ai collaboration: a taxonomy of design knowledge for hybrid intelligence systems. *arXiv preprint arXiv:2105.03354*, 2021.

[26] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://aclanthology.org/N19-1423`.

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

[28] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2020.

[29] Sander Dieleman, Aaron van den Oord, and Karen Simonyan. The challenge of realistic music generation: modelling raw audio at scale. In *Advances in Neural Information Processing Systems*, pages 7989–7999, 2018.

[30] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[31] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[32] Chris Donahue, Huanru Henry Mao, Yiting Ethan Li, Garrison W Cottrell, and Julian McAuley. Lakhnes: Improving multi-instrumental music generation with cross-domain pre-training. *arXiv preprint arXiv:1907.04868*, 2019.

[33] 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.

[34] Hao-Wen Dong, Cong Zhou, Taylor Berg-Kirkpatrick, and Julian McAuley. Deep performer: Score-to-audio music performance synthesis. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 951–955. IEEE, 2022.

[35] Douglas Eck and Juergen Schmidhuber. A first look at music composition using lstm recurrent neural networks. *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*, 103:48, 2002.

[36] Dan Ellis. Chroma feature analysis and synthesis. *Resources of laboratory for the recognition and organization of speech and Audio-LabROSA*, 5, 2007.

[37] Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. Gansynth: Adversarial neural audio synthesis. In *ICLR*, 2019.

[38] Jesse Engel, Lamtharn Hantrakul, Chenjie Gu, and Adam Roberts. DDSP: Differentiable digital signal processing. In *International Conference on Learning Representations*, 2020.

[39] Jesse Engel, Rigel Swavely, Lamtharn Hantrakul, Adam Roberts, and Curtis Hawthorne. Self-supervised pitch detection by inverse audio synthesis. In *International Conference on Machine Learning, Self-supervised Audio and Speech Workshop*, 2020.

[40] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12873–12883, 2021.

[41] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[42] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[43] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

[44] James D Hamilton. State-space models. *Handbook of econometrics*, 4:3039–3080, 1994.

[45] Lamtharn Hantrakul, Jesse H Engel, Adam Roberts, and Chenjie Gu. Fast and flexible neural audio synthesis. In *ISMIR*, pages 524–530, 2019.

[46] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the MAESTRO dataset. In *International Conference on Learning Representations*, 2019.

[47] Curtis Hawthorne, Ian Simon, Rigel Swavely, Ethan Manilow, and Jesse Engel. Sequence-to-sequence piano transcription with transformers. *arXiv preprint arXiv:2107.09142*, 2021.

[48] Curtis Hawthorne, Ian Simon, Adam Roberts, Neil Zeghidour, Josh Gardner, Ethan Manilow, and Jesse Engel. Multi-instrument music synthesis with spectrogram diffusion. *arXiv preprint arXiv:2206.05408*, 2022.

[49] Ben Hayes, Charalampos Saitis, and György Fazekas. Neural waveshaping synthesis. *arXiv preprint arXiv:2107.05050*, 2021.

[50] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16000–16009, 2022.

[51] Carlos Hernandez-Olivan and Jose R Beltran. Music composition with deep learning: A review. *arXiv preprint arXiv:2108.12290*, 2021.

[52] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. β-VAE: Learning basic visual concepts with a constrained variational framework. Technical report, 2017.

[53] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[54] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[55] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[56] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020.

[57] Cheng-Zhi Anna Huang, Tim Cooijmans, Adam Roberts, Aaron Courville, and Douglas Eck. Counterpoint by convolution. In *Proceedings of ISMIR 2017*, 2017.

[58] 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. URL `http://arxiv.org/abs/1809.04281`.

[59] Cheng-Zhi Anna Huang, Hendrik Vincent Koops, Ed Newton-Rex, Monica Dinculescu, and Carrie J Cai. Ai song contest: Human-ai co-creation in songwriting. In *ISMIR*, 2020.

[60] Natasha Jaques, Shixiang Gu, Dzmitry Bahdanau, José Miguel Hernández-Lobato, Richard E Turner, and Douglas Eck. Sequence tutor: Conservative fine-tuning of sequence generation models with kl-control. In *International Conference on Machine Learning*, pages 1645–1654. PMLR, 2017.

[61] Nicolas Jonason, Bob Sturm, and Carl Thomé. The control-synthesis approach for making expressive and controllable neural music synthesizers. In *2020 AI Music Creativity Conference*, 2020.

[62] Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aäron van den Oord, Sander Dieleman, and Koray Kavukcuoglu. Efficient neural audio synthesis. In *arXiv*, pages 2415–2424, 2018.

[63] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=Hk99zCeAb`.

[64] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[65] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8110–8119, 2020.

[66] Jong Wook Kim, Justin Salamon, Peter Li, and Juan Pablo Bello. Crepe: A convolutional representation for pitch estimation. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 161–165. IEEE, 2018.

[67] Jong Wook Kim, Rachel Bittner, Aparna Kumar, and Juan Pablo Bello. Neural music synthesis for flexible timbre control. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 176–180. IEEE, 2019.

[68] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[69] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014*, 2014.

[70] Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[71] Jungil Kong, Jaehyeon Kim, and Jaekyoung Bae. Hifi-gan: Generative adversarial networks for efficient and high fidelity speech synthesis. *Advances in Neural Information Processing Systems*, 33:17022–17033, 2020.

[72] Qiuqiang Kong, Yin Cao, Turab Iqbal, Yuxuan Wang, Wenwu Wang, and Mark D Plumbley. Panns: Large-scale pretrained audio neural networks for audio pattern recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28: 2880–2894, 2020.

[73] Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. Diffwave: A versatile diffusion model for audio synthesis. *arXiv preprint arXiv:2009.09761*, 2020.

[74] Kundan Kumar, Rithesh Kumar, Thibault de Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre de Brébisson, Yoshua Bengio, and Aaron C Courville. Melgan: Generative adversarial networks for conditional waveform synthesis. In *Advances in Neural Information Processing Systems*, pages 14910–14921, 2019.

[75] Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 29–37. JMLR Workshop and Conference Proceedings, 2011.

[76] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/

5.726791.

[77] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

[78] Sang-Hoon Lee, Hyun-Wook Yoon, Hyeong-Rae Noh, Ji-Hoon Kim, and Seong-Whan Lee. Multi-spectrogan: High-diversity and high-fidelity spectrogram generation with adversarial style combination for speech synthesis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13198–13206, 2021.

[79] Wonkwang Lee, Whie Jung, Han Zhang, Ting Chen, Jing Yu Koh, Thomas Huang, Hyungsuk Yoon, Honglak Lee, and Seunghoon Hong. Revisiting hierarchical approach for persistent long-term video prediction. *arXiv preprint arXiv:2104.06697*, 2021.

[80] Bochen Li, Xinzhao Liu, Karthik Dinesh, Zhiyao Duan, and Gaurav Sharma. Creating a multitrack classical music performance dataset for multimodal music analysis: Challenges, insights, and applications. *IEEE Transactions on Multimedia*, 21(2):522–535, 2018.

[81] Pei-Ching Li, Li Su, Yi-Hsuan Yang, Alvin WY Su, et al. Analysis of expressive musical terms in violin using score-informed and expression-based audio features. In *ISMIR*, pages 809–815, 2015.

[82] Jinglin Liu, Chengxi Li, Yi Ren, Feiyang Chen, Peng Liu, and Zhou Zhao. Diffsinger: Singing voice synthesis via shallow diffusion mechanism. *arXiv preprint arXiv:2105.02446*, 2, 2021.

[83] Ryan Louie, Andy Coenen, Cheng Zhi Huang, Michael Terry, and Carrie J Cai. Novice-ai music co-creation via ai-steering tools for deep generative models. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2020.

[84] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013.

[85] Rachel Manzelli, Vijay Thakkar, Ali Siahkamari, and Brian Kulis. Conditioning deep generative raw audio models for structured automatic music. In *19th International Society for Music Information Retrieval Conference*, 2018.

[86] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2794–2802, 2017.

[87] Marco Marchini, Rafael Ramirez, Panos Papiotis, and Esteban Maestre. The sense of ensemble: a machine learning approach to expressive performance modelling in string quartets. *Journal of New Music Research*, 43(3):303–317, 2014.

[88] Emile Mathieu, Tom Rainforth, Nana Siddharth, and Yee Whye Teh. Disentangling disentanglement in variational autoencoders. In *International Conference on Machine Learning*, pages 4402–4412. PMLR, 2019.

[89] RG McCurdy. Tentative standards for sound level meters. *Electrical Engineering*, 55 (3):260–263, 1936.

[90] Max Morrison, Zeyu Jin, Justin Salamon, Nicholas J Bryan, and Gautham J Mysore. Controllable neural prosody synthesis. In *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, volume 2020-Octob, pages 4437–4441, 2020. doi: 10.21437/Interspeech.2020-2918.

[91] 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.

[92] The American Heritage Dictionary of the English Language. American Heritage Dictionary Entry: Music. `https://ahdictionary.com/word/search.html?q=Music`, 2022. [Online; accessed 01-June-2022].

[93] Aaron Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George Driessche, Edward Lockhart, Luis Cobo, Florian Stimberg, et al. Parallel wavenet: Fast high-fidelity speech synthesis. In *International conference on machine learning*, pages 3918–3926. PMLR, 2018.

[94] Sageev Oore, Ian Simon, Sander Dieleman, Douglas Eck, and Karen Simonyan. This time with feeling: Learning expressive musical performance. *Neural Computing and Applications*, 32(4):955–967, 2020.

[95] OpenAI. MuseNet. `https://openai.com/blog/musenet/`, 2022. [Online; accessed 01-June-2022].

[96] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016.

[97] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

[98] Machel Reid, Yutaro Yamada, and Shixiang Shane Gu. Can wikipedia help offline reinforcement learning? *arXiv preprint arXiv:2201.12122*, 2022.

[99] 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*, pages 1198–1206, 2020.

[100] Yi Ren, Chenxu Hu, Xu Tan, Tao Qin, Sheng Zhao, Zhou Zhao, and Tie-Yan Liu. Fastspeech 2: Fast and high-quality end-to-end text to speech. *arXiv preprint arXiv:2006.04558*, 2020.

[101] Yi Ren, Jinglin Liu, and Zhou Zhao. Portaspeech: Portable and high-quality generative text-to-speech. *Advances in Neural Information Processing Systems*, 34:13963–13974, 2021.

[102] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014.

[103] Curtis Roads. Introduction to granular synthesis. *Computer Music Journal*, 12(2): 11–13, 1988.

[104] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music. Technical report, 2018. URL `https://goo.gl/magenta/musicvae-code`.

[105] Ruslan Salakhutdinov and Geoffrey Hinton. Deep boltzmann machines. In David van Dyk and Max Welling, editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 448–455, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr 2009. PMLR. URL `https://proceedings.mlr.press/v5/salakhutdinov09a.html`.

[106] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.

[107] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

[108] Christian Schörkhuber and Anssi Klapuri. Constant-q transform toolbox for music processing. In *7th sound and music computing conference, Barcelona, Spain*, pages 3–64, 2010.

[109] Diemo Schwarz. Corpus-based concatenative synthesis. *IEEE signal processing magazine*, 24(2):92–104, 2007.

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

[111] Siyuan Shan, Lamtharn Hantrakul, Jitong Chen, Matt Avent, and David Trevelyan. Differentiable wavetable synthesis. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4598–4602. IEEE, 2022.

[112] Chi-Ching Shih, Pei-Ching Li, Yi-Ju Lin, Yu-Lin Wang, Alvin WY Su, Li Su, and Yi-Hsuan Yang. Analysis and synthesis of the violin playing style of heifetz and oistrakh. In *Proceedings of the 20th International Conference on Digital Audio Effects (DAFx-17)*, 2017.

[113] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 32, 2019.

[114] Yang Song and Stefano Ermon. Improved techniques for training score-based generative models. *Advances in neural information processing systems*, 33:12438–12448, 2020.

[115] Yang Song and Diederik P Kingma. How to train your energy-based models. *arXiv preprint arXiv:2101.03288*, 2021.

[116] Stanley Smith Stevens, John Volkmann, and Edwin Broomell Newman. A scale for the measurement of the psychological magnitude pitch. *The journal of the acoustical society of america*, 8(3):185–190, 1937.

[117] Xu Tan and Xiaobing Li. A tutorial on ai music composition. In *Proceedings of the 29th ACM international conference on multimedia*, pages 5678–5680, 2021.

[118] Sergey Tulyakov, Ming-Yu Liu, Xiaodong Yang, and Jan Kautz. Mocogan: Decomposing motion and content for video generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1526–1535, 2018.

[119] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *9th ISCA Speech Synthesis Workshop*, pages 125–125, 2016.

[120] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. *Advances in neural information processing systems*, 29, 2016.

[121] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in Neural Information Processing Systems*, 30:6306–6315, 2017.

[122] Bryan Wang and Yi-Hsuan Yang. Performancenet: Score-to-audio music generation with multi-band convolutional residual network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1174–1181, 2019.

[123] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8798–8807, 2018.

[124] Xin Wang, Shinji Takaki, and Junichi Yamagishi. Autoregressive neural f0 model for statistical parametric speech synthesis. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(8):1406–1419, 2018.

[125] Xin Wang, Shinji Takaki, and Junichi Yamagishi. Neural source-filter waveform models for statistical parametric speech synthesis. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:402–415, 2019.

[126] Ziyu Wang, Yiyi Zhang, Yixiao Zhang, Junyan Jiang, Ruihan Yang, Junbo Zhao, and Gus Xia. Pianotree vae: Structured representation learning for polyphonic music. *arXiv preprint arXiv:2008.07118*, 2020.

[127] Yusong Wu, Joshua Gardner, Ethan Manilow, Ian Simon, Curtis Hawthorne, and Jesse Engel. Generating detailed music datasets with neural audio synthesis. In *Workshop on Machine Learning for Audio Synthesis at ICML*, 2022.

[128] Yusong Wu, Ethan Manilow, Yi Deng, Rigel Swavely, Kyle Kastner, Tim Cooijmans, Aaron Courville, Cheng-Zhi Anna Huang, and Jesse Engel. MIDI-DDSP: Detailed control of musical performance via hierarchical modeling. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=UseMOjWENv`.

[129] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *Proceedings of the International Conference on Machine Learning (ICML) Workshop*, 2015.

[130] Chih-Hong Yang, Pei-Ching Li, AW Su, Li Su, Yi-Hsuan Yang, et al. Automatic violin synthesis using expressive musical term features. In *Proceedings of the 19th International Conference on Digital Audio Effects (DAFx-16)*, pages 1–7. Brno, Czech Republic, 2016.

[131] Chengzhu Yu, Heng Lu, Na Hu, Meng Yu, Chao Weng, Kun Xu, Peng Liu, Deyi Tuo, Shiyin Kang, Guangzhi Lei, Dan Su, and Dong Yu. DurIAN: Duration Informed Attention Network for multimodal synthesis. *arXiv*, 2019.

[132] Jiangning Zhang, Xianfang Zeng, Yusu Pan, Yong Liu, Yu Ding, and Changjie Fan. Faceswapnet: Landmark guided many-to-many face reenactment. *arXiv preprint arXiv:1905.11805*, 2, 2019.

[133] Hang Zhao, Chuang Gan, Wei-Chiu Ma, and Antonio Torralba. The sound of motions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1735–1744, 2019.