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

Yusong Wu^{1}, **Ethan Manilow**^{2,4}, **Yi Deng**^{3}, **Rigel Swavely**^{4}, **Kyle Kastner**^{1},
Tim Cooijmans^{1}, **Aaron Courville**^{1}, **Cheng-Zhi Anna Huang**^{*1,4}, **Jesse Engel**^{*4}
^{1}Mila, Quebec Artificial Intelligence Institute, Université de Montréal,
^{2}Northwestern University, ^{3}New York University, ^{4}Google Brain
`wu.yusong@mila.quebec`
`{emanilow, annahuang, jesseengel}@google.com`

Abstract

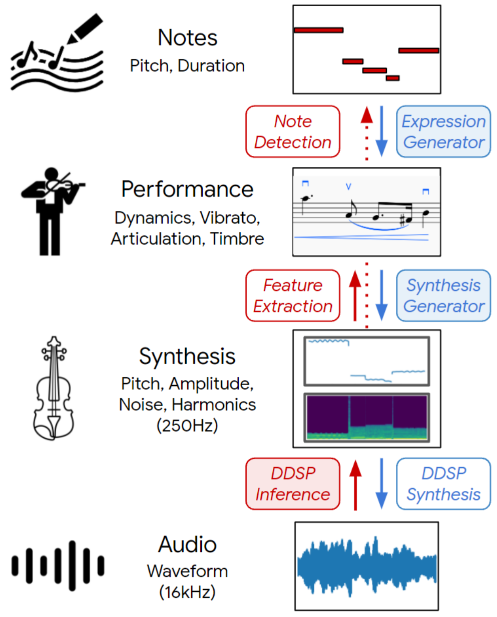
Musical expression requires control of both *what* notes are played, and *how* they are performed. Conventional audio synthesizers provide detailed expressive controls, but at the cost of realism. Black-box neural audio synthesis and concatenative samplers can produce realistic audio, but have few mechanisms 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.

1 Introduction

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 [1]. Deep generative models are expressive function approximators, capable of generating realistic samples in many domains [2–4], 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, where systems are optimized for both realism and control [5–8].

For music generation, despite recent progress, current tools still fall short of this ideal (Figure 1, right). Deep networks can either generate realistic full-band audio [9] or provide detailed controls of attributes such as pitch, dynamics, and timbre [10–14] but not both. Many existing workflows use the MIDI specification [15] to conventional DSP synthesizers [16, 17] provide extensive control but make

[*]equal contribution



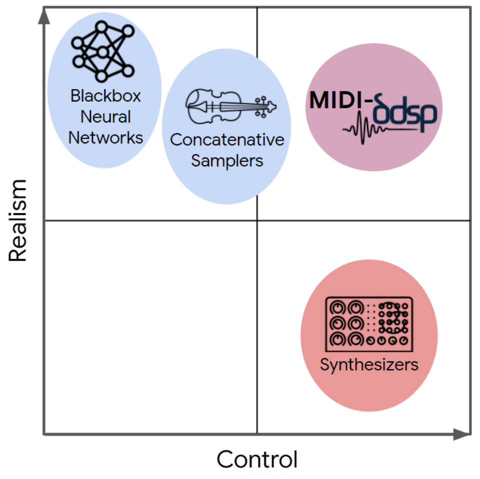


Figure 1: (Left) The MIDI-DDSP architecture. MIDI-DDSP extracts interpretable features at the performance and synthesis levels, building a modeling hierarchy by learning feature generation at each level. Red and blue components indicate encoding and decoding respectively. Shaded boxes represent modules 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 they have limited control. MIDI-DDSP enables both realistic neural audio synthesis and detailed user control.

it difficult to generate realistic instrument timbre, while concatenative samplers [18] play back high-fidelity recordings of isolated musical notes, but require manually stitching together performances with limited control over expression and continuity.

In this paper, we propose MIDI-DDSP, a hierarchical generative model of musical performance to provide both realism and control (Figure 1, left). Similar to conventional synthesizers and samplers that use the MIDI standard [15], 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 [12], 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 Figure 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 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.

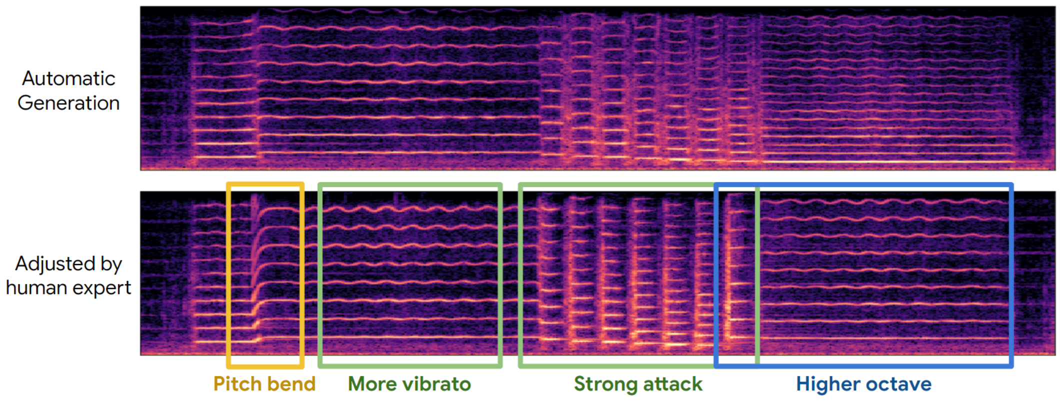


Figure 2: An example of detailed user control. Given an initial generation from the full MIDI-DDSP model (top), an expert musician can adjust notes (blue), performance attributes (green), and low-level synthesis parameters (yellow) to craft a personalized expression of a musical piece (bottom). To hear the difference in feeling, we highly recommend readers listen to the sample in the online supplement.

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, but has no fundamental barrier to adapting to multi-instrument polyphonic recordings as multi-pitch tracking and polyphonic transcription models progress [19–21]. 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 3)
- *Expression Attributes:* We introduce heuristics to extract mid-level per-note expression attributes from low-level synthesis parameters. (Figure 4)
- *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 2 and Figure 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 auto-regressively predicting note-wise expression attributes from a note sequence. (Tables 1a, 1b, 1c)
- *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 concatentive sampler software. (Figure 5)
- *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 [22] to generate and synthesize novel 4-part Bach chorales for a variety of instruments. (Figure 6)

Audio samples of all results and figures are provided in the online supplement[2]. We highly recommend readers to access the online supplement to the paper.

2 Related Work

Note Synthesis. Existing neural synthesis models allow either high-level manipulation of note pitch, velocity, and timing [13, 23, 14, 24], or low-level synthesis parameters [25–27]. MIDI-DDSP

[2]`https://midi-ddsp.github.io/`

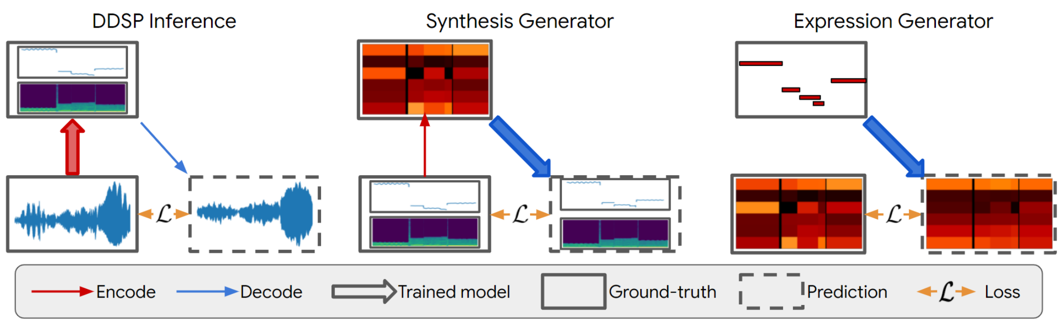


Figure 3: Separate training procedures for the three modules in MIDI-DDSP. (Left) The DDSP Inference module predicts synthesis parameters from audio and is trained via an audio reconstruction loss on the resynthesized audio. (Middle) The Synthesis Generator module predicts synthesis parameters from notes 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.

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 [26], a hierarchical model that autoregressively predicts frame-wise pitch and loudness contours to drive the original DDSP autoencoder [12]. 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. [28] and Dhariwal et al. [9] 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.

Expressive Performance Analysis and Synthesis. Many prior systems pair analysis and synthesis functions to capture expressive performance characteristics [29–31]. 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.

3 Model Architecture

3.1 DDSP Synthesis and Inference

Differentiable Digital Signal Processing (DDSP) [12] enables differentiable audio synthesis by using a harmonic plus noise model [32]. Full details are provided in Appendix B.1. 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.

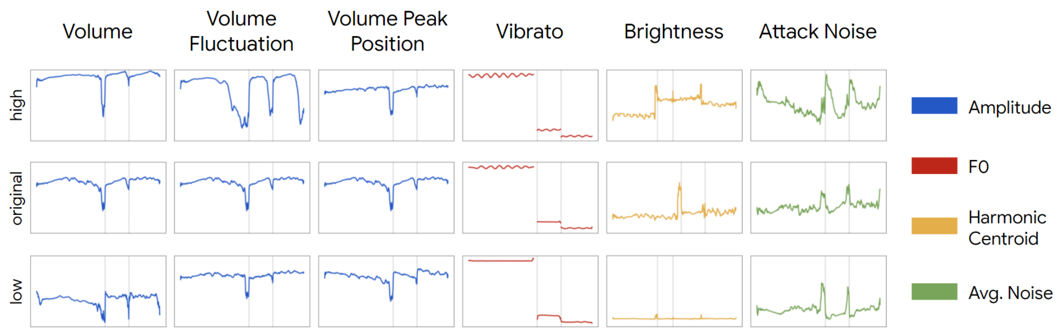


Figure 4: In MIDI-DDSP, manipulating note-level expression can effectively change the synthesis-level quantities. We show by taking a test-set sample (middle row) and adjusting each expression control value to lowest (bottom row) and highest (upper row), how each synthesis quantities (rightmost legend) would change. The dashed gray line in each plot indicates the note boundary.

Since the synthesis process is differentiable, Engel et al. [12] 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 [33, 12] of the resynthesized audio (Figure 3 left). $f_0(t)$ is extracted by a pre-trained CREPE model [34], and the loudness is extracted via an A-weighting of the power spectrum [35].

We extend this work for our DDSP Inference module, by providing and additional input features a log-scale Mel-spectrogram of the audio, that produces higher quality resynthesis (Table 1a). Full architectural details are provided in Appendix B.2.

3.2 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 six expression controls (detailed in Appendix B.3), scaled within $[0, 1]$. These are extracted from synthesis parameters $\boldsymbol{s}(t)$ and applied within the ith note, $\boldsymbol{n}_i(t)$, in a note sequence:

Volume: Controls the volume of a note, extracted by taken average amplitude over a note.

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

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 take the peak amplitude.

Brightness: Controls the timbre of a note where a higher value corresponds to larger high-frequency harmonic. Brightness is extracted by calculating the average harmonic centroid of a note.

Attack Noise: Controls how much noise occurs at the start of the note (the attack), e.g., the fluctuation of string and bow. At certain settings, this determines whether two notes sound consecutively or separately. The attack noise is extracted by taking a note's average noise magnitude in the first ten frames (40ms).

3.3 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 pooled 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:

$$\hat{f}_0(t) = g_\theta(\boldsymbol{c}(t)), \quad \hat{a}(t), \hat{\boldsymbol{h}}(t), \hat{\boldsymbol{\eta}}(t) = g_\phi(\boldsymbol{c}(t), \hat{f}_0(t)), \tag{1}$$

where θ denotes trainable parameters in the autoregressive RNN, and ϕ indicates trainable parameters in the convolutional GAN. Architectural details for both of these details is provided in Appendix B.4. The autoregressive RNN is trained using cross-entropy loss $\mathcal{L}_{ce}$. The generator of the convolutional GAN is trained by a multi-scale spectral loss $\mathcal{L}_{spec}$ (Eq. 12) and an adversarial objective consisting of a least-squares GAN (LSGAN) $\mathcal{L}_{lsgan}$ [36] loss and a feature matching loss $\mathcal{L}_{fm}$ [37] (Eq. 15 to Eq. 18). Thus, the total loss applied to the Synthesis Generator can be written as:

$$\mathcal{L} = (\mathcal{L}_{ce} + \mathcal{L}_{spec}) + \alpha(\mathcal{L}_{lsgan} + \gamma\mathcal{L}_{fm}). \tag{2}$$

In training, the ground-truth $f_0(t)$ is input to the convolutional GAN, thus there is no gradient from the convolutional GAN into the autoregressive RNN.

3.4 Expression Generator

The Expression Generator uses an autoregressive RNN to predict note expression controls from note sequence (Appendix B.6). 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 3 right). At inference time, the output note expression is generated autoregressively and deterministically.

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, note transcription models can be used to provide the note labels.

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 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? We encourage readers to listen to the samples provided in the online supplement.

4.1 Dataset

To demonstrate modeling a variety of instruments, we use the URMP dataset [38], a publicly-available audio dataset containing monophonic solo performances of a variety of instruments. URMP is widely used in music synthesis research [39–41, 20]. The URMP dataset contains solo performance recordings of 13 string instruments and wind instruments, which allows us to test generalization to 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 [13]. 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.

Table 1: Each module in MIDI-DDSP produces high-quality reconstruction and prediction. Reconstruction accuracy of each module are shown in table comparing to other methods.

Model	Spectral Loss
DDSP Inference	**4.28**
Engel et al. [12]	5.00

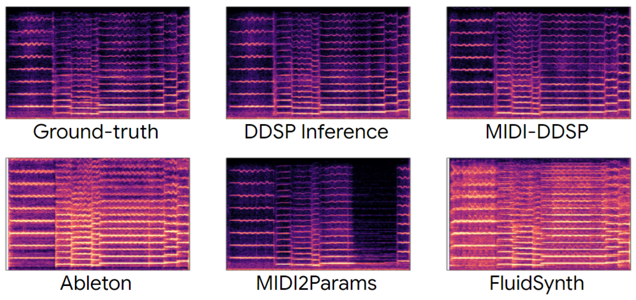
(a)

Model	RMSE
Synthesis Generator	**0.19**
MIDI2Params	0.26

(b)

Models	RMSE
Expression Generator	**0.14**
MIDI2Params	0.23

(c)



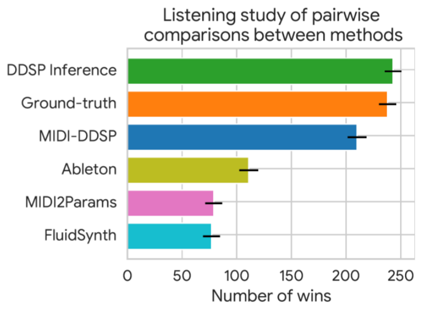


Figure 5: (left) Comparing the log-scale Mel spectrograms of synthesis results from test-set note sequences, MIDI-DDSP synthesizes more realistic audio (more similar to ground-truth and DDSP Inference) than prior work score-to-audio method MIDI2Params (enlarged in Figure 7). This is also reflected in the listening study (right), where the MIDI-DDSP synthesis is also perceived as more realistic than the professional concatenative sampler Ableton and the freely available FluidSynth.

4.2 Model Accuracy

Modules in MIDI-DDSP can accurately reconstruct output at multiple levels of the hierarchy (Figure 3). We evaluate the reconstruction quality of MIDI-DDSP by evaluating each module. Results are shown in Table 1.

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 1a, 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 1b, 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 auto-regressive, 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. auto-regressive at a much higher temporal resolution. We again compare to MIDI2Params, where we auto-regressively 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 1c, the Expression Generator can accurately predict the note expression. In comparison, MIDI2Params without performance-level modeling suffers from predicting the note expression on a higher level when compared to the frame-wise sequence models.

Table 2: The note expression outputs are strongly correlated with input adjustment. The Pearson correlation r-values are shown in the table (all entries $p < 0.0001$). The bold numbers indicate a Pearson r-value larger than 0.7, which we consider to indicate strong correlation between the input control and the respective output quantity. For simplicity, only four instruments are shown. More results can be found in Table 7.

	Volume	Vol. Fluc.	Vol. Peak Pos.	Vibrato	Brightness	Attack Noise
All instruments	**.97**	**.78**	.57	**.70**	**.92**	**.93**
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**

4.3 Audio Quality Evaluation by Humans

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 3.1), MIDI2Params [26], and two concatenative samplers: FluidSynth and Ableton (detailed in Appendix D.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 5 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 5 (right) are all statistically significant with the exception of ground truth versus DDSP Inference and MIDI2Params versus FluidSynth (Table 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 5, left). While the DDSP Inference module faithfully re-synthesizes the ground-truth audio, MIDI-DDSP generates coherent performance from a series notes and has rich, varying expressions across the notes. MIDI2Params failed to generate coherent expression within a note, generating unrealistic pitch and loudness contours. Also, MIDI2Params stopped the note in the middle when generating the fifth note, suggesting that such a frame-wise generation model is limited in modeling long-term dependency even inside a single note. On the contrary, the note expression modeling in MIDI-DDSP allows it to model dependency 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, it fails to generate expression dependency among notes automatically.

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 2 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.

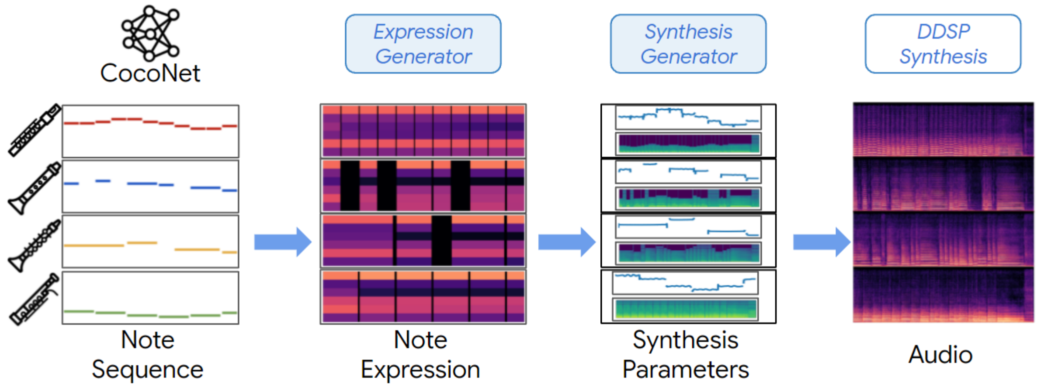


Figure 6: MIDI-DDSP can take input from different sources (human or other models) by designing explicit latent representations at each level. A full hierarchical generative model for music can be constructed by connecting MIDI-DDSP with an automatic composition model. Here, we show MIDI-DDSP taking note input from a score level Bach composition model and automatically synthesizing a Bach quartet by generating explicit latent for each level in the hierarchy.

Figure 4 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.

4.5 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.

On one end of this spectrum, Figure 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 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 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 6, MIDI-DDSP can be combined with a composition Bach chorales model COCONET [22], 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.

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.

References

[1] 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. *ArXiv*, abs/2010.05388,

2020.

[2] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. *arXiv preprint arXiv:2102.12092*, 2021.

[3] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

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

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

[13] 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.

[14] 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.

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

[16] 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.

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

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

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

[20] 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.

[21] 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.

[22] Cheng-Zhi Anna Huang, Tim Cooijmans, Adam Roberts, Aaron Courville, and Douglas Eck. Counterpoint by convolution. In *Proceedings of ISMIR 2017*, 2017. URL `https://ismir2017.smcnus.org/wp-content/uploads/2017/10/187_Paper.pdf`.

[23] 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.

[24] 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.

[25] 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.

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

[27] 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. URL `http://www.mdpi.com/2076-3417/7/12/1313`.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

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

[36] 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.

[37] 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.

[38] 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.

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

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

[41] 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.

[42] 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.

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

[44] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

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

[50] 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.

[51] 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.

[52] 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. URL `https://openreview.net/forum?id=rygGQyrFvH`.

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

A Appendix

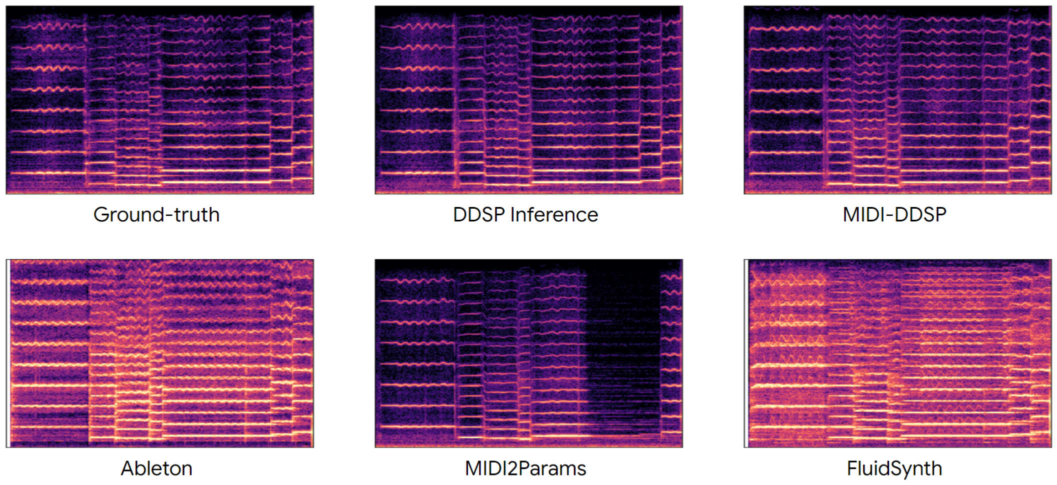


Figure 7: The enlarged log-scale Mel spectrograms of synthesis results in Figure 5

B Model Details

B.1 DDSP Synthesizer

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 defined with interpretable meanings. 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 $\boldsymbol{\eta}(t)$: $\text{exp-sigmoid}(x) = 2.0 \cdot \text{sigmoid}(x)^{\log 10} + 10^{-7}$. 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. 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. 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:

$$\left.\begin{array}{ll} \mathrm{IR}'(t) = \mathrm{IR}(t), & 0 \leq t \leq 16000 \\ \mathrm{IR}'(t) = \mathrm{IR}(t) \cdot \exp\left(-4(t-16000)\right), & 16000 < t \leq 48000 \end{array}\right\} \quad (3)$$

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

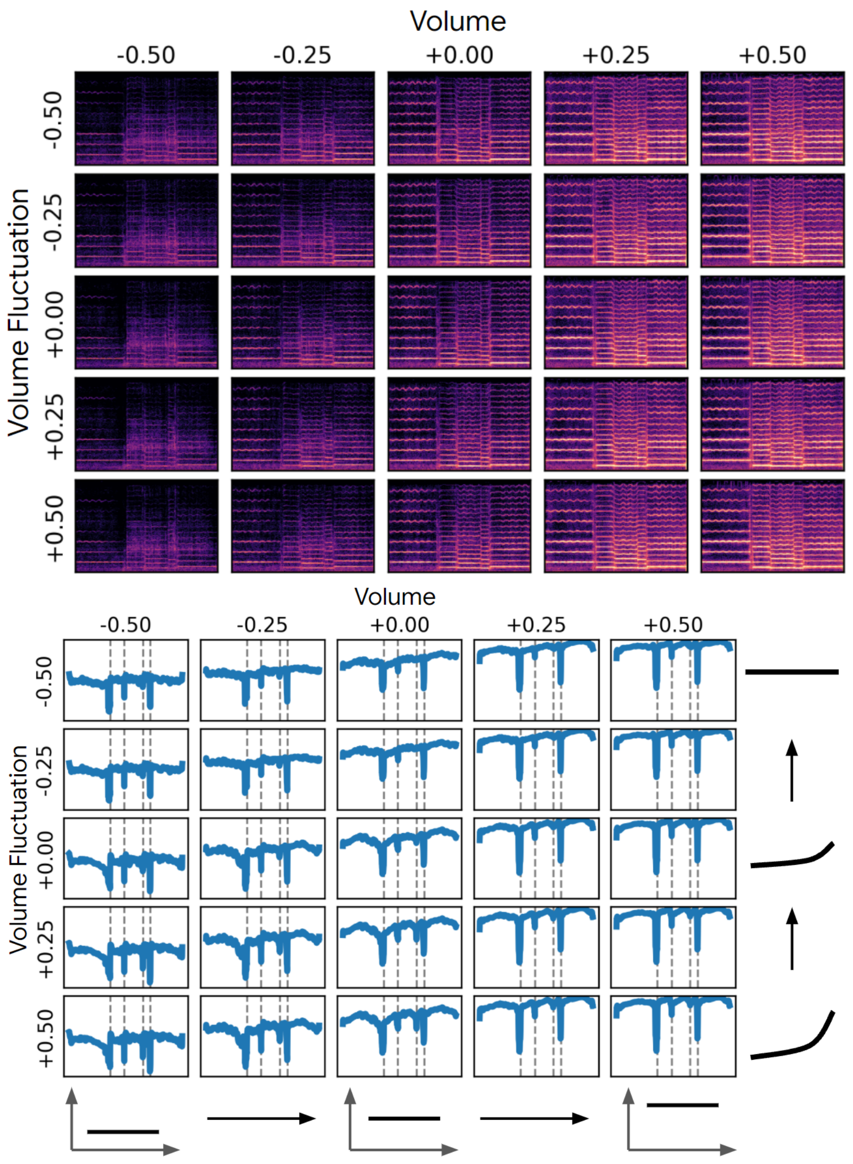


Figure 8: The effect of modifying note expression on an existing sample. The number indicates the amount of modification to the note expression control where (+0, +0) is the original sample. Upper figure shows the spectrogram of generations, and the bottom figure shows the amplitude of the generations. 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.

B.2 DDSP Inference

In DDSP Inference, a fully-connected network is used on fundamental frequency and loudness, and an 8-layer convolutional neural network (CNN) [42] is used on log-scale Mel-spectrogram to extract features. Then a bi-directional long-short term memory network (LSTM) [43] and a fully connected layer are used to map the features to synthesis parameters.

Similar to the original DDSP [12], the fundamental frequency is estimated by CREPE [34] model, and the loudness is extracted via an A-weighting of the power spectrum [35]. In extracting features from the audio, our approach differs from the autoencoder in the original DDSP [12] work. We use a CNN to extract additional features from the log-scale Mel-spectrogram to allow better feature extraction with more information for predicting synthesis parameters. Features are extracted by an

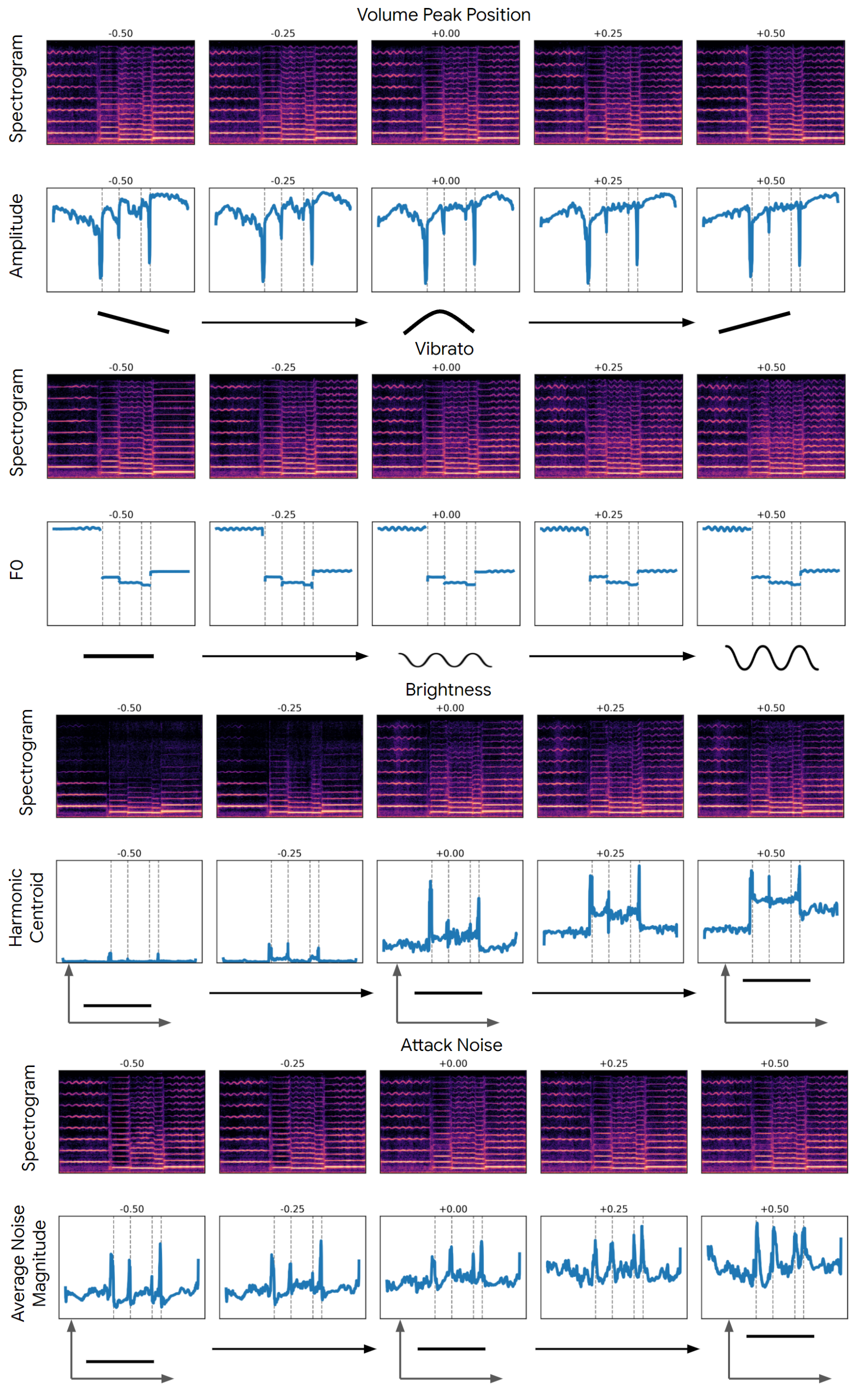


Figure 9: The effect of modifying note expression on an existing sample. The number indicates the amount of modification to the note expression control where (+0, +0) is the original sample. Upper figure shows the spectrogram of generations, and the bottom figure shows the amplitude of the generations. 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.

Table 3: The architecture of the CNN used to extract features from log-scale Mel-spectrogram in the DDSP Inference module.

ConvBlock	Output Size	Kernel Size	Stride	Filter Size	Dropout Rate
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**	**Kernel Size**	**Stride**	**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	-

8-layer CNN with batch normalization [44] and are concatenated with the features extracted by a fully-connected layer from fundamental frequency and loudness. The detailed architecture of the CNN in the DDSP Inference Module is shown in Table 3. The CNN network in the DDSP Inference module is similar to the one used in [45] for computational efficiency.

The DDSP Inference Module is optimized via Adam optimizer in a batch size of 16 and a learning rate of 0.0003. We choose a log-scale Mel-spectrogram of 64 dimensions in this work to extract features by CNN. The features extracted by CNN are mapped to a dimension of 256, as well as the features extracted from the fundamental frequency and loudness. If trained in the multi-instrument setting, an instrument embedding in 64 dimensions will be concatenated. The bi-directional LSTM has a hidden dimension of 256.

B.3 Expression Controls

Given frame-wise extracted synthesis parameters $\boldsymbol{s}(t)$ and a note starts at frame T_{on} and ends at frame T_{off}, we define $\tau \in [T_{on}, T_{off}]$ and total frame duration of the note $T_n = T_{off} - T_{on}$. The synthesis parameters for a single note can be known, namely fundamental frequency $f_0(\tau)$, amplitude $a(\tau)$, harmonic distribution $\boldsymbol{h}(\tau)$, and noise magnitude $\boldsymbol{\eta}(\tau)$.

In order to make the scalar note expression controls value better match the perceptual of human, the $a(\tau)$, and $\boldsymbol{\eta}(\tau)$ are transformed into log scale:

$$a'(\tau) = 20\log_{10} a(\tau), \quad \boldsymbol{\eta}'(\tau) = 20\log_{10} \boldsymbol{\eta}(\tau) \tag{4}$$

The note expression controls are extracted as follows:

Volume

$$\frac{1}{T_n}\sum_{i=1}^{T_n} a'(i). \tag{5}$$

Volume fluctuation

$$\sqrt{\frac{1}{T_n}\sum_{i=1}^{T_n}(a'(i) - \frac{1}{T_n}\sum_{j=1}^{T_n} a'(j))^2}. \tag{6}$$

Volume peak position

$$\frac{1}{T_n}\arg\max_i a'(i) \quad \forall i \in [1, T_n]. \tag{7}$$

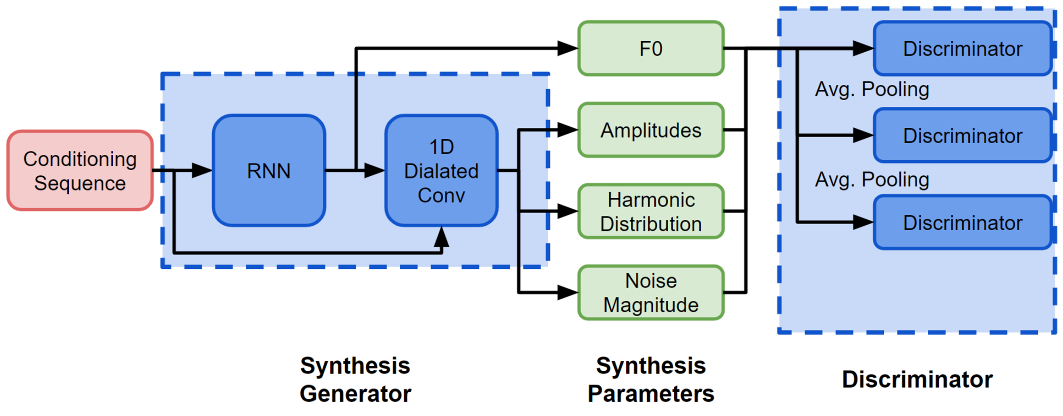


Figure 10: The architecture of the Synthesis Generator.

Vibrato Inspired by previous works on expressive performance analysis [46, 47, 30], the vibrato is calculated by applying Discrete Fourier Transform (DFT) to the fundamental frequency sequence:

$$\max_i \mathcal{F}\{f_0(t)\}_i, \tag{8}$$

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 zero. In calculating the DFT function, the fundamental frequency is zero-padded to 1000 frames.

Brightness

$$b = \frac{1}{T_n} \sum_i^{T_n} i \cdot \boldsymbol{h}^k(i), \tag{9}$$

where $\boldsymbol{h}^k(i)$ represents the k-th bin of the harmonic distribution in the i-th time-step.

Attack Noise

$$\sum_{i=1}^{10} \sum_{k=1}^{65} \boldsymbol{\eta}'^k(i), \tag{10}$$

where $\boldsymbol{\eta}'^k(i)$ represents the k-th bin of the noise magnitudes in the i-th time-step.

One can also come up with other note expression controls that describe expression controls in a note. Alternatively, the expression controls can be learned in an unsupervised way by neural networks. However, we leave that for future work.

B.4 Synthesis Generator

The architecture of the Synthesis Generator is shown in Figure 10. In constructing conditioning sequence from note expression and note sequence, frame-wise note expression and note pitch is expanded to frame-wise sequence by repeating the number of frames the note occupies. Then, fame-wise note expression and pitch are concatenated with binary note onsets, offsets, and a scalar note positioning code to provide additional information for note boundary.

The autoregressive RNN for $f_0(t)$ generation consists of a single-layer Bi-LSTM and a 2-layer GRU that autoregressively sample the deviation in pitch from note. As the $f_0(t)$ is generated by adding the deviation from the note pitch, a stacked dilated 1-D convolution [4] is used to generate the rest of synthesis parameters given both $\boldsymbol{c}(t)$ and $f_0(t)$ as condition. The stacked convolution consists of 4 stacks, and each stack has five layers of 1-D convolution with exponentially increasing dilation rate followed by ReLU activation [48] and layer normalization [49]. The autoregressive RNN outputs in

Table 4: The architecture of the dilated convolution network used in the Synthesis Generator.

Dilated Stack	Output Size	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 Conv	**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 5: The architecture of the discriminator block used in the Synthesis Generator.

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 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	$(T_{in}/2, 256)$	-	-	-

the semitone range of $[-1, 1]$. The final $f_0(t)$ is predicted by adding the note pitch in the semitone scale.

The conditioning sequence input is mapped to a dimension of 256 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. The autoregressive RNN in the Synthesis Generator has a hidden dimension of 256. A dropout of rate 0.5 is applied to all GRU units and autoregressive input during training to avoid overfitting and exposure bias.

The full details of the dilated convolutional network architecture are shown in Table 4.

The architecture of the discriminator used in the Synthesis Generator is shown in Figure 11, and the detailed architecture of the discriminator block is shown in Table 5. The discriminator is motivated by the multi-scale discriminator network in previous works generating waveforms and

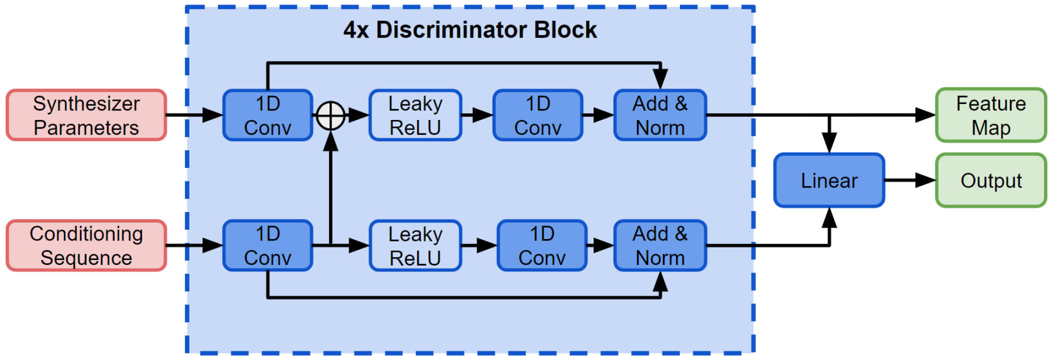


Figure 11: The architecture of the discriminator used in the Synthesis Generator.

spectrograms [37, 50]. It consists of 3 identical conditional discriminator networks running at different scales to learn features of synthesis parameters at different time resolutions. One single 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 [51], with skip connections and layer normalization [49].

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

$$\mathcal{L} = \mathcal{L}_{recon} + \alpha \mathcal{L}_{adv}. \tag{11}$$

The reconstruction loss consists of a cross-entropy loss to train the RNN that predicts fundamental frequency and a multi-scale spectral loss to train the dilated convolution that predicts the amplitude, harmonic distribution, and noise magnitude:

$$\mathcal{L}_{recon} = -\sum_i f_{0_i} \log \hat{f}_{0_i} + \mathcal{L}_{spec}. \tag{12}$$

The multi-scale spectral loss used in DDSP [12] is used for the reconstruction loss. The multi-scale spectral loss computes the L1 difference between Short-Time Fourier Transform (STFT) of the predicted and target audio in different FFT sizes. Given the 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 $\log S_i$:

$$\mathcal{L}_{spec}^{(i)} = ||S_i - \hat{S}_i||_1 + \beta || \log S_i - \log \hat{S}_i||_1, \tag{13}$$

$$\mathcal{L}_{spec} = \sum_i \mathcal{L}_{spec}^{(i)} \quad \forall i \in \{2048, 1024, 512, 256, 128, 64\}. \tag{14}$$

The adversarial loss used to train the dilated convolutional network in the Synthesis Generator combines a least-squares GAN (LSGAN) [36] objective and a feature matching objective objective [37]:

$$\mathcal{L}_{adv} = \mathcal{L}_{lsgan} + \gamma \mathcal{L}_{fm}. \tag{15}$$

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

$$\mathcal{L}_{lsgan} = \mathbb{E}_{\boldsymbol{c}} \left[\sum_{k=1,2,3} ||D_k(\hat{\boldsymbol{s}}, \boldsymbol{c}) - 1||_2 \right], \tag{16}$$

and the LSGAN objective for training the discriminator is:

$$\min_{D_k} \mathbb{E}\left[||D_k(\boldsymbol{s},\boldsymbol{c}) - 1||_2 + ||D_k(\hat{\boldsymbol{s}},\boldsymbol{c})||_2\right], \forall k = 1, 2, 3. \tag{17}$$

Given the output of i-th feature map from the k-th discriminator D_k, the feature map matching objective is calculated as L1 difference between corresponding feature map:

$$\mathcal{L}_{fm} = \mathbb{E}_{\boldsymbol{s},\boldsymbol{c}}\left[\sum_{i=1}^{4} \frac{1}{N_i}||D_k^{(i)}(\boldsymbol{s},\boldsymbol{c}) - D_k^{(i)}(\hat{\boldsymbol{s}},\boldsymbol{c})||_1\right], \tag{18}$$

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

In training, the gradient from the adversarial loss is blocked before the 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.

At inference time, the autoregressive RNN in the Synthesis Generator sample fundamental frequency deviation using nucleus sampling [52] with a $p = 0.95$ to avoid sudden unrealistic change in fundamental frequency contour.

B.5 Improvement of Using GAN in Synthesis Generator

Without GAN, the model will suffer from the "over-smoothing" problem where the model generates uniform and smoothed harmonic distribution and noise magnitudes over the whole note. We show in Figure 12 for a qualitative effect of using the GAN. With the introduction of adversarial training, the proposed model overcomes the over-smoothing problem potentially caused by one-to-many mapping.

B.6 Expression Generator

In the input of the Expression Generator, the discrete pitch is mapped into a latent space of 64 dimensions via embedding layer, and the scalar duration is mapped into a latent space of 64 dimensions via a fully-connect layer. The input to the bi-directional GRU is a concatenation of two 64 dimension inputs. For the multi-instrument model, an instrument id is additionally mapped to a 64 dimension vector via embedding layer and concatenated to the input. 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 uses a two-layer Multi-layer Perceptron (MLP) consists of a fully connected layer with a layer normalization before the ReLU nonlinearity used in [12].

The Expression Generator is trained on a sequence length of 64 notes and a batch size of 256. Adam optimizer [53] is used in training with a learning rate of 0.0001. The mean-square-loss (MSE) is used to train the Expression Generator.

B.7 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.

C Dataset

The 13 instruments in the URMP dataset are violin, viola, cello, double bass, flute, oboe, clarinet, saxophone, bassoon, trumpet, horn, trombone, tuba. The recordings in the dataset have a sample rate of 48kHz but are down-sampled to 16kHz to match the sample rate of the DDSP synthesis

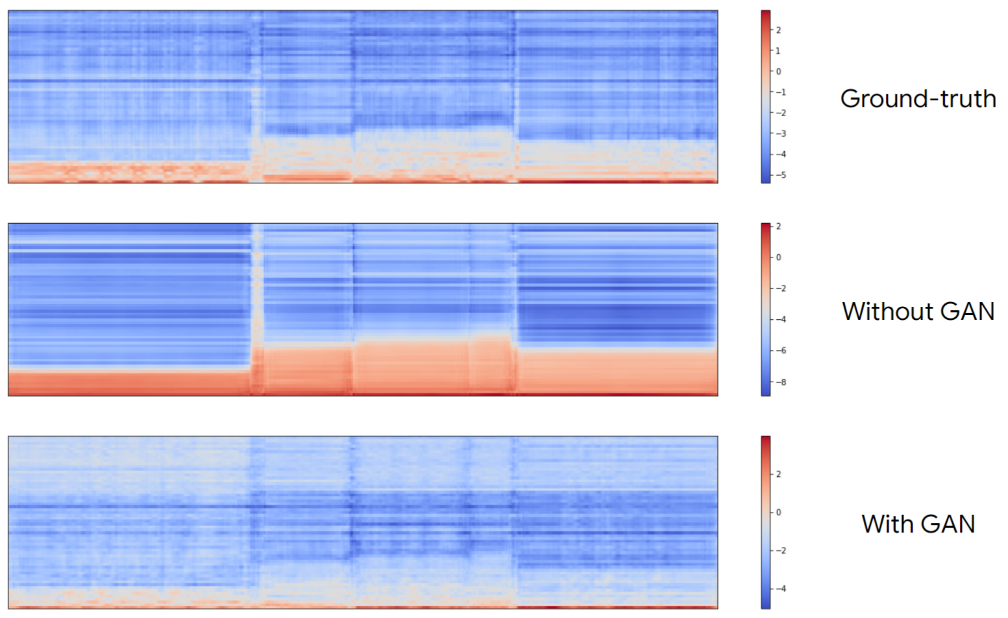


Figure 12: The effect of GAN in overcoming the "over-smoothing" problem in harmonic distribution generation. From top to bottom: (top) the ground-truth harmonic distribution of the test-set sample, (middle) the harmonic distribution of the same sample predicted without a discriminator used in training, (bottom) the harmonic distribution predicted with discriminator and adversarial training.

module. Recordings in the dataset are chunk into segments of 4 seconds with 50% overlap to train the Synthesis Generator.

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 recording but post-hoc adjusted the piece so that the same piece does not appear in both training and test set.

D Experiment Details

D.1 Details of Comparing Methods

We use the orchestral strings pack in Ableton[3] without any manual adjustment to synthesize the audio from Ableton. The FluidSynth use the sound font of `FluidR3_GM.sf2`[4]. The MIDI2Param system proposed by [26] is implemented on our code base following the official implementation on GitHub[5]. The MIDI2Param system is only trained in the single instrument setting for listening test experiments as practiced by the original paper but is trained in a multi-instrument setting with an add of instrument embedding in other experiments.

D.2 Details of Listening Test

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 6.

[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`

Table 6: A post-hoc comparison of each pair on their pairwise comparisons with each other, using the Wilcoxon signed-rank test for matched samples. p value less than $0.01/15 = 6.67 \times 10^{-4}$ yields a statistically significant difference. Only two pairs are not significantly different (DDSP Inference vs. Ground-truth, MIDI2Params vs. FluidSynth), and are marked in bold.

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
MIDI-DDSP	Ground-truth	4	0	60	1.00e-37
MIDI-DDSP	MIDI-DDSP	9	0	55	7.25e-26
Ground-truth	MIDI-DDSP	44	0	20	0.00018

Table 7: The Pearson correlation result of all instruments described in Table 2. The Pearson correlation r-values are shown in the table, while p-values are omitted as in all entries $p < 0.0001$. The bold numbers indicate a Pearson r-value larger than 0.7 that can be considered strongly correlated. For simplicity, only four instruments are shown.

	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**

E Expression Attribute Control

Table 7 shows the full results of the Pearson correlation test of the note expression controls.

F An Example User Experience

This section presents a step-by-step user experience of an expert using and adjusting MIDI-DDSP to obtain satisfied music performance. Specifically, the human expert as a violin player will iterate on adjusting note expression controls, starting from what the expression generator predicts for a given piece. Figure 13 shows the original music score "Viva La Vida" as input to MIDI-DDSP. The synthesized audio is shown in Figure 17.

1. Design and Adjust Articulations The expert first designs the articulations of the piece based on automatic generation, marking up/down bow and the tie between the note, indicating the connections between notes. The results are shown in Figure 14, where ties indicate the notes connected should be played as legato (played consecutively) and should have less attack noise.

The expert adjusts the attack noise based on what the articulations designed. In here, the expert lowers the attack noise of the legato notes. The expert first applies the adjustments to the notes. Then, the expert fine-tuned the attack noise adjustments based on the synthesis result of the earlier adjustment. The synthesized audio is shown in Figure 18.

2. Design and Adjust Fingering The expert designs the fingering based on one's knowledge and experience of violin performance. The results are shown in Figure 15. The expert designs the first two measures of the piece to be played on E string ("E- - - -..." mark on Figure 15), and assign the fingering of each note (numbers above the notes in Figure 15). The choice of the string and the fingering of notes affects the brightness of the timbre. That is, playing on the E string should have a brighter timbre, and playing on an open string (fingering number "0") will have a brighter timbre compared to pressing the string (fingering number other than "0").

The expert adjusts the notes' brightness based on the fingering design, based on the principle described above. The expert will first apply an initial adjustment to the brightness, then fine-tune the number according to the synthesis result. The synthesized audio is shown in Figure 19.

3. Adjust Vibrato The expert adjusts the vibrato of each note. Longer notes will be adjusted to have more vibrato to match how the expert would play the violin. Similarly, the expert first adjusts the note's vibrato based on the model prediction and fine-tune the numbers based on the synthesis result. The synthesized audio is shown in Figure 20.

4. Design Special Articulations The expert designs special articulations of the notes for artistic purposes. In this example, the expert designed some notes to play as staccato. The results are shown in Figure 16 where the purple dot above the notes indicates it to play as staccato.

To adjust notes to staccato, the volume fluctuation, volume peak position, and attack noise needs to be adjusted from the original value. Specifically, the expert increases the volume fluctuation, decreases the volume peak position, and increases attack noise. Similarly, the expert first adjusts the vibrato of the note and fine-tune the numbers based on the synthesis result. In this step, the expert report that decreasing vibrato can help generate better staccato notes. The synthesized audio is shown in Figure 21.

5. Fine-tune the Piece After all designing and adjustments are made, if needed, the expert takes a final revision of the piece by fine-tuning the expression controls of the notes. In this example, the expert does not fine-tune the piece in the end as the audio synthesized by the previous step is good enough.

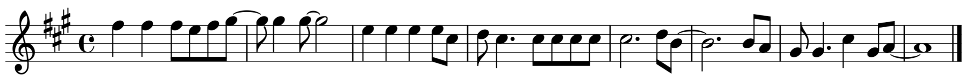


Figure 13: The original score "Viva La Vida" input to MIDI-DDSP

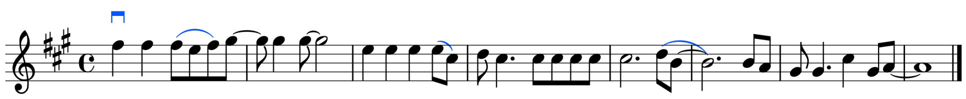


Figure 14: The score with articulations (blue) designed based on automatic generation.

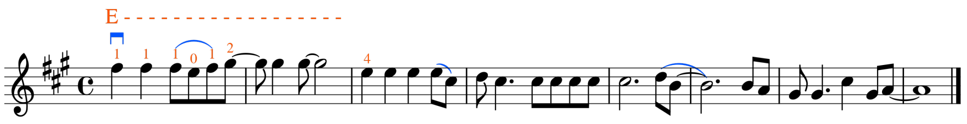


Figure 15: The score after designing articulations (blue) and fingerings (orange).

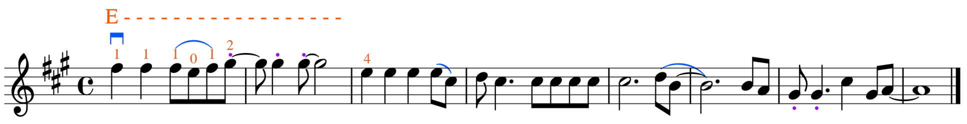


Figure 16: The score after designing articulations (blue), fingerings (orange) and special articulations (purple).

G Open-Source Image Attribution

The icons used throughout the paper are used under the Creative Commons license via the Noun Project. We gratefully acknowledge the following creators of these images:

- Audio by cataicon from the Noun Project.
- bassoon by Symbolon from the Noun Project.
- Clarinet by Symbolon from the Noun Project.
- composer by Pham Duy Phuong Hung from the Noun Project.
- Flute by Symbolon from the Noun Project.
- Neural Network by Ian Rahmadi Kurniawan from the Noun Project.
- oboe by Symbolon from the Noun Project.
- Synthesizer by Jino from the Noun Project.
- Violin by Olena Panasovska from the Noun Project.
- Violinist by Luis Prado from the Noun Project.

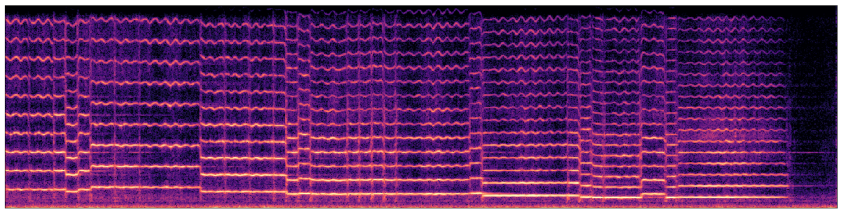


Figure 17: The synthesis audio predicted by MIDI-DDSP given score as input.

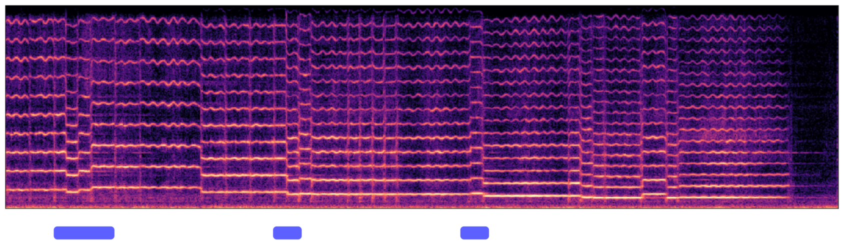


Figure 18: The synthesis audio after designing and adjusting articulations (blue).

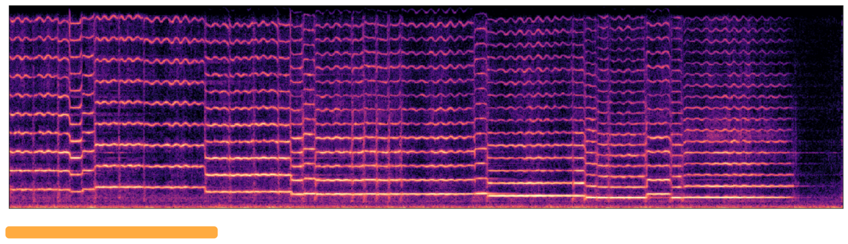


Figure 19: The synthesis audio after designing and adjusting fingerings (orange).

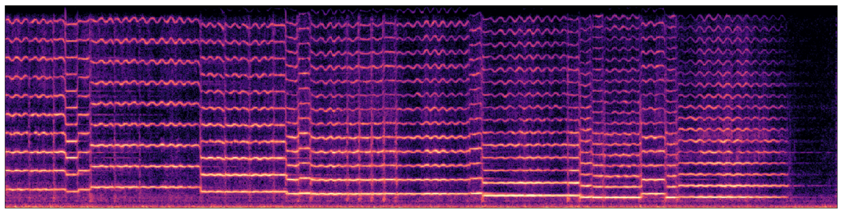


Figure 20: The synthesis audio after adjusting vibrato.

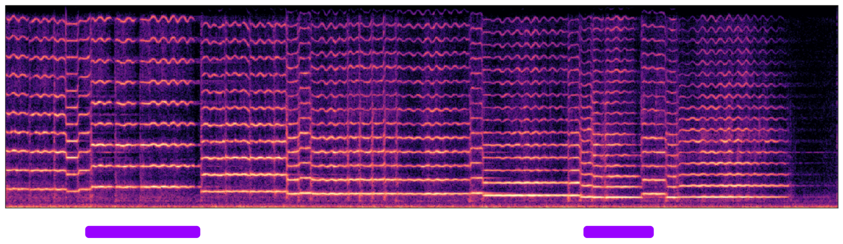


Figure 21: The synthesis audio after designing and adjusting special articulations (purple) and fine-tuning whole piece.