FAST AND FLEXIBLE NEURAL AUDIO SYNTHESIS

Lamtharn Hantrakul
Google Brain
hanoih@google.com

Jesse Engel
Google Brain
jesseengel@

Adam Roberts
Google Brain
adarob@

Chenjie Gu
Google DeepMind
gcj@

ABSTRACT

Autoregressive neural networks, such as WaveNet, have opened up new avenues for expressive audio synthesis. High-quality speech synthesis utilizes detailed linguistic features for conditioning, but comparable levels of control have yet to be realized for neural synthesis of musical instruments. Here, we demonstrate an autoregressive model capable of synthesizing realistic audio that closely follows fine-scale temporal conditioning for loudness and fundamental frequency. We find the appropriate choice of conditioning features and architectures improves both the quantitative accuracy of audio resynthesis and qualitative responsiveness to creative manipulation of conditioning. While large autoregressive models generate audio much slower than real-time, we achieve these results with a more efficient WaveRNN model, opening the door for exploring real-time interactive audio synthesis with neural networks.

1. INTRODUCTION

Expressive musical instruments, whether digital or acoustic, enable players to use relatively low-dimensional gestures to control perceptual qualities of audio such as pitch, dynamics, and timbre in real time [5, 10]. Progress in *Neural Audio Synthesis*, directly rendering audio with deep neural networks, has revolutionized the field of speech synthesis [13, 15, 18, 19] by replacing hand-designed functions with data-driven design, and is similarly poised to create a brand new class of expressive musical instruments [1, 3, 4, 6, 12].

Much of this progress is due to deep autoregressive models, such as WaveNet [18] and Tacotron [15, 19]. While these models can generate a wide range of realistic audio, using them as expressive musical instruments is not straightforward as they require fine-grained domain-specific conditioning information, such as phonemes [18] or mel-spectrograms [15] for speech, and are prohibitively slow at generating audio.

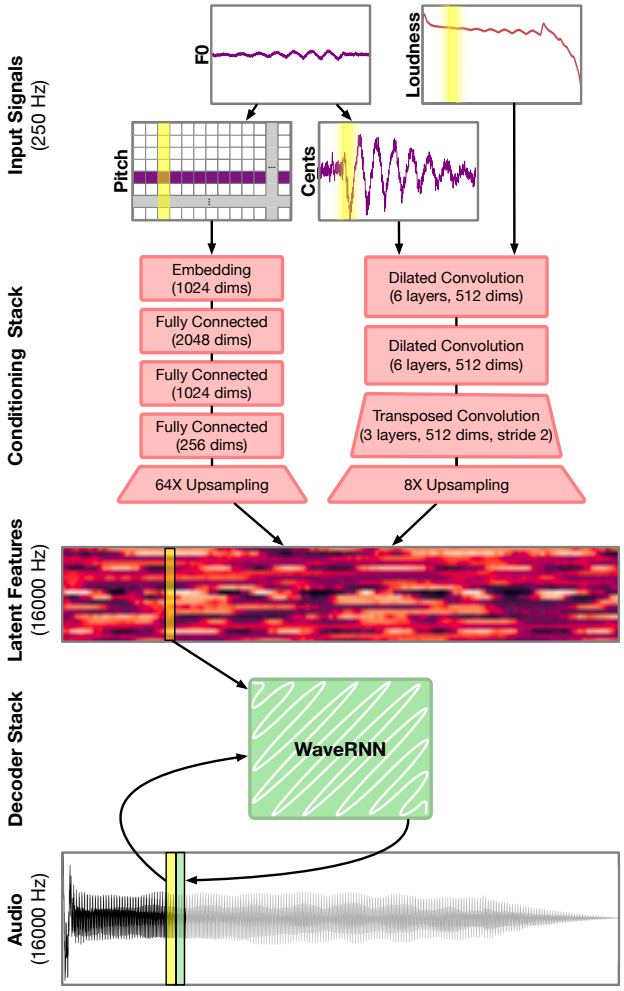

Figure 1. Real-time neural synthesizer architecture. Fundamental frequency (F0) and loudness features are fed through a conditioning stack and upsampled to audio rate. Using our Pitch/Cents representation, F0 is split into pitch one-hot embeddings and continuous deviations in cents. The latent features are concatenated and fed into a WaveRNN which generates audio autoregressively. Audio samples can be heard in the online supplement [1].

In this paper, we make progress in overcoming these challenges by using domain-specific conditioning features to drive a simpler and more efficient WaveRNN [7] model to generate audio of musical instruments. We explore a variety of conditioning features and architectures and demonstrate that a WaveRNN-based model driven by time-distributed and fine-scale musical features is capable of synthesizing realistic audio faster than real time. Our key

findings include:

- *Fine-grain control over loudness.* By conditioning on extracted loudness features, WaveRNNs can resynthesize audio that closely matches the original loudness, from long attacks and decays to fast changes like a tremolo.
- *Fine-grain control over pitch.* By conditioning on extracted fundamental frequency features, WaveRNNs can resynthesize audio that closely matches the original frequencies, from a constant pitch, to subtle vibratos and large glissandos.
- *Feature selection.* Conditioning on perceptual loudness consistently outperforms using amplitude energy, while conditioning on discrete pitches and continuous cents outperforms using a single continuous frequency feature.
- *Real-time generation*. Since the models are built around WaveRNN, even unoptimized kernels synthesize audio faster than real-time in batch mode. We show our results hold for casaul conditioning stacks, opening the door for low-latency interactivity with a properly optimized implementation.

Audio for all examples shown in this paper can be found in the online supplement [1] .

2. RELATED WORK

2.1 Musical Conditioning

WaveNet autoencoders can infer a latent conditioning signal from raw audio [4]. This enables unique opportunities for timbre transfer and morphing [2,12], but the latent code is relatively high dimensional and unintuitive to manipulate due to entanglement of perceptual attributes.

Discrete pianorolls can serve as an intuitive intermediate representation to control generation of realistic polyphonic audio in focused domains such as solo piano performance [6,9]. However, many instruments have dynamic pitch and volume that cannot be captured with discrete pianorolls. In contrast, this work examines using continuous pitch and loudness conditioning to create a synthesizer capable of such dynamic control.

2.2 Fast Synthesis

A successful approach to speeding up generation is rendering audio in parallel. Parallel Wavenet [17] distills a teacher WaveNet model into a student network that generates audio in parallel using inverse autoregressive flows. This dramatically reduces inference latency but requires training several networks and carefully tuning several heuristic losses. Other flow-based models such as WaveGLOW [14] can train audio generation flows directly, but so far have only been successful at inverting spectrograms. Generative adversarial networks provide another approach to parallel generation and GANSynth [3] recently proved four orders of magnitude faster than WaveNet baselines by generating magnitudes and phases in the spectral domain. However, these models are not capable of handling fine-scale conditioning or variable-length sequences considered here.

The streaming nature of autoregressive models makes them uniquely suited for real-time performance. WaveRNN and LPCNet [7,16] are single-layer recurrent neural networks that reduce complexity to generate 24kHz 16-bit speech at speeds up to 4×real time, even on mobile device CPUs. These models can also run inference in a streaming fashion; a critical requirement for interactive and live applications.

WaveRNN in particular, achieves its performance through a set of architectural and engineering innovations, including 1) the representation of 16-bit audio as a tuple of two 8-bit integers, which enables efficient parameterization of the neural network using a dual-softmax layer, 2) special GRU cells for the two 8-bit integers which achieves high quality synthesized audio, and 3) custom kernels on GPUs and CPUs for dense and sparse models respectively.

3. EXPERIMENTAL DETAILS

3.1 Dataset

We focus our work on a smaller subset of the NSynth dataset [4] identical to GANSynth [3]. These total 70,379 examples, comprising mostly of strings, brass, woodwinds and mallets with pitch labels within MIDI range 24-84 (F0 of ∼32-1000 Hz). Each sample is 4 seconds long and sampled at 16KHz, resulting in 64,000 dimensions.

3.2 WaveRNN

Our WaveRNN model consists of a 1280-unit GRU, two 640×512 fully connected layers (projection) and two 512×256 fully connected layers (logits). The output has two size-256 softmax layers, each predicting 8 bits of the audio (16 bits in total). The model size is on par with WaveRNN for speech synthesis [7]. Compared to other fast audio synthesis models (Parallel WaveNet [17], GANSynth [3], and WaveGlow [14]), WaveRNN has a simpler training setup: the training loss is simply negative log-likelihood and the set of hyper-parameters to tune is small.

3.3 Conditioning

Our work explores conditioning with fine-grain pitch and amplitude control. Details of the conditioning architecture are shown in Figure 1. Below we motivate the choice of representation, followed by the choice of conditioning architecture.

3.3.1 Amplitude Representation

We experiment with two representations for amplitude: Root Mean Square (RMS) *Energy* and an A-weighted Log Amplitude *Loudness*.

[1] `http://bit.ly/2GcCPNV`

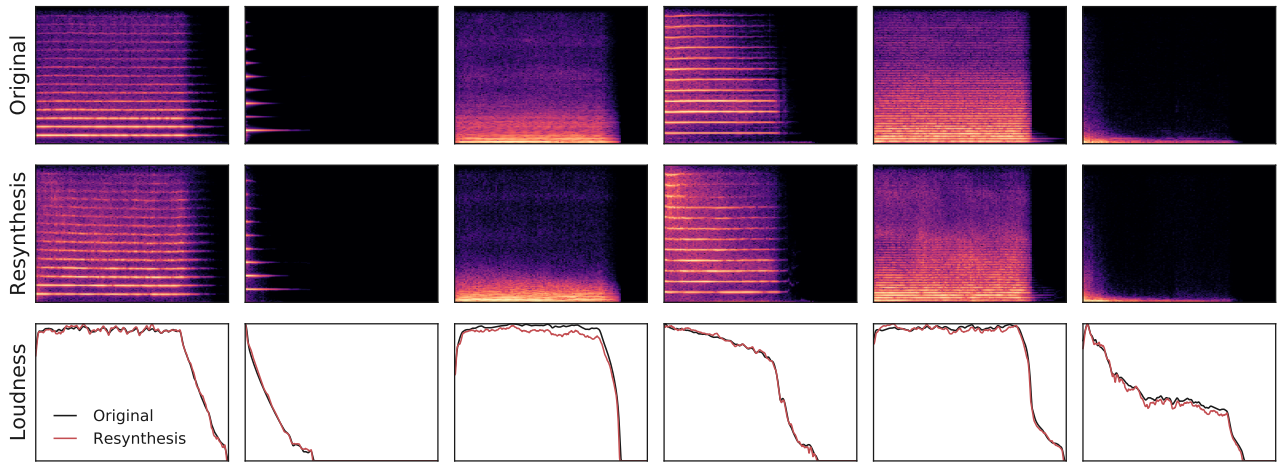

Figure 2. Audio resynthesized from extracted features. Spectrograms of examples from the NSynth dataset and and audio resynthesized from audio features (loudness and fundamental frequency) extracted from the original audio. The bottom row shows the loudness features of the original audio and resynthesized audio. Spectral and loudness contours, and fundamental frequency contours (not shown), are largely reproduced by the resynthesized audio. These samples, and those in the other Figures, were produced with the best performing Loudness + Pitch/Cents conditioned model.

Energy: Energy is computed from the STFT of the waveform with `hop_length=64` and `n_fft=2048`, yielding energy vectors of length 1000 for 4 seconds of audio (250 Hz). For training, we normalize these values across the entire dataset.

Loudness: There are many detailed psychometric models of perceived loudness [11]. For clarity, we opt for a simple A-Weighting of the power spectrum, which places greater emphasis on higher frequencies, followed by log-scaling. The loudness vector is centered and has equal length to energy. The exact computational steps are included in the Appendix.

3.3.2 Amplitude Conditioning

Wave2Midi2Wave [6] successfully conditioned a WaveNet to generate realistic piano audio based on a convolutional stack encoding a pianoroll. We adopt a similar architecture for encoding amplitude. The conditioning network consists of a stack of 12 dilated convolution layers (with dilations 1, 2, 4, 8, 16, 32, 1, 2, 4, 8, 16, 32), followed by three transposed convolutions with stride 2. All convolution layers have 512 filters with kernel size 3. For causal conditioning, we simply zero-pad and shift the receptive fields to not include future context.

3.3.3 Fundamental Frequency

We use CREPE [8], a recent and data-driven pitch tracking model with state-of-the-art performance. With hop size of 64, CREPE produces a vector of frequencies of length 1000. We convert these into MIDI pitch using librosa's `hz_to_midi()` function. From this point, we explore different representations of pitch.

Normalized Frequency: Dividing the vector of MIDI values by 127.0 yields a normalized vector representing pitch on a linear scale.

Octaves/Notes/Cents: We experiment with splitting the F0 vector into an octave one-hot $\mathbb{R}^5$, note one-hot $\mathbb{R}^{12}$ and a continuous vector of floats for cents. Each component is a vector of length 1000.

Pitches/Cents: Our most effective representation consists of a pitch one-hot $\mathbb{R}^{60}$ and a continuous vector of floats for cents. Each component is a vector of length 1000.

3.3.4 Fundamental Frequency Conditioning

For normalized frequency, we use the same convolutional stack as for amplitude features.

For Octave/Notes/Cents and Pitches/Cents, we use separate conditioning stacks for the one-hot component and continuous component (shown in Figure 1). The vector of continuous cents is encoded using the same convolutional stack as amplitude features.

For one-hot features, we use a mulitilayer perceptron (MLP) encoder network. The one-hots first pass through an embedding layer and then projected via a series of fully connected (FC) layers with RELU non-linearity into a final target dimensionality. For Octave/Notes/Cents, the two branches of encoders for octave and notes each use embedding units 1024 and FC units 2048, 1024, 512. The final octave and note activations are concatenated to produce a tensor of shape (1024, 1000). For Pitches/Cents, a single pitch encoder is used with embedding units 1024 and FC units of 2048, 1024 and 256. The final activation shape is (256, 1000). Unlike the convolutional stacks, the one-hot MLP encoding stacks are causal by definition.

3.4 Conditioning the WaveRNN

All conditioning features are time-series with lower sampling frequency (250Hz) than audio. To be used as local conditioning vectors by WaveRNN they are upsampled via

replication to 16 KHz, concatenated and added as an additional bias term in the GRU gate equations.

3.5 Training

During training, the model is tasked with resynthesizing the high dimensional audio as accurately as possible, combining the low dimensional conditioning inputs and teacher-forced previous "outputs" to autoregressively make next-step predictions. We minimize the negative log-likelihood loss of the WaveRNN's coarse and fine logits, which is computed via softmax cross-entropy.

The system was implemented in TensorFlow and each model trained on 1 Million steps. We use the ADAM optimizer with epsilon of 1e-8, momentum of 0.9, max gradient norm of 1.0 and an exponential moving average rate of 0.9999. Learning rate decay is 0.7 for every 100k steps. We fine-tuned models over batch sizes of 128, 256 and 512 and learning rates of either 9e-5 or 1e-4, and report best performing models.

4. METRICS

Resynthesis: Since log-likelihood is not a direct measure of sample quality, we quantitatively evaluate our models through the task of resynthesis. Audio features are extracted from the source audio and used to synthesize a new sound with the same features. Some examples of resynthesis are shown in Figure 2. We take the L_1 distance of extracted features between the orignal and resynthesized audio to be the fine-scale perceptual error in resynthesis. Aditionally, since the NSynth dataset has labels for pitch and instrument family (a proxy for timbre), we use these labels to train a classifer which we use to evaluate the global similarity of the resynthesis. The classifier has identical structure to the ones implemented in [4] and [3], and we provide complete details in the Appendix. We calculate each metric on resynthesized audio from the full test set of 17,600 samples.

Loudness L_1 distance: The loudness vector is extracted from the synthesized audio and L_1 distance computed against the input's conditioning loudness vector (ground truth). A better model will produce lower L_1 distances, indicating input and generated loudness vectors closely match. Note this distance is *not* back-propagated through the network as a training objective.

F0 L_1 distance: Pitch tracking using CREPE, like with any pitch tracker, is not completely reliable. Instabilities in pitch tracking, such as sudden octave jumps at low volumes, can result errors not due to model performance and need to be accounted.

CREPE outputs a useful confidence in its prediction of F0 for every frame. By examining the accuracy of pitch tracking on ground truth audio, we found that applying a confidence threshold of 0.85 filtered out areas of unreliable pitch tracking and yielded the best trade-off against false positives. Only F0 from time frames above this threshold are considered in our analysis. For models representing pitch using Octaves/Notes/Cents and Pitches/Cents, we recompute the equivalent normalized frequency. The F0 L_1 distance is reported in MIDI space for easier interpretation; an average F0 L_1 of 1.0 corresponds to a semitone difference.

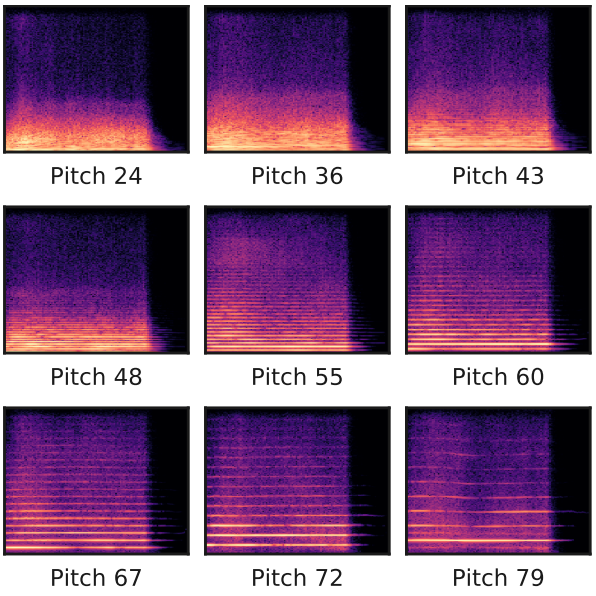

Figure 3. Synthesizing audio at different pitches with the same loudness envelope.

F0 Outliers: Audio examples with F0 confidence below 0.85 for the full length of the example indicate a failure of pitch tracking and are considered "outliers" of the measurement. 398 *ground truth* audio samples in the test set are categorized as outliers, producing a baseline for this metric of $398/17600 = 0.02$. For *generated* audio, examples completely below the 0.85 confidence threshold, are similarly removed. By inspection, we also found L_1 distances above an octave to correspond to pitch tracking failures and remove the samples as outliers. Better performing models have lower values close to the 0.02 baseline.

Pitch Error: The classifier pitch prediction gives a more global measurement of pitch correspondence during resynthesis. On ground truth samples, the classifier has a 0.06 error rate, which is the best that generated samples can hope to achieve.

Instrument Family Error: Instrument family labels are a rough measure of timbral similarity. We assume that if the original and resynthesized audio have similar timbre then they should be classified as the same instrument family. Although the classifier is not perfect, with an error rate of 0.22, it is state-of-the-art for the task and dataset, providing a rough measure of relative performance between models.

5. RESULTS AND DISCUSSION

5.1 Resynthesis

Table 1 shows the quantitative metrics for models with different conditioning on the resynthesis task. Models with causal conditioning are given in parentheses next to their non-causal equivalents.

	Signal Processing	CREPE		NSynth Classifier	
	Loudness (L_1)	**F0 (L_1)**	**F0 Outliers**	**Pitch Error**	**Family Error**
Loudness	0.14 (0.33)	6.20 (6.89)	0.57 (0.66)	0.98 (0.99)	0.75 (0.85)
Loudness + Normalized Frequencies	0.16 (0.24)	1.42 (4.17)	0.07 (0.15)	0.35 (0.81)	0.60 (0.82)
Loudness + Octave/Note/Cents	0.11 (0.14)	1.21 (1.81)	0.07 (0.08)	0.33 (0.25)	0.61 (0.78)
Loudness + Pitch/Cents	**0.10** (**0.10**)	**0.94** (**1.00**)	**0.06** (**0.07**)	**0.16** (**0.12**)	**0.53** (**0.72**)
Ground Truth	-	-	**0.02**	**0.06**	**0.22**

Table 1. Ablation study showing detailed conditioning improves resynthesis accuracy. Loudness is extracted directly from the audio as described in Section 3.3.1. CREPE is used for tracking fundamental frequency (F0), and a classifier pretrained on the NSynth dataset is used to predict pitch and instrument family (see Appendix). Models with causal convolutions in their conditioning stacks have their numbers in parentheses, while the rest use non-causal convolutions. Conditioning on F0 in Pitch/Cents tuples outperforms both Octave/Note/Cents tuples and frequencies as normalized floats, and the trend holds for both causal and non-causal conditioning.

F0 representations: A clear trend is present across all metrics: performance improves as F0 conditioning moves from Normalized Frequency, to Octave/Note/Cents, to Pitch/Cents representations. Interestingly, this improvement is not only present in the frequency-based metrics (*F0 L_1, F0 Outliers, Pitch Error*), but in the metrics for volume and timbre as well (*Loudness L_1, Family Error*).

Qualitatively, we found the models to have different failure modes. Models trained with Octaves/Notes/Cents conditioning held correct fundamental frequencies more reliably than Normalized Frequency, but would on rare occasions be completely offset by a large and erroneous interval for the length of the note. This ambiguity seems to arise from discontinuities at octave boundaries, such as between MIDI notes B2 and C3 that are close in absolute frequency. While Normalized Frequency does not suffer from this effect, we found many models trained on this representation produced audio that would dip "flat" in frequency. This could be due to the reduced effective range of input, which needs to cover the entire range of frequencies on a normalized continuous scale. Unlike speech, slight deviations in frequency are perceived as notes being out of tune.

Non-causal vs Causal: The trends in F0 conditioning are reinforced by the fact they are shared between both causal and non-causal variants of the models. Non-causal conditioning allows incorporating future information into current predictions which helps performance in almost cases. Despite this, the best performing models see less of a performance drop, which is promising for future low latency applications. Increasing the capacity of the causal encoder stack may achieve parity in performance.

Energy vs Loudness: Table 1 in the Appendix justifies the preference for loudness over energy as a conditioning signal. We compare variants of the best performing model (Loudness + Pitch/Cents vs. Energy + Pitch/Cents) and find the loudness-conditioned model outperforms the energy-conditioned model on all metrics. It is worth emphasizing that Loudness L_1 is a fair evaluation metric in this case because it is not used as a loss during training and is more aligned to human perception.

Missing Conditioning: Finally, Table 1 in the Appendix also compares a model conditioned only with Loudness to one conditioned only with Pitch/Cents. Predictably, each does better than the other on their respective metrics, and fails on the other complementary metrics, demonstrating that both levels of conditioning are required. Interestingly, the Instrument Family Error is lower for the loudness-conditioned model, indicating aspects of timbre are likely more highly correlated with loudness contours than pitch contours.

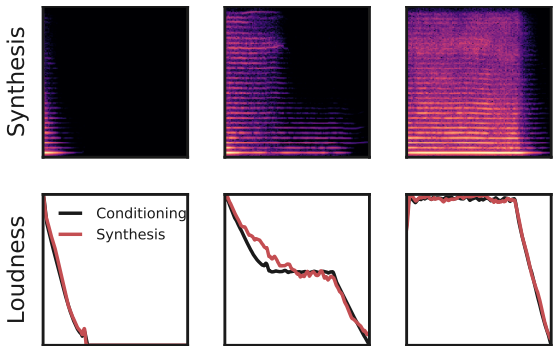

Figure 4. Interpolating in loudness conditioning.

Timbre: In the absence of timbre conditioning, the models learns to correlate timbre with pitch and loudness contours in the NSynth dataset. For example, the combination of a short decay and high F0 vector is a mallet-like sound whereas a long decay and low F0 vector is a cello-like sound. This is evidenced by the reduced Instrument Family Error in tandem with lower Loudness and F0 L_1 distances. Naively adding instrument family conditioning and spectrogram conditioning did little to improve the metrics or control of generated timbre. We believe exploring new methods of timbre control is a rich area for future research and would enable applications like interactive instrument morphing.

5.2 Creative Conditioning

We perform qualitative studies with modified out-of-dataset conditioning vectors. All examples are generated with the Loudness + Pitch/Cents model, and audio for all examples can be found in the online supplement [1] .

Interpolation of Loudness vectors: On the bottom row of of Figure 4, we show three loudness vectors. The left and right vectors were selected from the test set to

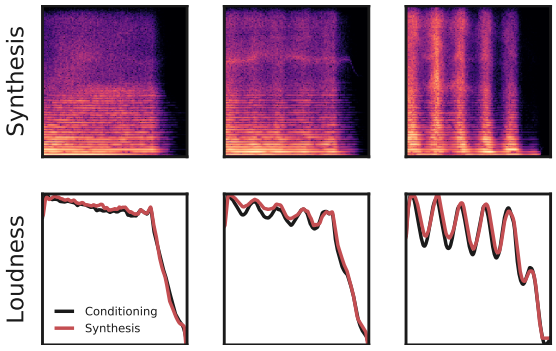

Figure 5. Applying tremolo to loudness conditioning.

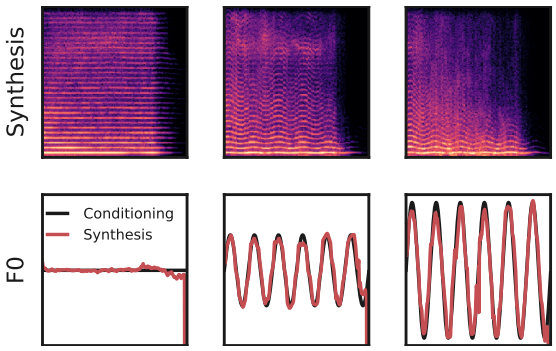

Figure 6. Applying vibrato to frequency conditioning. F0 extracted with CREPE.

demonstrate two extremes: fast decay (left) and long sustain (right). The middle is a synthetic linear interpolation of the two vectors. The top row shows spectrograms for audio synthesized by our model conditioned on each of these loudness vectors with the same F0 vector held at constant pitch. The model is able to closely adhere to the loudness curves even for the interpolated example. More importantly, the spectrogram reveals how loudness conditioning functions more than a naive "amplitude envelope", since the harmonic content changes non-linearly with the loudness signal. This rich behavior draws an analogy to the behavior of real acoustic instruments, where varying excitation introduces rich non-linear changes to the harmonic spectra based on the characteristics of the instrument.

Tremolo: Figure 5 shows the spectrogram (top) and extracted loudness vector (bottom) for an example with increasing an increasing intensity of tremolo (left-to-right) added to the original loudness vector. The generated audio closely tracks the loudness contours through diverse modulation of harmonic content. Note how the reduction in power is uneven across the frequency spectrum, dampening higher harmonics more than fundamental frequencies. A naive multiplicative tremolo would reduce power equally across all frequency bands.

Vibrato: Figure 6 shows audio synthesized from a baseline F0 vector with constant pitch. The middle vector adds a tremolo of a semitone while the rightmost vector adds a tremolo of about 2 semitones. The loudness vector is held constant in the synthesized audio. As seen by oscillations in frequency of the corresponding spectrograms, the model can generate audio reflective of increasing intensities of vibrato.

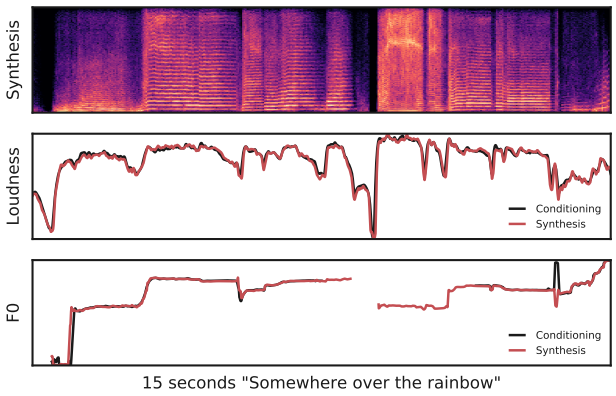

Figure 7. Re-synthesis of out-of-domain input contours extracted from a live vocalist singing "Somewhere over the rainbow". Break in F0 corresponds to silence.

Out-of-domain inputs: Figure 7 shows audio re-synthesized from conditioning signals extracted from a vocalist singing "Somewhere Over the Rainbow". The model was never trained on sequences longer than 4 seconds, nor samples with fast-moving amplitude and pitch variations such as the jump in "Some-*where*". Nonetheless, the model is able to generalize and synthesize audio tightly following these modulations. This opens the door for a variety of interactive applications. The user can provide input contours extracted from live singing, guitar playing or generate these directly from a MIDI Polyphonic Expression (MPE) controller or touchscreen interface.

5.3 Generation Speed

For this work, we draw the distinction between *"real-time throughput"* (producing x seconds of audio in wall time less than or equal to x seconds), and *"low-latency generation"* (producing audio with little to no delay from a conditioning input).

The original WaveRNN paper [7] achieved both through systems optimizations of the underlying kernels. These optimizations motivate our use of WaveRNN for future applications, but are not yet implemented in this paper. Despite this, even with unoptimized kernels, we see dramatic speedups over traditional WaveNet models and are able to achieve faster than real-time throughput speeds for batches of audio on commonly available hardware. For example, our best performing model generates 82 seconds of audio (batch size 21) in 60 seconds on an NVIDIA GTX 1080 GPU ($\sim 1.4\times$ real time).

6. CONCLUSION

In this work, we demonstrate state-of-the-art synthesis of musical instrument sounds with fine-grain temporal control over loudness and pitch. The model learns tight correlations between loudness and pitch, being able to introduce non-linear spectral modulations beyond a naive tremolo or vibrato. The comparable performance between non-causal and causal models points towards streaming applications such as a low-latency re-synthesis guitar pedal or live vocal effect.

7. REFERENCES

[1] Alexandre Défossez, Neil Zeghidour, Nicolas Usunier, Léon Bottou, and Francis Bach. SING: symbol-to-instrument neural generator. *CoRR*, abs/1810.09785, 2018.

[2] Jesse Engel. Hands on, with nsynth super. `https://magenta.tensorflow.org/nsynth-super`. Accessed: 2019-03-01.

[3] Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. Gansynth: Adversarial neural audio synthesis. *CoRR*, abs/1902.08710, 2019.

[4] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. Neural audio synthesis of musical notes with wavenet autoencoders. *CoRR*, abs/1704.01279, 2017.

[5] Neville H Fletcher and Thomas D Rossing. *The physics of musical instruments*. Springer Science & Business Media, 2012.

[6] 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. *CoRR*, abs/1810.12247, 2018.

[7] 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. *CoRR*, abs/1802.08435, 2018.

[8] 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.

[9] Rachel Manzelli, Vijay Thakkar, Ali Siahkamari, and Brian Kulis. Conditioning deep generative raw audio models for structured automatic music. In *ISMIR*, 2018.

[10] Eduardo Reck Miranda and Marcelo M Wanderley. *New digital musical instruments: control and interaction beyond the keyboard*, volume 21. AR Editions, Inc., 2006.

[11] Brian CJ Moore, Brian R Glasberg, and Thomas Baer. A model for the prediction of thresholds, loudness, and partial loudness. *Journal of the Audio Engineering Society*, 45(4):224–240, 1997.

[12] Noam Mor, Lior Wolf, Adam Polyak, and Yaniv Taigman. A universal music translation network. *CoRR*, abs/1805.07848, 2018.

[13] Wei Ping, Kainan Peng, Andrew Gibiansky, Sercan Ömer Arik, Ajay Kannan, Sharan Narang, Jonathan Raiman, and John Miller. Deep voice 3: 2000-speaker neural text-to-speech. *CoRR*, abs/1710.07654, 2017.

[14] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis. *CoRR*, abs/1811.00002, 2018.

[15] R. J. Skerry-Ryan, Eric Battenberg, Ying Xiao, Yuxuan Wang, Daisy Stanton, Joel Shor, Ron J. Weiss, Robert Clark, and Rif A. Saurous. Towards end-to-end prosody transfer for expressive speech synthesis with tacotron. In *ICML*, 2018.

[16] Jean-Marc Valin and Jan Skoglund. Lpcnet: Improving neural speech synthesis through linear prediction. *CoRR*, abs/1810.11846, 2018.

[17] Aäron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis C. Cobo, Florian Stimberg, Norman Casagrande, Dominik Grewe, Seb Noury, Sander Dieleman, Erich Elsen, Nal Kalchbrenner, Heiga Zen, Alex Graves, Helen King, Tom Walters, Dan Belov, and Demis Hassabis. Parallel wavenet: Fast high-fidelity speech synthesis. *CoRR*, abs/1711.10433, 2017.

[18] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alexander Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *Arxiv*, 2016.

[19] Yuxuan Wang, R. J. Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J. Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, Quoc V. Le, Yannis Agiomyrgiannakis, Robert Clark, and Rif A. Saurous. Tacotron: Towards end-to-end speech synthesis. In *INTERSPEECH*, 2017.