DDSP-BASED NEURAL WAVEFORM SYNTHESIS OF POLYPHONIC GUITAR PERFORMANCE FROM STRING-WISE MIDI INPUT

Nicolas Jonason
KTH Royal Institute of Technology
Stockholm, Sweden
njona@kth.se

Xin Wang
National Institute of Informatics
Tokyo, Japan
wangxin@nii.ac.jp

Erica Cooper
National Institute of Informatics
Tokyo, Japan
ecooper@nii.ac.jp

Lauri Juvela
Aalto University
Espoo, Finland
lauri.juvela@aalto.fi

Bob L. T. Sturm
KTH Royal Institute of Technology
Stockholm, Sweden
njona@kth.se

Junichi Yamagishi
National Institute of Informatics
Tokyo, Japan
jyamagis@nii.ac.jp

ABSTRACT

We explore the use of neural synthesis for acoustic guitar from string-wise MIDI input. We propose four different systems and compare them with both objective metrics and subjective evaluation against natural audio and a sample-based baseline. We iteratively develop these four systems by making various considerations on the architecture and intermediate tasks, such as predicting pitch and loudness control features. We find that formulating the control feature prediction task as a classification task rather than a regression task yields better results. Furthermore, we find that our simplest proposed system, which directly predicts synthesis parameters from MIDI input performs the best out of the four proposed systems. Audio examples and code are available.

1. INTRODUCTION

The synthesis of expressive and realistic guitar performance has been attempted in several ways. One is sample-based synthesis [1], which requires a database of purpose-made sample recordings. Another way is physical modeling synthesis [2], which requires solving systems of partial differential equations modeling the entire guitar. This paper considers a third approach: neural synthesis, a data-driven approach that does not require a purpose-made sample library or the specification of the physics of the instrument.

One family of neural synthesis techniques, termed Differentiable Digital Signal Processing (DDSP) [3], involves integrating digital signal processors into neural networks. One particular DDSP configuration combines harmonic oscillators, noise filtering and a trainable reverb [3]. This has been used to model instruments such as violin (monophonic) [3], various wind instruments [4, 5] and piano [6] with high quality from small amounts of training data.

While Engel *et al.* [3] originally controlled the networks with pitch and loudness, later work extended these models to other forms of input such as higher level expression features [5] and MIDI input [6, 5, 7, 8]. Other work has shown that one can train polyphonic DDSP models, where *voices* are rendered in parallel and then summed, by optimizing a spectral loss between the ground truth polyphonic audio and a mixture of synthesized voices [6, 9].

Guitar poses several challenges from a neural synthesis perspective. First, it is a polyphonic instrument, meaning that multiple voices can be active at the same time each with varying pitch. Guitar performance involves idiosyncrasies such as bending strings, sliding, and other articulations. In the case of bends, pitch is not discrete; and in the case of playing legato, pitch can vary within a single string excitation. Furthermore, the type, angle and location of the excitation of a string each affects the resulting sound. Unlike piano performance, where detailed transcriptions can be digitally captured (e.g., by a Disklavier [10]), obtaining objective and detailed ground truth transcription for guitar performances is difficult. For example, strummed chords often include muted strings that are sometimes omitted from the transcription [11].

Our work adapts and develops previous work in neural synthesis [3, 8] to the synthesis of acoustic guitar performance using GuitarSet [11] a dataset of hexaphonic acoustic guitar recordings with accompanying transcriptions. While some earlier work has explored neural synthesis of single guitar notes [12, 13, 14], our work is among the first to use neural networks to generate performances of guitar from string-wise MIDI input, i.e MIDI input containing one channel of MIDI data per string. Recent work by Wiggins *et al.* [15] on this topic is discussed in section 5. We train four different neural synthesizers on hexaphonic and microphone recordings of acoustic guitar performances from string-wise MIDI input, and compare them with objective metrics and subjective evaluation against natural audio and a sample-based baseline. The starting point of our study is a two-stage control-synthesis architecture that generates audio by first predicting control features and then using a synthesis model to generate audio from the predicted control features. Our first finding is that we obtain better results from treating control feature prediction as a classification task rather than as a regression task. We then use joint training of control and synthesis sub-models, which further improves performance. Finally, we present a unified architecture that merges control and synthesis sub-modules and simplifies training. Audio examples[1] and code[2] are available.

[1] https://erl-j.github.io/neural-guitar-web-supplement/
[2] https://github.com/erl-j/ddsp-guitar

2. PROPOSED SYSTEMS

We now introduce our four proposed systems, shown in Figure 1, starting with our default configuration and then its subsequent modifications.

2.1. Default configuration

Our default configuration (`ctr-syn-rg`) uses a control-synthesis architecture [8]. This architecture synthesises audio from MIDI in two stages. First, the control model takes MIDI input and predicts control features. Then, a synthesis model takes the control features and outputs an audio waveform. The control model $\mathcal{C}$ is written as

$$\hat{f}_0, \hat{l}, \hat{p}, \hat{c} = \mathcal{C}(X_{\text{pitch}}, X_{\text{vel}}, s) \tag{1}$$

It takes string-wise MIDI input represented with:

- one-hot quantized pitch $X_{\text{pitch}} \in \{0,1\}^{(n_{\text{strings}}, n_{\text{frames}}, n_{\text{pitch bins}})}$
- one-hot quantized velocity $X_{\text{vel}} \in \{0,1\}^{(n_{\text{strings}}, n_{\text{frames}}, n_{\text{vel bins}})}$
- a vector of one-hot encoded string indices $s \in \{0,1\}^{(n_{\text{strings}}, n_{\text{strings}})}$

Its output consists of predictions of four control features for each string: fundamental frequency (F0) $\hat{f}_0$, loudness $\hat{l}$, periodicity $\hat{p}$, and spectral centroid $\hat{c}$ where $\hat{f}_0, \hat{l}, \hat{p}, \hat{c} \in \mathbb{R}^{(n_{\text{strings}}, n_{\text{frames}})}$. We set $n_{\text{pitch bins}} = 305$, $n_{\text{vel bins}} = 64$, and $n_{\text{strings}} = 6$, whereas n_{frames} depends on the feature frame rate and render duration. The motivation for predicting the fundamental frequency, rather than solely relying on the MIDI pitch given by the score, is to attempt to model subtle changes in string pitch which are not explicit in the score such as those on muted strings and between string excitations. We include periodicity in order to help the synthesis model distinguish between tonal and non-tonal sections [4] and spectral centroid to provide timbral information to the synthesis model [16]. Section 3.1 details how control features are extracted from the hexaphonic audio.

The synthesis model consists of two parts, a synthesis decoder and a harmonic+noise+reverb synthesizer. The synthesis decoder $\mathcal{D}$, is written:

$$H, a, N = \mathcal{D}(\hat{f}_0, \hat{l}, \hat{p}, \hat{c}, s) \tag{2}$$

The synthesis decoder takes the string-wise predicted control features from $\mathcal{C}$ and outputs three synthesis parameters;

- harmonic amplitudes $H \in \mathbb{R}^{(n_{\text{strings}}, n_{\text{frames}}, n_{\text{harmonics}})}$
- global harmonic amplitude $a \in \mathbb{R}^{(n_{\text{strings}}, n_{\text{frames}})}$
- filtered noise band amplitudes $N \in \mathbb{R}^{(n_{\text{strings}}, n_{\text{frames}}, n_{\text{noise bands}})}$

These synthesis parameters are then fed to a 6-voice harmonic + noise + reverb synthesizer $\mathcal{H}$, the outputs of which are summed to predict the final waveform $y \in \mathbb{R}^{(n_{\text{samples}})}$.

Figure 2 details the control-synthesis architecture. The control model $\mathcal{C}$ consists of a neural network combining bi-directional long short-term memory (LSTM) [17] operating across the time dimension and scaled dot-product self-attention [18] operating across the string dimension to account for inter-string dependencies. The use of bi-directionality is motivated by the fact that certain aspects of a guitar performance requires looking at future notes. For example, in order to render the sound of fingers sliding across the strings before a chord change, we need to know that a chord change is coming up. The synthesis decoder $\mathcal{D}$ uses a bi-directional LSTM-based neural network [3]. The harmonic + noise synthesizer uses 128 harmonics and 128 noise filter bands, with the same design as in [3]. While acoustic guitar does exhibit inharmonicity due to string stiffness, the inharmonicity is difficult for the average listener to perceive [19], and we leave the inclusion of inharmonicity to future work. We use one trainable reverb per string to account for differences in the position of each string in relation to the guitar body and the microphone. Each reverb has a 0.25-second trainable impulse response. With the exception of the self-attention across strings, all network layers process the strings independently. The control model and synthesis decoder use a hidden-layer size of 512. The control model, synthesis decoder and trainable reverb have 53.7M, 18.2M, and 72k parameters respectively, totalling 72M parameters.

In the default configuration, we train the control and synthesis models separately. First, we train the synthesis model to minimize the multi-scale spectral loss [20, 3] (MSSL) between natural microphone audio y and audio synthesized from ground truth control features y':

$$\mathcal{L}_{syn} = \text{MSSL}(y, y') \tag{3}$$

Secondly, we train the control model to predict control features from MIDI input. In line with earlier work [4, 5, 7, 8], the default configuration treats the prediction of control features as a regression task. The loss, notated $\mathcal{L}_{\text{rg}}$, is based on the mean squared error (MSE) of the predicted control features:

$$\mathcal{L}_{\text{rg}} = \mathcal{L}_{f_0} + \mathcal{L}_l + \mathcal{L}_p + \mathcal{L}_c \tag{4}$$

$$\mathcal{L}_{f_0} = \sum_{i=1}^{n_{\text{strings}}} \sum_{t=1}^{T} (f_0(i,t) - \hat{f}_0(i,t))^2 \cdot l(i,t) \cdot p(i,t) \tag{5}$$

$$\mathcal{L}_l = \sum_{i=1}^{n_{\text{strings}}} \sum_{t=1}^{T} (l(i,t) - \hat{l}(i,t))^2 \tag{6}$$

$$\mathcal{L}_p = \sum_{i=1}^{n_{\text{strings}}} \sum_{t=1}^{T} (p(i,t) - \hat{p}(i,t))^2 \cdot l(i,t) \tag{7}$$

$$\mathcal{L}_c = \sum_{i=1}^{n_{\text{strings}}} \sum_{t=1}^{T} (c(i,t) - \hat{c}(i,t))^2 \cdot l(i,t) \tag{8}$$

For string $i \in [1, n_{\text{strings}}]$ and feature frame $t \in [1, T]$ the target loudness l is used to weight the MSE for f_0, p, c according to signal strength thus discounting prediction errors in quieter sections. We also discount the F0 loss in non-pitched sections, by weighting the f_0 MSE by target periodicity.

2.2. Control feature prediction as a classification task

After initial experiments with the default configuration produced poor results, we experimented with treating control feature prediction as a classification task rather than a regression task since this has been shown to yield better performance in other contexts [21]. We refer to this system as `ctr-syn-cl`.

We quantize and one-hot encode the four control features with $n_{f_0 \text{ bins}} = 305$, and $K = 64$ bins for l, p, c. We denote the new, one-hot encoded features as $F_0 \in \{0,1\}^{(n_{\text{strings}}, n_{\text{frames}}, n_{f_0 \text{ bins}})}$, $L, P, C \in \{0,1\}^{(n_{\text{strings}}, n_{\text{frames}}, K)}$. The new control model is written:

$$\hat{F}_0, \hat{P}, \hat{C}, \hat{L} = \mathcal{C}_{\text{cl}}(X_{\text{pitch}}, X_{\text{vel}}, s) \tag{9}$$

It outputs probabilities for each control feature over its quantization bins. Control features are then generated by argmax sampling over the estimated quantization bin probabilities. For training, we

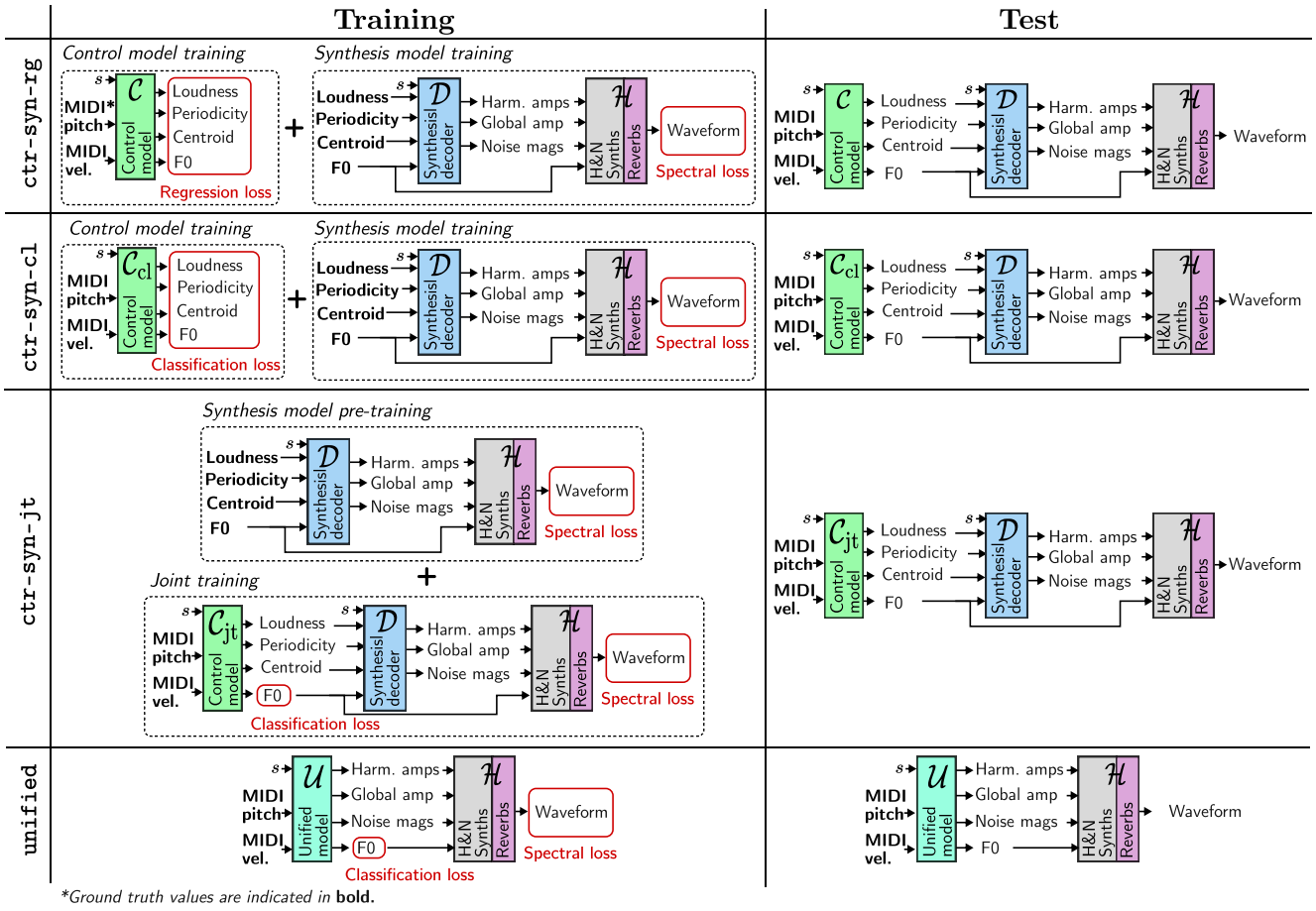

Figure 1: Overview of the four proposed systems. `ctr-syn-rg` is a control-synthesis model with control feature regression where the control and synthesis models are trained separately. `ctr-syn-cl` is a control-synthesis model with control feature classification where the control and synthesis are trained separately. `ctr-syn-jt` trains a control feature classifier jointly with a pre-trained synthesis model. `unified` merges the control model and synthesis decoder into a single network.

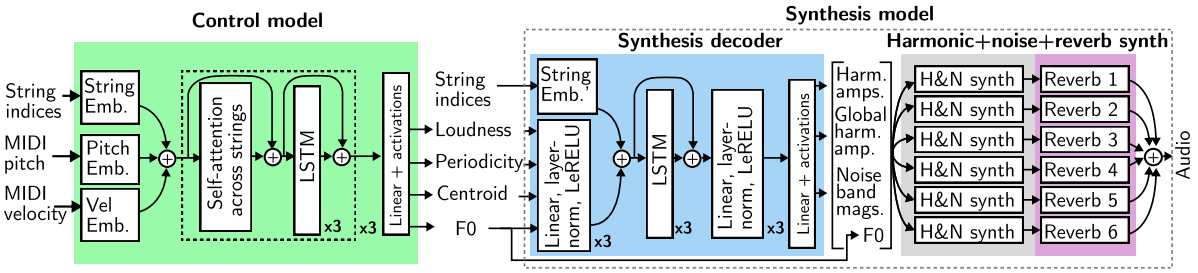

Figure 2: Details of the control-synthesis architecture

define the feature classification loss $\mathcal{L}_{\text{cl}}$ on weighted negative log probabilities.

$$\mathcal{L}_{\text{cl}} = \mathcal{L}_{F_0} + \mathcal{L}_L + \mathcal{L}_P + \mathcal{L}_C \tag{10}$$

$$\mathcal{L}_{F_0} = -\sum_{i=1}^{n_{\text{strings}}} \sum_{t=1}^{T} \sum_{b=1}^{n_{f_0\text{ bins}}} F_0(i,t,b) \log(\hat{F}_0(i,t,b)) \cdot l(i,t) \cdot p(i,t) \tag{11}$$

$$\mathcal{L}_L = -\sum_{i=1}^{n_{\text{strings}}} \sum_{t=1}^{T} \sum_{b=1}^{K} L(i,t,b) \log(\hat{L}(i,t,b)) \tag{12}$$

$$\mathcal{L}_P = -\sum_{i=1}^{n_{\text{strings}}} \sum_{t=1}^{T} \sum_{b=1}^{K} P(i,t,b) \log(\hat{P}(i,t,b)) \cdot l(i,t) \tag{13}$$

$$\mathcal{L}_C = -\sum_{i=1}^{n_{\text{strings}}} \sum_{t=1}^{T} \sum_{b=1}^{K} C(i,t,b) \log(\hat{C}(i,t,b)) \cdot l(i,t). \tag{14}$$

`ctr-syn-cl` has 72.3M parameters with 54M belonging to $\mathcal{C}_{\text{cl}}$.

2.3. Joint training of control and synthesis sub-modules

Past work has shown that joint training of sub-modules can lead to better performance for neural waveform synthesis of piano sounds [22]. We therefore propose a system where the control model is jointly trained together with a pre-trained synthesis model. We refer to this system as `ctr-syn-jt`. The new control model is written:

$$\hat{F}_0, \hat{l}, \hat{p}, \hat{c} = \mathcal{C}_{\text{jt}}(X_{\text{pitch}}, X_{\text{vel}}, s) \tag{15}$$

During joint training, we generate a predicted F0 contour $\hat{f}_0$ by argmax sampling of $\hat{F}_0$, which is passed to the synthesis model to generate a waveform $\hat{y}$. The loss used during the joint training is:

$$\mathcal{L}_{\text{jt}} = \mathcal{L}_{F_0} + \text{MSSL}(y, \hat{y}) \tag{16}$$

where y is the natural microphone audio and $\hat{y}$ is the audio waveform synthesized from predicted control features. The loss $\mathcal{L}_{\text{jt}}$ does not apply direct supervision to the predictions of p, l and c to give the model flexibility in terms of the information passed from the control model. However, we keep supervision of F0 since it has been shown to be difficult to tune the frequency of an oscillator with gradient descent using a point-wise spectral loss [23, 24]. `ctr-syn-jt` has 72.2M parameters in total with 53.9M belonging to $\mathcal{C}_{\text{jt}}$.

2.4. Unified model

Following promising results from initial experiments with joint training of control and synthesis model, we simplify our approach

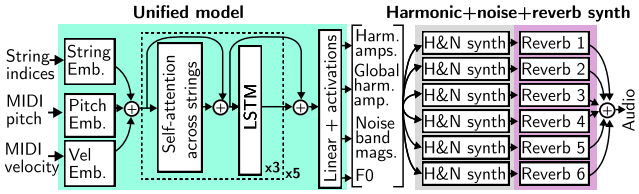

Figure 3: Details of the unified model architecture.

by merging the control model and synthesis decoder into a single network $\mathcal{U}$ that predicts synthesis parameters directly from the MIDI input and is shown in Figure 3. We refer to this system as `unified`. We write $\mathcal{U}$ as:

$$\hat{F}_0, H, a, N = \mathcal{U}(X_{\text{pitch}}, X_{\text{vel}}, s) \tag{17}$$

.

The unified model is trained jointly with the trainable reverb to minimize the sum of the multi-scale spectral loss and the F0 classification loss:

$$\mathcal{L}_u = \mathcal{L}_{F_0} + \text{MSSL}(y, \hat{y}) \tag{18}$$

`unified` has 89.8M parameters with 89.7M belonging $\mathcal{U}$.

3. EXPERIMENTS

3.1. Experimental conditions

Dataset: We use the *GuitarSet* dataset [11], which contains audio and pitch annotations for 360 acoustic guitar performances, totalling just over 3 hours. GuitarSet encompasses six different guitar players each playing solo and accompaniment for three chord progressions in five different styles in two different tempos in random keys. The performances are recorded with both a microphone as well as a hexaphonic pickup. A hexaphonic pickup is a guitar pickup that produces one audio channel for each guitar string. It is important to note, however, that the correspondence between channels and strings is imperfect as the pickups also capture some signal from neighbouring strings. This phenomenon is called bleed. To reduce bleed, the GuitarSet authors provide hexaphonic recordings processed with the KAMIR [25] bleed removal algorithm. MIDI pitch annotation, extracted semi-automatically from the hexaphonic recordings, is included in GuitarSet. Importantly, the pitches of the MIDI pitch annotation are not quantized to semitones. Since velocity annotation is not included in GuitarSet, we use the peak unit scaled dB(A) loudness of every note as a proxy for MIDI velocity [8].

We split the dataset such that no recordings from the same performer-progression-style triple occur in both partitions. Splitting it randomly instead risks having recordings of a specific performer playing the same progression in the same style appearing in both test and development splits. To limit author influence on which recordings end up in each split, we assign random aliases to each player, progression and style prior to splitting. Using the aliases, we select 36 recordings for testing while balancing for diversity of players, progressions and styles. We use the remaining 324 recordings for model development, from which we randomly choose 306 for training, and keep the remaining 18 recordings for validation.

Feature extraction: Ground truth values for the control features are extracted from KAMIR-processed hexaphonic recordings [11]. We compute F0 and periodicity using a PyTorch implementation of *CREPE* [26, 27] and scale F0 with MIDI spacing [3] to $[0, 1)$ where 0 corresponds to $(35Hz)$ and 1 is $(1200Hz)$. We measure loudness with A-weighting using *librosa* [28] and scale it to $[0, 1)$ where 0 is -80 dB and 1 is 0 dB [3]. Finally, we compute spectral centroid with *librosa* and divide it by the Nyquist frequency.

Model training and inference: All training uses the ADAM optimizer [29], a learning rate decay of 0.99 and early stopping with a patience of 5 epochs. We train one synthesis model that is shared by `ctr-syn-rg`, `ctr-syn-cl` and `ctr-syn-jt`, using a learning rate of 3×10^{-4}. Control models for `ctr-syn-rg` and `ctr-syn-cl` are trained using a learning rate of 1×10^{-4}. Joint training of control and synthesis model in `ctr-syn-jt` uses a learning rate of 1×10^{-4}. The `unified` system's training uses a learning rate of 1×10^{-4}. Training excerpts are 8 seconds in duration, extracted from recordings at random time offsets. Both input features and control features have a feature frame rate of 128 Hz. Audio is generated at 48 kHz. Computing the MSSL uses window sizes $[192, 384, 768, 1526, 3072, 6144, 12288]$. In order to render full recordings from the test set, we window the conditioning into 8-second windows with a 4-second skip length and mix the resulting windowed audio with 100 ms linear crossfade starting 2 seconds into the preceding window.

3.2. Evaluation

We now perform objective and subjective evaluation of the proposed systems. We also include audio produced from the pretrained synthesis model that is used by `ctr-syn-rg`, `ctr-syn-cl` and `ctr-syn-jt` given ground truth control features, denoted as `oracle-syn`. The `oracle-syn` baseline system is included to test the performance of the synthesis model in isolation and assumes access to the ground truth control features which is not the case in the MIDI-to-audio use-case. The subjective evaluation also includes natural audio as well as renders of the MIDI with the free commercial sample-based guitar synthesizer *Ample Guitar lite* [30].

Objective evaluation: While rendering the test samples, we compute the MSSL between natural and synthesized audio for each 8-second window. The average MSSL across all 8-second clips for all recordings is shown in Table 1. To evaluate pitch accuracy, we follow a series of steps. We first extract F0 estimates for each string from the synthesized string channels using CREPE, and then quantize them into semitones. Subsequently, we filter out frames in which there are no active input MIDI notes on the corresponding string. Finally, we compare these semitone estimates to two

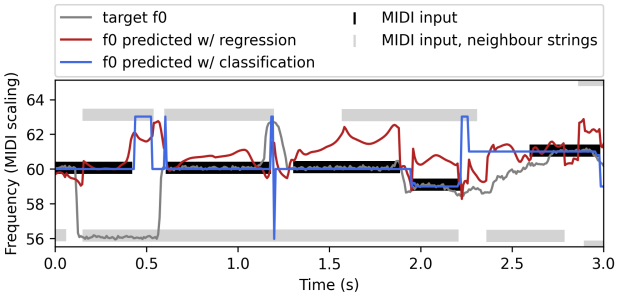

Figure 4: Fourth string predicted F0 from `ctr-syn-rg` and `ctr-syn-cl` against the target F0 (CREPE) and input MIDI pitch. Also shown is MIDI input from strings 3 and 5. In the first half-second we see the target F0 erroneously jump to the pitch of the 3rd string. We also see high inaccuracy in the regression-based F0 prediction with respect to both target F0 and MIDI input pitch.

Table 1: Objective and subjective evaluation results.

system	MSSL ↓	CREPE acc.↑	MIDI acc.↑	MOS↑
`natural`	-	-	-	4.10
`ample-gtr`	-	-	-	4.08
`oracle-syn`	6.21	0.94	0.94	3.08
`ctr-syn-rg`	10.51	0.33	0.34	1.18
`ctr-syn-cl`	8.02	0.89	0.96	2.64
`ctr-syn-jt`	7.64	0.89	0.97	3.00
`unified`	7.71	0.90	0.97	3.38

reference sources: the string-wise semitone-quantized F0 values estimated by CREPE from the natural hexaphonic audio and the string-wise semitone-quantized input MIDI pitch data. The resulting average accuracies, denoted `CREPE acc.` and `MIDI acc.` are shown in Table 1.

Subjective evaluation: We conducted an online listening test. Since the durations of the full recordings (18 to 44 seconds) are considerably longer than what is typically presented in a listening test, we segment all audio into halves and quarters, giving us 216 segments with durations ranging from 4 to 22 seconds. We recruited 66 unique listeners who self-reported having experience playing guitar. Each listener rated either one or two sets of 108 samples, totalling 70 sets, balanced to have approximately the same number of samples for each system. The listeners were asked to rate the samples on a 5-point scale (very bad, bad, acceptable, good, very good), taking into account sound quality, naturalness and appropriateness of pitch. We perform a statistical analysis of the system scores using a two-sided Mann-Whitney U test [31] with Bonferroni correction with $\alpha = 0.05$. The last column of Table 1 shows the mean opinion scores (MOS) of the systems. According to the two-sided Mann-Whitney U test with Bonferroni correction, there are statistically significant differences in the MOS ratings among all pairs of systems, except for two pairs: `natural` and `ample-gtr`, as well as `oracle-syn` and `ctr-syn-jt`. We find that `ctr-syn-rg` performs the worst of all systems with an overall MOS of 1.18. The next best is `ctr-syn-rg` with an overall MOS of 2.64. After that `ctr-syn-jt` obtains a MOS of 3.00. Finally, `unified` performs the best out of all four proposed systems with a MOS of 3.38, even outperforming `oracle-syn`. This is lower than both `ample-gtr` and `natural`, which obtain a MOS of 4.08 and 4.10.

4. DISCUSSION

Out of the proposed systems, we find that `ctr-syn-rg` shows the worst spectral loss whereas the best spectral losses are obtained by `ctr-syn-jt` and `unified`. However, the MSSL for `oracle-syn` is much better than that of our best proposed systems.

In the terms of pitch accuracy, we observe that `ctr-syn-rg`, which predicts F0 with regression, performs poorly in semitone pitch accuracy while systems that predict F0 with classification (`ctr-syn-rg`, `ctr-syn-cl`,`unified`) obtains far higher semitone pitch accuracies. Based on visual inspection of F0 predictions, shown in Figure 4, we hypothesize that the failure of regression to accurately predict F0 originates from errors in the target F0 contours caused by bleed from neighbouring strings. In attempting to minimize the MSE to the flawed target, the regression model places its prediction between the pitch of the target string and the active pitches from neighbouring strings. We also observe that the classification-based F0 predictions are closer in semitone accuracy to the MIDI pitch input than they are to the CREPE contour, which they are trained to predict. We also observe that `oracle-syn` obtains less than 1.0 in terms of pitch accuracy with respect to the CREPE F0 contour it receives as input. Explanations for this could include variablity in the CREPE predictions in frames with lower periodicity (such as during onsets) and frames affected by bleed.

The subjective evaluation results indicate, first, that all of the proposed improvements have resulted in audible improvements of the synthesized guitar performance. Secondly, we did not observe any significant differences in MOS between the sample-based baseline and the natural audio. This can possibly be attributed to the fact that the natural audio comes from an academic dataset intended for music transcription research rather than listening. Therefore, the recording conditions, post-processing and precision of tuning and intonation may not have been prioritized in the recording of GuitarSet to the same extent as in the recording of the sample library used by `ample-gtr`. It is also intriguing that `unified` had a higher MOS rating than `oracle-syn`. One explanation for this could be that `unified` generates more stable pitch contours than the noisy pitch contours from CREPE used by `oracle-syn` as can be seen in Figure 4. Furthermore, although the proposed `unified` system was not as good as the sample-based system `ample-gtr`, we can see that it was able to synthesize guitar sounds of reasonable quality despite being trained on only a few hours of data.

The fact that `unified` obtained the highest MOS suggests that using intermediate control features hurts synthesis quality compared to predicting synth parameters end-to-end from MIDI. However, these control features might still have the benefit of offering interesting control surfaces for a user [5].

Additionally, the fact that the proposed systems whose F0 contour was closest to the MIDI pitch contour raises the question whether we can use the input MIDI pitch as a proxy for F0 rather than the predicted F0 on frames where notes are active. We provide an option to snap the F0 contour to the MIDI pitch at inference time in our code.

5. COMPARISON WITH SIMILAR WORK

Recent work by Wiggins et al. [15] also proposed a different architecture for DDSP based guitar synthesis. The most important differences in architecture is their use of an inharmonic oscillator and using MIDI pitch directly as the pitch contour during training. It is important to note that while both works use the GuitarSet [11] dataset, [15] uses a different data splitting scheme resulting in less data used for training; they use 6 minutes of training data while we use about 2.5 hours of training data. Additionally, they target 16kHz while we target 48kHz. With these differences in mind, we invite the reader to listen to their audio examples [3]. Future work will seek to make meaningful comparisons between the two approaches across various training data sizes and sampling rates.

6. CONCLUSION

We have developed four different DDSP-based neural waveform synthesis systems for polyphonic guitar performance from string-wise MIDI input, and compared them with both objective metrics and subjective evaluation against natural audio and a sample-based baseline.

We find that switching from control feature regression to control feature classification improves performance considerably. Furthermore, we find that joint training of control and synthesis submodules further improves the synthesis quality. We have also suggested a simplification of the control synthesis architecture, which further improves synthesis quality.

Although the proposed systems were not as good as the sample-based baseline, our `unified` system is able to synthesize guitar performance of reasonable quality with only a few hours of data.

7. LIMITATIONS & FUTURE WORK

We will now discuss limitations of our work and how these might be adressed in future work. Our proposed approach relies on a manually MIDI transcriptions and on hexaphonic audio being available in order to extract F0 contours for each string. The need for a transcription can potentially be met by an automatic transcription system. Similarly, the requirement for hexaphonic data for extracting F0 contours might be addressed with multi-pitch estimation. Additionally, the errors in the target F0 contours derived from the hexaphonic data could possibly be reduced with additional processing of the CREPE predictions.

Our work did not investigate using MIDI pitch as a proxy for F0 during training as done by [15]. Future work could investigate how using the MIDI pitch as a proxy for F0 contour during training affects synthesis quality across various training data set sizes.

Finally, since our proposed systems uses bi-directional layers, they are not suitable for real-time applications. Future work could investigate the viability of using a causal model instead.

8. ACKNOWLEDGMENTS

This work is supported by JST CREST Grants (JPMJCR18A6 and JPMJCR20D3) and MEXT KAKENHI Grants (21K17775, 21H04906, 21K11951, 22K21319) as well as MUSAiC: Music at the Frontiers of Artificial Creativity and Criticism (ERC-2019-COG No. 864189).

[3] These can be found https://twilight-bougon-45a.notion.site/A-Differentiable-Acoustic-Guitar-Model-for-String-Specific-Polyphonic-Synthesis-ee24cf07004f44a7aef78302be2621ea

9. REFERENCES

[1] Diemo Schwarz, "Concatenative sound synthesis: The early years," *Journal of New Music Research*, vol. 35, pp. 3–22, 03 2006.

[2] Mikael Laurson, Cumhur Erkut, Vesa Välimäki, and Mika Kuuskankare, "Methods for Modeling Realistic Playing in Acoustic Guitar Synthesis," *Computer Music Journal*, vol. 25, no. 3, pp. 38–49, 2001, Publisher: The MIT Press.

[3] Jesse Engel, Chenjie Gu, Adam Roberts, and others, "DDSP: Differentiable Digital Signal Processing," in *International Conference on Learning Representations*, 2019.

[4] Nicolas Jonason and Bob Sturm, "Neural music instrument cloning from few samples," in *25th International Conference on Digital Audio Effects (DAFx20in22), Vienna, Austria, September 2022*, 2022.

[5] Yusong Wu, Ethan Manilow, Yi Deng, Rigel Swavely, Kyle Kastner, Tim Cooijmans, Aaron Courville, Cheng-Zhi Anna Huang, and Jesse Engel, "MIDI-DDSP: Detailed Control of Musical Performance via Hierarchical Modeling," in *International Conference on Learning Representations*, 2021.

[6] Lenny Renault, Rémi Mignot, and Axel Roebel, "DDSP-Piano: a Neural Sound Synthesizer Informed by Instrument Knowledge," *AES - Journal of the Audio Engineering Society Audio-Accoustics-Application*, 2023, Publisher: Audio Engineering Society Inc.

[7] Rodrigo Castellon, Chris Donahue, and Percy Liang, "Towards realistic midi instrument synthesizers," in *NeurIPS Workshop on Machine Learning for Creativity and Design*, 2020.

[8] 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.

[9] Masaya Kawamura, Tomohiko Nakamura, Daichi Kitamura, Hiroshi Saruwatari, Yu Takahashi, and Kazunobu Kondo, "Differentiable Digital Signal Processing Mixture Model for Synthesis Parameter Extraction from Mixture of Harmonic Sounds," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2022, pp. 941–945, ISSN: 2379-190X.

[10] 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.

[11] Qingyang Xi, Rachel M Bittner, Johan Pauwels, Xuzhou Ye, and Juan Pablo Bello, "GuitarSet: A Dataset for Guitar Transcription.," in *ISMIR*, 2018, pp. 453–460.

[12] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Mohammad Norouzi, Douglas Eck, and Karen Simonyan, "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders," in *Proceedings of the 34th International Conference on Machine Learning*. July 2017, pp. 1068–1077, PMLR, ISSN: 2640-3498.

[13] The Sound of AI Community, "From Words to Sound: Neural Audio Synthesis of Guitar Sounds with Timbral Descriptors," Sept. 2022, Publication Title: Proceedings of the 3rd Conference on AI Music Creativity Publisher: AIMC.

[14] Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts, "GANSynth: Adversarial Neural Audio Synthesis," in *International Conference on Learning Representations*, 2018.

[15] Andrew Wiggins and Youngmoo Kim, "A Differentiable Acoustic Guitar Model for String-Specific Polyphonic Synthesis," in *2023 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, New Paltz, NY, USA, Oct. 2023, pp. 1–5, IEEE.

[16] Emery Schubert, Joe Wolfe, and Alex Tarnopolsky, "Spectral centroid and timbre in complex, multiple instrumental textures," Aug. 2004.

[17] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, "Attention Is All You Need," Dec. 2017, arXiv:1706.03762 [cs].

[19] Matti Karjalainen and Hanna Järveläinen, "Is inharmonicity perceivable in the acoustic guitar?," in *Proc. of Forum Acusticum*, 2005, vol. 2005.

[20] 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*, vol. 28, pp. 402–415, 2019, Publisher: IEEE.

[21] Shihao Zhang, Linlin Yang, Michael Bi Mi, Xiaoxu Zheng, and Angela Yao, "Improving Deep Regression with Ordinal Entropy," in *The Eleventh International Conference on Learning Representations*, 2022.

[22] Xuan Shi, Erica Cooper, Xin Wang, Junichi Yamagishi, and Shrikanth Narayanan, "Can Knowledge of End-to-End Text-to-Speech Models Improve Neural Midi-to-Audio Synthesis Systems?," in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2023, pp. 1–5, IEEE.

[23] Joseph Turian and Max Henry, "I'm Sorry for Your Loss: Spectrally-Based Audio Distances Are Bad at Pitch," in *"I Can't Believe It's Not Better!"NeurIPS 2020 workshop*, 2020.

[24] Ben Hayes, Charalampos Saitis, and György Fazekas, "Sinusoidal Frequency Estimation by Gradient Descent," in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, June 2023, pp. 1–5.

[25] Thomas Pratzlich, Rachel M. Bittner, Antoine Liutkus, and Meinard Muller, "Kernel Additive Modeling for interference reduction in multi-channel music recordings," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, South Brisbane, Queensland, Australia, Apr. 2015, pp. 584–588, IEEE.

[26] 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)*. 2018, pp. 161–165, IEEE.

[27] Max Morrison, "torchcrepe," 2022.

[28] Brian McFee, Colin Raffel, Dawen Liang, D. Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto, "librosa: Audio and Music Signal Analysis in Python," in *Processings of the 14th Python in Science Conference*, 2015.

[29] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[30] AmpleSound, "Ample Guitar M Lite," 2015, AmpleSound.

[31] *Mann–Whitney Test*, pp. 327–329, Springer New York, New York, NY, 2008.