A COMPARISON OF VIRTUAL ANALOG MODELLING TECHNIQUES FOR DESKTOP AND EMBEDDED IMPLEMENTATIONS

Jatin Chowdhury

Center for Computer Research in Music and Acoustics
Stanford University
Palo Alto, CA
jatin@ccrma.stanford.edu

ABSTRACT

We develop a virtual analog model of the Klon Centaur guitar pedal circuit, comparing various circuit modelling techniques. The techniques analyzed include traditional modelling techniques such as nodal analysis and Wave Digital Filters, as well as a machine-learning technique using recurrent neural networks. We examine these techniques in the contexts of two use cases: an audio "plug-in" designed to be run on a consumer-grade desktop computer, and a guitar pedal-style effect running on an embedded device. Finally, we discuss the advantages and disdvantages of each technique for modelling different circuits, and targeting different platforms.

1. INTRODUCTION

The Klon Centaur is an overdrive guitar pedal designed by Bill Finnegan in the early 1990's, that has developed cult acclaim amongst guitarists [1]. The circuit is notable for producing "transparent distortion" [2], a term used to describe the way the pedal seems to add distortion to a guitar's sound without otherwise affecting the tone. While the original manufacturing run of the pedal ended in 2004, many "clones" of the pedal have been produced by other manufacturers, adding to its cult following.

Circuit modelling is typically broken down into "white-box" and "black-box" approaches [3]. A "white-box" approach uses knowledge of the internal mechanisms of the circuit, often modelling the physical interactions of the electrical components. Popular white-box methods include nodal analysis [4], Port-Hamiltonian analysis [5], Wave Digital Filters [6, 7], and nonlinear state space analysis [8].

"Black-box" circuit modelling methods generally use measurements taken from the circuit being modelled and attempt to model the response of the circuit without knowledge of the internal workings of the system. Traditional black-box techniques include impulse response measurements [9] and extensions thereof, including the Weiner-Hammerstein method [3]. Recently, researchers have begun using machine learning methods for black-box modelling. In [10], the authors use a WaveNet style architecture to generate an output signal sample-by-sample. In [11] the authors use a deep fully-connected networks to approximate nonlinear state-space solutions for a circuit, effectively a "grey-box" approach. Finally, in [12] the authors use a recurrent neural network to model the behavior of audio circuits with control parameters.

The structure of the paper will be as follows: in §2 we give background information on circuit modelling using nodal analysis and Wave Digital Filters. §3 describes the use of recurrent neural networks for circuit modelling, and outlines the model and training process used for emulating the "Gain Stage" circuit from the Klon Centaur. In §4 we discuss the real-time implementation of a complete emulation of the Klon Centaur pedal using the methods outlined in the previous sections. §5 shares the results of Klon Centaur emulation as well as recommendations for circuit modelling using the methods discussed here.

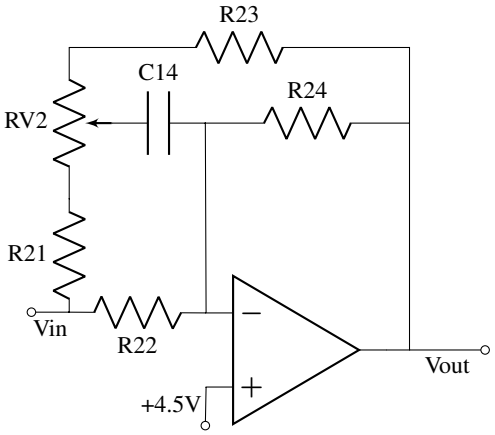

Figure 1: *Klon Centaur Tone Control Circuit*

2. TRADITIONAL CIRCUIT MODELLING TECHNIQUES

First, we examine the use of traditional circuit modelling techniques, specifically nodal analysis and Wave Digital filters, using sub-circuits from the Klon Centaur as examples.

2.1. Nodal Analysis

The process for creating a digital model of a circuit using nodal analysis is as follows:

1. Convert the circuit into the Laplace domain.
2. Form a Laplace domain transfer function of the circuit.
3. Use a conformal map to transform the circuit into the digital domain.

As an example circuit, we examine the Tone Control circuit from the Klon Centaur (see fig. 1). The first step is to convert the circuit into the Laplace domain, using the Laplace variable $s = j\omega$. The

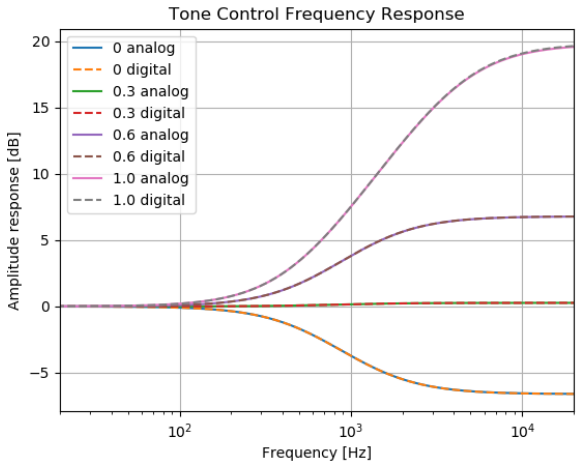

Figure 2: *Tone control frequency response at various values of the Treble parameter, comparing the responses of the analog filter with the digital model.*

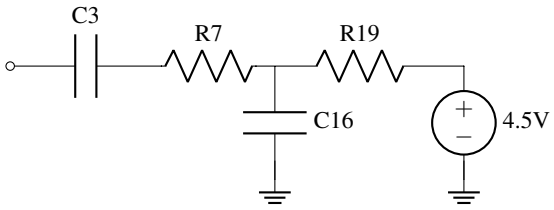

Figure 3: *Klon Centaur Feed-Forward Network 1 Circuit*

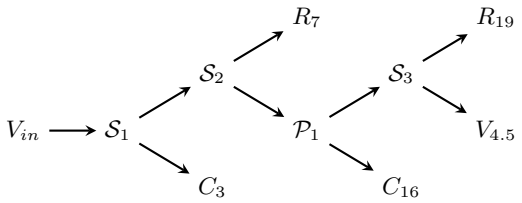

Figure 4: *WDF tree for the Klon Centaur Feed-Forward Network 1 Circuit. $\mathcal{S}$ and $\mathcal{P}$ nodes refer to series and parallel adaptors respectively.*

impedances for each principle circuit component: resistors (Z_R), capacitors (Z_C), and inductors (Z_L), are as follows:

$$Z_R = R, \quad Z_C = \frac{1}{Cs}, \quad Z_L = Ls \tag{1}$$

From there, using linear circuit theory, one can construct a Laplace Domain transfer function for the circuit. Note that this assumes an ideal operational amplifier operating in its linear region. For more information on this process, see [13]. For the tone control circuit, the Laplace domain transfer function can be written as:

$$\frac{V_{out}(s)}{V_{in}(s)} = \frac{C_{14}\left(\frac{1}{R_{22}} + \frac{1}{R_{21}+R_{v2b}}\right)s + \frac{1}{R_{22}}\left(\frac{1}{R_{21}+R_{v2b}} + \frac{1}{R_{23}+R_{v2a}}\right)}{C_{14}\left(\frac{1}{R_{23}+R_{v2a}} + \frac{1}{R_{24}}\right)s + \frac{-1}{R_{24}}\left(\frac{1}{R_{21}+R_{v2b}} + \frac{1}{R_{23}+R_{v2a}}\right)} \tag{2}$$

Note that we refer to the the section of potentiometer R_{v2} that is above the wiper as R_{v2a}, and the section below as R_{v2b}, and that we ignore the DC offset created by the 4.5V voltage source at the positive terminal of the op-amp.

Next we use a conformal map to transform the transfer function from the Laplace domain to the z-plane where it can be implemented as a digital filter. The most commonly used conformal map is the bilinear transform, defined as

$$s \leftarrow \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}} \tag{3}$$

Where T is the sample period of the digital system. For more information on the use of the bilinear transform to digitize an analog system, see [14]. The resulting filter is known as a "high-shelf" filter, that accentuates high frequency content in the signal. The resulting frequency response of the digital model, validated against the response of the analog circuit is shown in fig. 2.

2.1.1. Advantages and Limitations

The advantages of nodal analysis are that the circuit model is simple and computationally efficient. The model can be constructed with minimal knowledge of circuit theory, and a basic understanding of digitial signal processing. The main disdvantage is that nodal analysis cannot be used for nonlinear circuits, though it can be extended to model this class of circuits through modified nodal analysis (MNA) [15]. Another disadvantage of nodal analysis-based methods is that (typically) large portions of the system need to be recomputed when a circuit element is changed, such as a potentiometer. While this computation is fairly simple in the example shown here, it can become vastly more difficult for more complex systems.

2.2. Wave Digital Filters

The Wave Digital Filter (WDF) formalism allows circuits to be modelled in modular and flexible manner. Originally developed by Alfred Fettweis in the 1970's [6], WDFs have recently gained popularity in modelling audio circuits, and have been extended to model a wider class of circuits [7]. The WDF formalism defines each circuit element as a port with some characteristic resistance R_0, and uses wave variables passing through each port, rather than the typical voltage and current variables. The incident wave at a certain port is defined as:

$$a = v + R_0 i \tag{4}$$

where v is the voltage across the port, and i is the current passing through the port. The reflected wave is similarly defined as:

$$b = v - R_0 i \tag{5}$$

A Wave Digital Filter defines circuit elements (resistors, capacitors, inductors, etc.) in the wave domain, and allows the elements to be connected by series and parallel adaptors also defined in the wave domain. The full derivation of these WDF elements is given in [6] and [7].

Once each circuit element and adaptor has been defined, they are connected together in a structure often referred to as a WDF tree. As an example, we examine the "feed-forward network 1" from

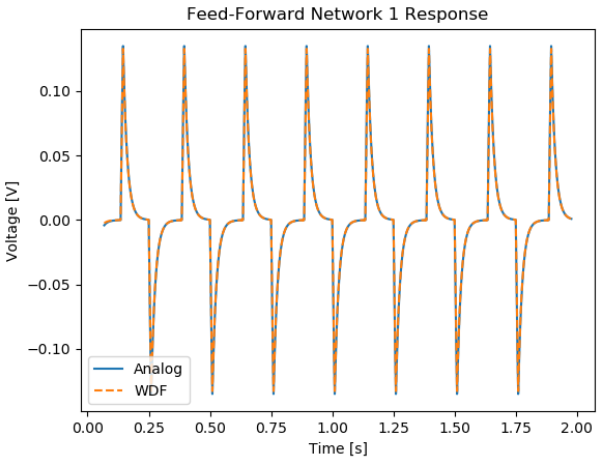

Figure 5: *WDF simulation results compared to the analog reference.*

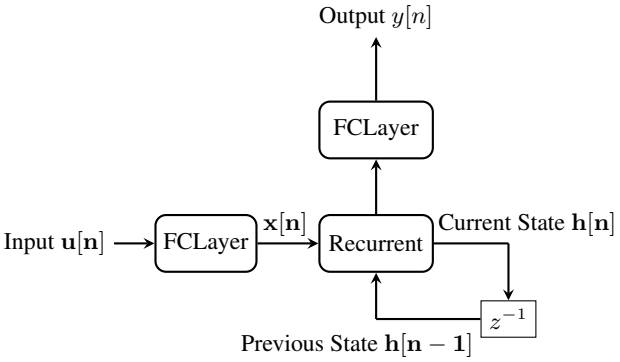

Figure 6: *RNN Architecture used for modelling the Klon Centaur gain stage circuit. Nodes labelled "FCLayer" denote fully connected neural network layers. The input vector* $\mathbf{u}[\mathbf{n}]$ *consists of a single input audio sample, as well as parameter values and sample rate information.*

the Klon Centaur circuit (see fig. 3). The corresponding WDF tree is shown in fig. 4. Simulation results compared to the analog reference are shown in fig. 5.

2.2.1. Advantages and Limitations

The primary advantage of the Wave Digital approach is its modularity. The ability to construct a circuit model with each circuit component treated completely independently in the digital domain opens up many interesting possibilities for circuit prototyping, modelling circuit-bent instruments and more. Additionally, this modularity allows each circuit component to be discretized separately, even using different conformal maps, which can improve model behavior for certain classes of circuits (see [3]). Finally, the separability of components means than when a component is changed (e.g. a potentiometer), the component change is propagated so that only components with behavior that depends on the impedance of the changed component need to be recomputed.

The main disdvantage of WDFs is their difficulty in handling circuits with complex topologies or multiple nonlinearities. While the recent addition of $\mathcal{R}$-type adaptors to the Wave Digital formalism [7] has begun to make these circuits tractable, the WDF models of these types of circuits are significantly more computationally complex. Further, the use of $\mathcal{R}$-type adaptors can somewhat compromises the modularity that makes WDFs advantageous in the first place.

3. RECURRENT NEURAL NETWORK MODEL

While several styles of machine-learning based models have been used for modelling analog audio circuitry [10, 11, 16], we choose the recurrent neural network approach developed in [12] as our starting point. Using a recurrent neural network (RNN) allows the for a significantly smaller neural network than would be possible with a traditional deep neural network or convolutional neural network, meaning that the network can be evaluated much faster for real-time use, while maintaining a smaller memory footprint (an important advantage on embedded platforms). Additionally, recurrent neural networks are a sensible candidate for modelling distortion circuits, particularly circuits with stateful behavior, given the fact that recurrent network building blocks, such as gated recurrent units, themselves resemble audio distortion effects and can directly be used as such [17]. In the following paragraphs, we outline the use of an RNN for modelling the gain stage circuit from the Klon Centaur pedal.

3.1. Model Architecture

The model architecture described in [12] consists of a single recurrent layer followed by a fully connected layer consisting of a single "neuron". In our model, we add a time-distributed fully connected layer before the recurrent layer. The full model architecture is shown in fig. 6. For training, all models are implemented in `Python` using the `Keras` framework [18].

In the architecture described in [12], an input vector for the networks is constructed to contain a single input audio sample, along with parameter values for any parameters present in the circuit. In our model we use a vector of 3 values as the network input: a single audio sample, the value of the circuit "Gain" parameter, and the sample period of the input audio. In previous neural network circuit modelling literature, the neural network must always be used to process audio at the same sample rate at which it was trained. For practical audio effects this limitation is problematic, since users often wish to use their effects at arbitrary sample rates, and resampling the audio by a non-integer ratio in real-time can be computationally expensive. By including the sample period as an input to the network, and by training the network on audio at different sample rates, we can effectively train the network to give accurate output for audio input at any sample rate within the range of sample rates used for training.

3.1.1. Recurrent Layer

Recurrent layers are typically comprised of one of two types of recurrent units: Long Short-Term Memory units (LSTMs) or Gated Recurrent Units (GRUs). For this application, we choose to use

GRUs [19] since they require fewer operations, allowing for faster computation, and since they requre fewer weights, thereby allowing the model to have a smaller memory footprint. The GRU consists of three "gates": the update gate $z[n]$, reset gate $r[n]$, and the new gate $c[n]$. These gates are used to compute the cell's current output $h[n]$ from its current input $x[n]$ and previous output $h[n-1]$ as follows:

$$z[n] = \sigma(W_z x[n] + U_z h[n-1] + b_z) \tag{6}$$

$$r[n] = \sigma(W_r x[n] + U_r h[n-1] + b_r) \tag{7}$$

$$c[n] = \tanh(W_c x[n] + r[n] \circ U_c h[n-1] + b_c) \tag{8}$$

$$h[n] = z[n] \circ h[n-1] + (1 - z[n]) \circ c[n] \tag{9}$$

Where W_z, W_r, W_c are the kernel weights for each gate, U_z, U_r, U_c are the recurrent weights for each gate, and b_z, b_r, b_c are the biases for each gate. Note that as the inputs and outputs to the GRU layer may be vectors, all products in the above equations are assumed to be standard matrix-vector products, except those Hadamard products denoted $\circ$. $\sigma(x)$ refers to the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$.

The most important "hyperparameter" of the network architecture used here is the choice of how many recurrent units to use in the recurrent layer. This choice will determine the accuracy of the network as well as the computational complexity of evaluating the output of the network in real-time. In this work, we use two models: a "small" model using a dense layer of size 4, followed by a recurrent layer with 8 GRUs, and a "large" model using a dense layer of size 8 followed by a recurrent layer with 16 GRUs.

3.1.2. *Fully Connected Layer*

A fully connected layer computes an output vector $y[n]$ from input vector $x[n]$ as follows:

$$y[n] = \alpha(W x[n] + b) \tag{10}$$

Where W is the kernel weights, b is the layer bias, and $\alpha(x)$ is the layer activation. In our model, the first fully connected layer uses the activation function $\alpha(x) = \tanh(x)$, and the final fully connected layer uses no activation, i.e., $\alpha(x) = x$.

3.2. Training Data

Our dataset consists of $\sim$ 4 minutes of electric guitar recordings, from a variety of electric guitars including a Fender Stratocaster and a Gibson Les Paul. The guitars are recorded "direct" meaning that the recorded signal is equivalent to the signal received by the pedal coming directly from the guitar. Recordings were made using a Focusrite Scarlett audio interface at 96 kHz, and then resampled to a random sample rate on the range [24 kHz, 96 kHz], using the `Librosa` library [20]. The recordings were then separated into segments of 0.5 seconds each.

Since the original Klon Centaur pedal is quite expensive ($>$ 1500 USD), we used a SPICE simulation of the Centaur circuit in order to obtain a "ground truth" reference dataset. The reference dataset measures the output voltage of the summing amplifier from the circuit at five different values for the "Gain" potentiometer.

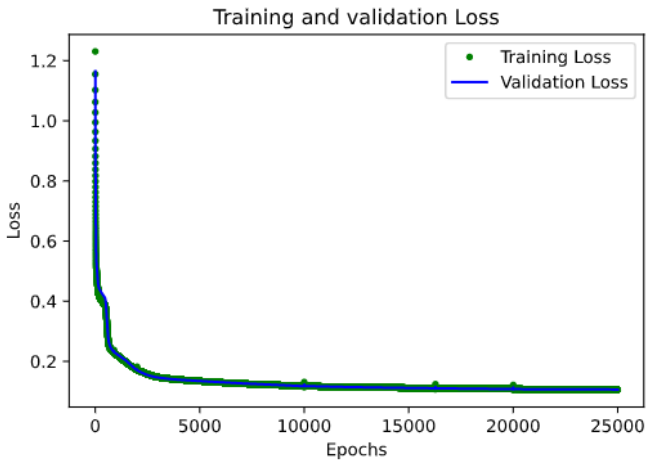

Figure 7: *Training accuracy for the "small" RNN, shown over epochs.*

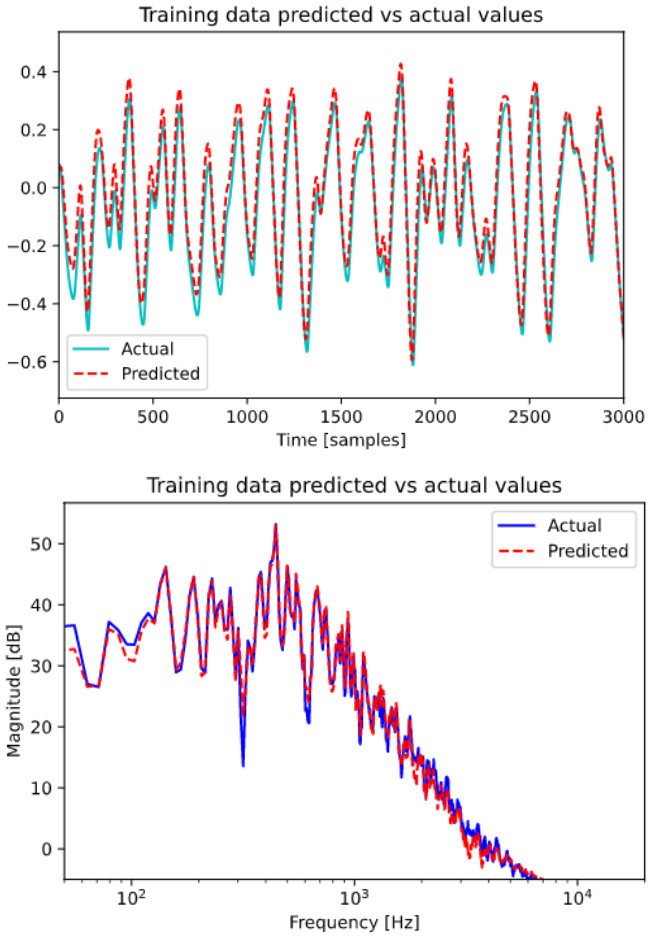

Figure 8: *Comparison of predicted output of the "large model" against reference output shown in the time domain (above) and frequency domain (below). The frequency domain plot uses frequency band smoothing using $1/24$ octave bands for improved clarity.*

3.3. Training

We trained our models on 950 audio samples, using 50 samples for validation. Training was performed using the Adam optimizer [21], with an initial learning rate of 5×10^{-4}. Each model was trained for more than 20,000 epochs, and training was stopped manually as the models began to show signs of overfitting to the training data. For training, we used the same loss functions as in [12]: a weighted error-to-signal ratio summed with a DC loss term. For evaluating the model accuracy, we use an un-weighted error-to-signal ratio (ESR) defined as,

$$\mathcal{E}_{ESR} = \frac{\sum_{n=0}^{N-1} |y[n] - \hat{y}[n]|^2}{\sum_{n=0}^{N-1} |y[n]|^2} \tag{11}$$

where $y[n]$ is the reference output, and $\hat{y}[n]$ is the predicted output of the network.

3.3.1. Training Results

For each model, the trained network achieved a validation ESR of less than 10%, a result comparable to that found in [12]. The results of training for both models are shown in table 1. The training accuracy over epochs for the small model is shown in fig. 7. Results comparing the output of the large model to the reference output are shown in fig. 8. Note that the high frequency response of the RNN output is slightly damped compared to the reference.

Model	**Small**	**Large**
# Dense	4	8
# GRU	8	16
# Epochs	25000	21000
Training ESR	8.08	6.62
Validation ESR	9.18	7.79

Table 1: *Hyper-parameters and accuracy results for each model. Hyper-parameters include the sizes of the GRU layer and the first dense layer, as well as the number of epochs over which the model was trained. Training and validation accuracies are given as error-to-signal ratio percentages.*

3.4. Advantages and Limitations

The recurrent neural network is a flexible and powerful black-box modelling tool for stateful nonlinear systems. The main limitation of the RNN model is its computational complexity for large models, mostly due to the fact that the $\tanh$ and sigmoid functions required by the recurrent layer can be costly to compute. However, the model presented in [12] gives a distinct advantage over traditional black-box modelling techniques in that it allows control parameters to be included in the model. As shown above, the parameter inclusion method can be extended to allow the neural network to be used for audio at a range of sample rates, a further improvement on other neural network-based techniques.

4. IMPLEMENTATION

In order to compare the virtual analog methods described above, we construct two emulations of the Klon Centaur circuit: one emulation using traditional circuit modelling methods (non-ML implementation), and a second using a recurrent neural network (ML implementation). The Centaur circuit can be broken down into four separable parts (see fig. 9):

1. Input Buffer
2. Gain Stage
3. Tone Control
4. Output Buffer

Due to their relative simplicity and linearity, in both emulations the input buffer, output buffer, and tone control circuits were modelled using nodal analysis. The "Gain Stage" circuit can be further broken down into six (mostly) separable parts (see fig. 10):

1. Feed-Forward Network 1 (FF-1)
2. Feed-Forward Network 2 (FF-2)
3. Pre-Amp Stage
4. Amplifier Stage
5. Clipping Stage
6. Summing Amplifier

In the ML implementation, we treat the Gain Stage as a black box with a single user-facing control (the "Gain" control). The RNN model is designed to completely replace the Gain Stage in the circuit model. In the non-ML implementation, we use nodal analysis to model the amplifier stage, and summing amplifier circuits. For FF-2 and the clipping stage, we use a wave digital filter. Since FF-1 and the pre-amp circuit share a capacitor, we construct a joint WDF model of these two circuits, using the voltage output from the pre-amp circuit as the input to the amplifier stage, and the current output from FF-1 (summed with the current outputs of FF-2 and the clipping stage) as the input to the summing amplifier.

4.1. Audio Plugin

Digital audio effects are often implemented as audio plugins that can be used by mixing engineers, producers, and musicians in a consumer digital audio workstation (DAW) software. Common plugin formats include the Avid Audio Extension (AAX), Steinberg's Virtual Studio Technology (VST), and Apple's Audio Unit (AU) for desktop use, as well as Apple's Audio Unit v3 (AUv3) for mobile use. The JUCE C++ framework[1] is commonly used to create cross-platform, cross-format plugins.

As a demonstration of the two circuit emulations, we construct an audio plugin containing both models, allowing the user to switch between the two models for comparison. The plugin is implemented using JUCE/C++, along with a real-time Wave Digital Filter library[2] for the WDF models. For computing the output of the RNN models, we have implemented a custom inferencing engine in C++, with two modes, one using the Eigen linear algebra library [22], the second using only the C++ standard library. In the future, we plan to add a third mode, that uses the Tensorflow Lite library.[3]

[1] `https://github.com/juce-framework/JUCE`

[2] `https://github.com/jatinchowdhury18/WaveDigitalFilters`

[3] `https://www.tensorflow.org/lite/`

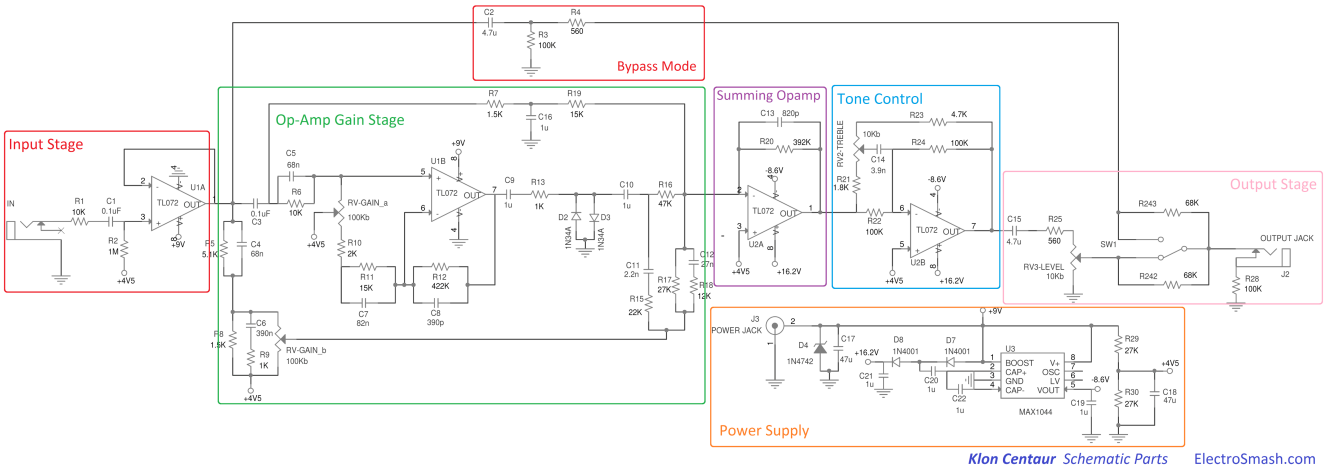

Figure 9: *Full circuit schematic for the Klon Centaur guitar pedal with different circuit sections outlined. Adapted from [2].*

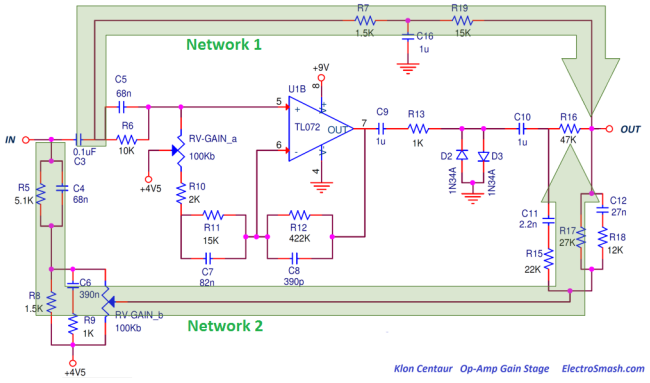

Figure 10: *Circuit schematic for the gain stage from the Klon Centaur pedal, with the two Feed-Forward networks highlighted. Adapted from [2].*

Figure 11: *Audio plugin implementation of the Klon Centaur circuit model. Note controls for "Gain", "Treble", and "Level" analogous to the original circuit, as well as the "neural" parameter to control whether the emulation uses the traiditional circuit model, or the RNN model.*

4.2. Embedded Implementation

Digital audio effects are sometimes implemented on embedded devices for use in stage performances, often in the form of a guitar pedal, or synthesizer module. Deploying an audio effect on an embedded device can be difficult, due to the constraints in processing power and memory availability. Further, in order to achieve a more expressive performance, musicians often prefer effects that add minimal latency to the signal, meaning that the embedded implementation must be able to run with a very small buffer size.

We chose the Teensy 4.0 microcontroller as our embedded platform, since it contains a reasonbly powerful floating point processor at a relatively low price point. The Teensy can be purchased along with an Audio Shield, which provides 16-bit stereo audio input/output at 44.1 kHz sampling rate. The Teensy has gained popularity in the audio community due to the Teensy Audio Library[4] that contains useful audio DSP functionality, as well as the Faust programming language which allows audio effects and synthesizers made in Faust to be exported for use on the Teensy [23]. The Teensy 4.0 with the audio shield can be purchased for 35 USD.

The Teensy implementation is writteen in C++ using the Teensy Audio Library, along with the same WDF library as the audio plugin, as well as the standard library mode of the same RNN inferencing engine. The emulation can be compiled to use either the ML or non-ML implementation. Variables in the code can be connected to potentiometers or push-buttons to control model parameters in real-time.

5. RESULTS

The results of the real-time implementations described above can best be seen through audio performance examples. To that end, we provide video examples of both implementations being used in real-time on a guitar input being performed live. These examples can be seen on YouTube.[5] From subjective listening, the ML and

[4] https://www.pjrc.com/teensy/td_libs_Audio.html

[5] https://www.youtube.com/playlist?list=PLrcXtWXbPsj11cNBamVyMmDcWY1SXZHvz

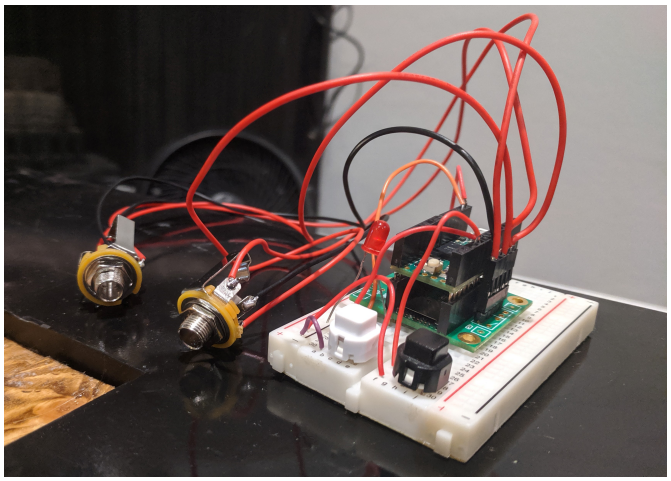

Figure 12: *Teensy microcontroller implementation.*

non-ML implementations sound very similar, although the ML implementation has slightly damped high frequencies, as predicted by the results of model training (see fig. 8). The high frequency damping is slightly more noticeable when the audio input is something other than a guitar, e.g. drums. These issues could likely be alleviated by training on a more diverse set of audio, and possibly by adjusting the loss function to weight high frequencies more.

5.1. Performance

We also evaluate the computational performance of the emulations. For real-time performance it is important to have fast computational performance in order to reduce audio latency. In table 2, we show the speeds of the various models at different input block sizes. Note that at all block sizes, the ML implementation outperforms the non-ML implementation.

Block Size	NonML Speed	ML Speed
8	0.0723437	0.0528792
16	0.0703079	0.0510437
32	0.0652856	0.0511147
64	0.0662835	0.0502434
128	0.0666593	0.0495194
256	0.0696844	0.0480298
512	0.0669037	0.0477946
1024	0.060816	0.0488841
2048	0.0695175	0.0488309
4096	0.0623839	0.0472191

Table 2: *Benchmark results comparing processing speed of the audio plugin implementation using ML processing vs. non-ML processing. Speed is measured in compute time per second of audio processed.*

5.2. Recommendations

From the process of implementing the circuit emulations described above, we provide the following recommendations for circuit modellers:

- For simple, linear circuits, nodal analysis is the easiest and best performing circuit modelling method.
- When modularity is important, prefer Wave Digital Filters. This modularity can refer to the circuit topology, the components in the circuit, or the way in which the components are discretized.
- For complex nonlinear systems, particularly systems with multiple nonlinear elements, or stateful nonlinear topologies, consider using recurrent neural networks.
- Small RNNs can outperform more complex circuit modelling methods while still maintaining model accuracy.

6. CONCLUSION

In this writing we have constructed two emulations of the Klon Centaur guitar pedal circuit, using circuit modelling techniques including nodal analysis, wave digital filters, and recurrent neural networks. We describe and compare the advantages and limitations of each method, and show how they can be used together to achieve good results. We implement the circuit emulations in the form of an audio plugin and guitar-pedal style effect embedded on a Teensy microcontroller. Finally, we provide recommendations for utilising different circuit modelling methods for different types of circuits, and for different platforms. The code for both implementations, as well as the model training, is open source and can be found on GitHub.[6]

In future works, we would like to extend the RNN framework to be able to implement larger networks in real-time. Specifically, the Differentiable Digital Signal Processingg (DDSP) library from Google's Magenta project [24] implements complex audio effects including timbral transfer, dereverberation, and more using an auto-encoder that contains two 512-unit GRUs, along with several other complex operations. Being able to implement the DDSP auto-encoder for use on real-time signals would be a powerful tool for musicians and audio engineers.

7. ACKNOWLEDGMENTS

The author would like to thank Pete Warden and the EE292 class at Stanford University for inspiring this project, as well as Julius Smith, Kurt Werner, and Jingjie Zhang for assistance with Wave Digital Filter modelling. Thanks as well to the Center for Computer Research in Music and Acoustics (CCRMA) for providing computing resources.

8. REFERENCES

[1] West Warren, "Builder Profile: Klon's Bill Finnegan," *Premier Guitar*.

[2] "Klon centaur analysis," *ElectroSmash*.

[3] Francois Germain, *Non-oversampled physical modeling for virtual analog simulations*, Ph.D. thesis, Stanford University, June 2019.

[6] https://github.com/jatinchowdhury18/KlonCentaur

[4] D.T. Yeh, *Digital Implementation of Musical Distortion Circuits by Analysis and Simulation*, Ph.D. thesis, Stanford Univeristy, June 2009.

[5] Antoine Falaize and Thomas Helie, "Passive guaranteed simulation of analog audio circuits: A Port-Hamiltonian approach," *Applied Sciences*, vol. 6(10), pp. 273, Sept. 2016.

[6] A. Fettweis, "Wave digital filters: Theory and practice," *Proceedings of the IEEE*, vol. 74, no. 2, pp. 270–327, Feb. 1986.

[7] Kurt James Werner, *Virtual Analog Modeling of Audio Circuitry Using Wave Digital Filters*, Ph.D. thesis, Stanford Univeristy, June 2016.

[8] Martin Holters and Udo Zolzer, "A generalized method for the derivation of non-linear state-space models from circuit schematics," in *Proc. of the 23rd European Signal Processing Conference (EUSIPCO)*, Sept. 2015, pp. 1078–1082.

[9] Julius O. Smith, *Spectral Audio Signal Processing*, `http://ccrma.stanford.edu/~jos/sasp/`, accessed 2020-5-22, online book, 2011 edition.

[10] Eero-Pekka Damskagg, Lauri Juvela, and Vesa Valimaki, "Real-time modeling of audio distortion circuits with deep learning," in *Proc. of the 16th Sound and Music Computing Conference*, May 2019.

[11] Julian D. Parker, Fabian Esqueda, and Andre Bergner, "Modelling of nonlinear state-space systems using a deep neural network," in *Proc. of the 22nd Int. Conference on Digital Audio Effects (DAFx-19)*, Sept. 2019.

[12] Alec Wright, Eero-Pekka Damskagg, and Vesa Valimaki, "Real-time black-box modelling with recurrent neural networks," in *Proc. of the 22nd Int. Conference on Digital Audio Effects (DAFx-19)*, Sept. 2019.

[13] Edward W. Maby, *Solid State Electronic Circuits*, vol. 4, 1 edition, 2014.

[14] Julius O. Smith, *Physical Audio Signal Processing*, `http://ccrma.stanford.edu/~jos/pasp/`, accessed 2020-5-22, online book, 2010 edition.

[15] Chung-Wen Ho, A. Ruehli, and P. Brennan, "The modified nodal approach to network analysis," *IEEE Transactions on Circuits and Systems*, vol. 22, no. 6, pp. 504–509, 1975.

[16] Marco A. Martínez Ramirez and Joshua D. Reiss, "Modeling of nonlinear audio effects with end-to-end deep neural networks," *arXiv e-prints*, p. arXiv:1810.06603, Oct. 2018.

[17] Jatin Chowdhury, "Complex nonlinearities for audio signal processing," `https://ccrma.stanford.edu/~jatin/papers/Complex_NLs.pdf`, Feb. 2020.

[18] François Chollet et al., "Keras," `https://github.com/fchollet/keras`, 2015.

[19] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014.

[20] Brian McFee et al., "librosa/librosa: 0.8.0," July 2020.

[21] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[22] Gaël Guennebaud, Benoît Jacob, et al., "Eigen v3," http://eigen.tuxfamily.org, 2010.

[23] Romain Michon, Yann Orlarey, Stéphane Letz, and Dominique Fober, "Real time audio digital signal processing with Faust and the Teensy," in *Proc. of the 16th Sound and Music Computing Conference*, May 2019.

[24] Jesse Engel, Lamtharn (Hanoi) Hantrakul, Chenjie Gu, and Adam Roberts, "DDSP: Differentiable digital signal processing," in *International Conference on Learning Representations*, 2020.