Neural grey-box guitar amplifier modelling with limited data

Citation for published version:
Wright, A, Valimaki, V, Miklanek, S & Schimmel, J 2023, Neural grey-box guitar amplifier modelling with limited data. in *Proceedings of the 26th International Conference on Digital Audio Effects (DAFx23).* Proceedings of the International Conference on Digital Audio Effects, Aalborg University, Copenhagen, Denmark, pp. 151-158. <https://www.dafx.de/paper-archive/2023/DAFx23_paper_52.pdf>

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Publisher's PDF, also known as Version of record

Published In:
Proceedings of the 26th International Conference on Digital Audio Effects (DAFx23)

NEURAL GREY-BOX GUITAR AMPLIFIER MODELLING WITH LIMITED DATA

Štěpán Miklánek[1,2,*], *Alec Wright*[1], *Vesa Välimäki*[1,†] *and Jiří Schimmel*[2]

[1]Acoustics Lab, Department of Information and Communications Engineering, Aalto University, Espoo, Finland
[2]Department of Telecommunications, Brno University of Technology, Brno, Czech Republic
stepan.miklanek@vut.cz

ABSTRACT

This paper combines recurrent neural networks (RNNs) with the discretised Kirchhoff nodal analysis (DK-method) to create a grey-box guitar amplifier model. Both the objective and subjective results suggest that the proposed model is able to outperform a baseline black-box RNN model in the task of modelling a guitar amplifier, including realistically recreating the behaviour of the amplifier equaliser circuit, whilst requiring significantly less training data. Furthermore, we adapt the linear part of the DK-method in a deep learning scenario to derive multiple state-space filters simultaneously. We frequency sample the filter transfer functions in parallel and perform frequency domain filtering to considerably reduce the required training times compared to recursive state-space filtering. This study shows that it is a powerful idea to separately model the linear and nonlinear parts of a guitar amplifier using supervised learning.

1. INTRODUCTION

Virtual analogue (VA) modelling [1, 2, 3] is a broad topic that still offers room to combine different approaches to obtain faithful digital models of real devices. Approaches to VA modelling are often divided into “black-box” methods, which require almost no knowledge of the target device’s inner workings [4]. These methods are limited because they typically do not allow for simulating user controls, which are present in most analogue devices.

On the other hand, there are “white-box” VA modelling techniques, which create discretised versions of the actual electrical circuits and allow for simulation of the behaviour of variable components such as potentiometers [5, 6, 7]. Typically, these methods are based on wave digital filters [2, 8, 9] or nonlinear state-space representations [10, 11, 12]. Nonlinear white-box models often require the utilisation of approximation or look-up tables in order to run efficiently in real time [10]. The overall modelling accuracy is also affected by the exactness of physical models of nonlinear components, namely diodes, transistors or vacuum tubes, which often require fitting to measurements from real components [13]. Furthermore, component values listed in schematics may need to be optimised using data recorded from the device to account for inaccurate schematics and component tolerances [14].

With the emergence of the concept of differentiable digital signal processing [15], recent works have shown that it is possible to adapt white-box modelling methods in a deep learning scenario to either discover unknown values of components in analogue circuit models [16] or to replace some of the computationally demanding nonlinear circuit elements with small neural networks [17, 18].

Finally, we can define the “grey-box” approach to VA modelling. The methods that fall into this category require some knowledge of the inner structure of given devices but also rely on measurements [19], such as the state trajectory network method [20]. Various grey-box techniques were previously successfully applied to create models of guitar amplifiers [21], time-varying effects [22] or dynamic range compressors [23].

Simulation of vacuum tube guitar amplifiers is among the popular VA modelling subfields. Numerous works utilising deep learning methods for black-box guitar amplifier modelling have been published in the last few years [24, 25, 26, 27, 28]. One way to adapt these models to emulate user controls is to capture audio datasets of various control settings and allow the model to learn the dependencies from data using conditioning [29]. However, simulating multiple controls with conditioning calls for a reliable automated method for sampling numerous control combinations, making this task unsuitable for manual data collection [30].

In this paper, we propose a grey-box model that combines recurrent neural networks (RNNs) for the nonlinear preamplifier and power amplifier simulation, with a white-box linear state-space model of a guitar amplifier equaliser section, commonly referred to as the tone stack [31]. The main contributions of this paper are as follows. We show that our model needs only a fraction of the data to learn the tone stack behaviour accurately, when compared to an RNN baseline model, whilst also generalising far better to unseen data. Although the differentiable tone stack model was already presented in [16], we utilise the frequency sampling of the tone stack state-space model for frequency domain filtering, which results in considerably faster training times when compared to recursive time domain filtering. In addition, we adapt the discretised Kirchhoff nodal analysis (DK-method) to efficiently derive multiple state-space filters in a deep learning framework.

The rest of the paper is structured as follows. Sec. 2 describes the modelled guitar amplifier and the data acquisition process. In Sec. 3, we describe the differentiable tone stack model, the frequency sampling of the tone stack filter for deep learning purposes, and a method used for frequency domain filtering. Sec. 4 describes the proposed neural network model and the hyperparameters used for training. Sec. 5 presents our experiments. Sec. 6 summarises the objective and subjective results. Finally, Sec. 7 concludes.

2. MODELLED DEVICE

The device in question is a Marshall JVM 410H vacuum tube amplifier. For the channel setting modelled in the work, the circuit topology consists of a preamplifier followed by a tone stack and a

[*] Work carried out during a research visit to the Aalto Acoustics Lab in Feb.–Mar. 2023.

[†] This research is part of the activities of the Nordic Sound and Music Computing Network—Nordic SMC (NordForsk project number 86892).

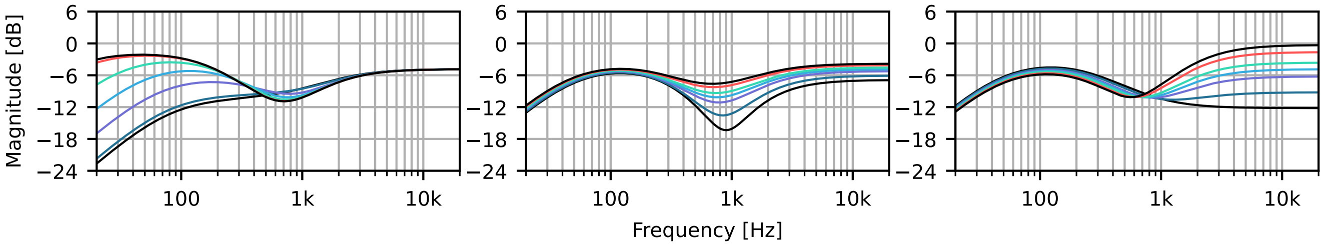

Figure 1: *Frequency responses of the JVM tone stack model with each control (bass, middle, and treble) varied from 0 to 10.*

Table 1: *Tone stack settings for evaluation on parameter values not seen in training. The values are normalised to the range* $[0, 1]$ *and denoted by* c_b *for the bass control,* c_m *for the middle control, and* c_t *for the treble control.*

c_b	c_m	c_t	c_b	c_m	c_t	c_b	c_m	c_t
0.1	0.3	0.7	0	0	1	1	0	0
0.3	0.7	0.1	0	1	0	1	0	1
0.65	0.85	0.35	0	1	1	1	1	0

power amplifier. The preamplifier is controlled by a “gain” knob, which adjusts the signal level before the tube clipping stages. The tone stack has a well known topology used in Fender, Vox, and Marshall amplifiers (FMV), described in Sec. 3.1. Its frequency response can be altered by three potentiometers, labelled as “bass”, “middle”, and “treble”. Fig. 1 shows frequency responses of the JVM tone stack model as each control is varied. After the tone stack, the signal level is adjusted by a “volume” knob followed by the power amplifier, which has three controls. A “master” knob adjusts the signal level before the power tubes, and “resonance” and “presence” knobs alter the low- and high-frequency content.

2.1. Dataset Description

It has previously been shown that the amount of training data required for black-box neural guitar amplifier modelling is relatively low [28]. Thus, we composed a 6-min-long dataset of guitar and bass audio files of different playing styles and genres. We split the dataset into three parts: the first 4 min are used for training, and the remaining 2 min are split in half for validation and testing. All the sounds were taken from IDMT datasets described in [32, 33].

For the sake of simplicity, we model a single channel of the amplifier, which is labelled as “OD1” and produces a relatively high amount of distortion. We further limit ourselves to only making the tone stack knobs fully controllable. Therefore, we set the channel volume to maximum (10) and the gain, master, resonance and presence to midpoint (5) in all cases. Then, the entire dataset was processed through the amplifier multiple times with different settings of the bass, middle, and treble controls. We varied each knob from minimum (0) to 10 with a step of 2. Each time a single control was varied, the remaining two were set to 5. This resulted in 6 different output signals for each control. In addition, we also recorded 3 output signals, where all knobs were set to 0, 5, and 10.

We captured 9 additional output signals whilst processing only the 1-min test subset to further evaluate the model performance on unseen parameter settings. These tone stack settings, normalised to the range of [0, 1], are shown in Table 1. For the first three output signals, the tone stack was set to values between the steps used for capturing the training subset. In the remaining cases, the tone stack was set to various combinations of extreme values to assess how well the models were able to generalise the non-orthogonal control behaviour described in [31].

The recording was carried out in the same fashion as in [29]. The output signals were recorded at a sampling rate of 44.1 kHz from the speaker output of the amplifier using a Two Notes Torpedo Captor 8 reactive load connected to a line input of an RME UCX USB audio interface.

3. DIFFERENTIABLE TONE STACK MODEL

We identified the DK-method [5] as a suitable analogue circuit discretisation technique to implement a differentiable version of the tone stack model. The DK-method allows an automated derivation of the state-space model from a list of virtual electronic components. We use the version of the DK-method proposed in [12]. In comparison with the earlier version [5], it allows more efficient handling of the variable components (potentiometers), which is useful not only for inference but also for adapting it for deep learning using PyTorch [34] as shown in Sec. 3.1. In this work, we use the DK-method only to derive a linear model. For more information about modelling nonlinear systems with the DK-method, we refer to other sources [10, 11, 35].

3.1. Derivation of the Tone Stack Model

The first step is to construct so-called incidence matrices that specify to which circuit nodes the individual components are connected. The number of rows in the incidence matrix is equal to the number of components, and the number of columns is equal to the number of circuit nodes (excluding the ground node). Entries in each row are given by positive and negative poles of the components, and are marked by $(+1)$ and (-1), respectively. The negative pole is omitted for components connected to the ground node. In the case of the tone stack model, we need to create five incidence matrices where $\mathbf{N}_\mathrm{R}$ is for resistors, $\mathbf{N}_\mathrm{V}$ for variable resistors, $\mathbf{N}_\mathrm{x}$ for capacitors, $\mathbf{N}_\mathrm{u}$ for the voltage source v_in, and $\mathbf{N}_\mathrm{o}$ for the output voltage v_out.

Next, we can build the system matrix $\mathbf{S}_0$, excluding the variable resistors, defined as

$$\mathbf{S}_0 = \begin{pmatrix} \mathbf{N}_\mathrm{R}^\mathsf{T}\mathbf{G}_\mathrm{R}\mathbf{N}_\mathrm{R} + \mathbf{N}_\mathrm{x}^\mathsf{T}\mathbf{G}_\mathrm{x}\mathbf{N}_\mathrm{x} & \mathbf{N}_\mathrm{u}^\mathsf{T} \\ \mathbf{N}_\mathrm{u} & \mathbf{0} \end{pmatrix}, \tag{1}$$

where $\mathbf{G}_\mathrm{R}$ and $\mathbf{G}_\mathrm{x}$ are diagonal matrices containing the parameter values of each resistor and capacitor respectively. Following the discretisation scheme from [12], the values in $\mathbf{G}_\mathrm{R}$ are conductances given by $\frac{1}{R_i}$, and the values in $\mathbf{G}_\mathrm{x}$ are computed by $\frac{2C_i}{T}$, where T is the sampling period.

To ensure the system matrix $\mathbf{S}_0$ is invertible, we augment the matrices $\mathbf{N}_\mathrm{R}$ and $\mathbf{G}_\mathrm{R}$ with virtual constant resistors [10], which

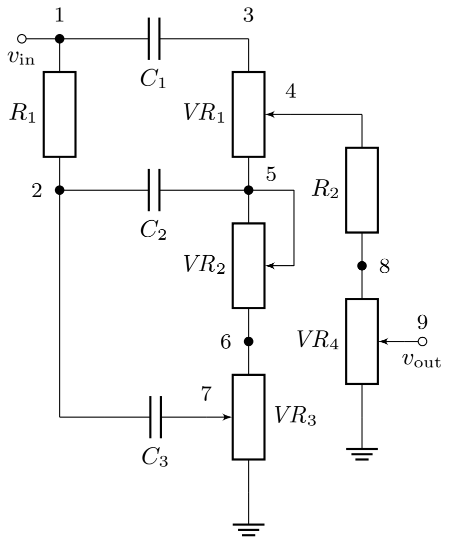

Figure 2: *Schematic of the JVM tone stack circuit, where the potentiometers VR_i are adjusted by control knobs.*

are connected in parallel to the resistors of the potentiometers VR_3 and VR_4. Consequently, we derive the coefficient matrices

$$\mathbf{A_0} = \begin{pmatrix}2\mathbf{G}_\mathrm{x}\mathbf{N}_\mathrm{x} & \mathbf{0}\end{pmatrix} \mathbf{S_0}^{-1} \begin{pmatrix}\mathbf{N}_\mathrm{x} & \mathbf{0}\end{pmatrix}^\mathsf{T} - \mathbf{I} \tag{2}$$

$$\mathbf{B_0} = \begin{pmatrix}2\mathbf{G}_\mathrm{x}\mathbf{N}_\mathrm{x} & \mathbf{0}\end{pmatrix} \mathbf{S_0}^{-1} \begin{pmatrix}\mathbf{0} & \mathbf{I}\end{pmatrix} \tag{3}$$

$$\mathbf{D_0} = \begin{pmatrix}\mathbf{N}_\mathrm{o} & \mathbf{0}\end{pmatrix} \mathbf{S_0}^{-1} \begin{pmatrix}\mathbf{N}_\mathrm{x} & \mathbf{0}\end{pmatrix}^\mathsf{T} \tag{4}$$

$$\mathbf{E_0} = \begin{pmatrix}\mathbf{N}_\mathrm{o} & \mathbf{0}\end{pmatrix} \mathbf{S_0}^{-1} \begin{pmatrix}\mathbf{0} & \mathbf{I}\end{pmatrix} \tag{5}$$

and the helper matrices

$$\mathbf{Q} = \begin{pmatrix}\mathbf{N}_\mathrm{V} & \mathbf{0}\end{pmatrix} \mathbf{S_0}^{-1} \begin{pmatrix}\mathbf{N}_\mathrm{V} & \mathbf{0}\end{pmatrix}^\mathsf{T} \tag{6}$$

$$\mathbf{U}_\mathrm{x} = \begin{pmatrix}\mathbf{N}_\mathrm{x} & \mathbf{0}\end{pmatrix} \mathbf{S_0}^{-1} \begin{pmatrix}\mathbf{N}_\mathrm{V} & \mathbf{0}\end{pmatrix}^\mathsf{T} \tag{7}$$

$$\mathbf{U}_\mathrm{o} = \begin{pmatrix}\mathbf{N}_\mathrm{o} & \mathbf{0}\end{pmatrix} \mathbf{S_0}^{-1} \begin{pmatrix}\mathbf{N}_\mathrm{V} & \mathbf{0}\end{pmatrix}^\mathsf{T} \tag{8}$$

$$\mathbf{U}_\mathrm{u} = \begin{pmatrix}\mathbf{0} & \mathbf{I}\end{pmatrix} \mathbf{S_0}^{-1} \begin{pmatrix}\mathbf{N}_\mathrm{V} & \mathbf{0}\end{pmatrix}^\mathsf{T} \tag{9}$$

where $\mathbf{0}$ and $\mathbf{I}$ are zero and identity matrices of appropriate dimensions. Finally, we compute the state-space matrices $\mathbf{A} \in \mathbb{R}^{3\times3}$, $\mathbf{B} \in \mathbb{R}^{3\times1}$, $\mathbf{D} \in \mathbb{R}^{1\times3}$, and $\mathbf{E} \in \mathbb{R}^{1\times1}$ as

$$\mathbf{A} = \mathbf{A_0} - 2\mathbf{G}_\mathrm{x}\mathbf{U}_\mathrm{x}(\mathbf{R_V} + \mathbf{Q})^{-1}\mathbf{U}_\mathrm{x}^\mathsf{T} \tag{10}$$

$$\mathbf{B} = \mathbf{B_0} - 2\mathbf{G}_\mathrm{x}\mathbf{U}_\mathrm{x}(\mathbf{R_V} + \mathbf{Q})^{-1}\mathbf{U}_\mathrm{u}^\mathsf{T} \tag{11}$$

$$\mathbf{D} = \mathbf{D_0} - \mathbf{U}_\mathrm{o}(\mathbf{R_V} + \mathbf{Q})^{-1}\mathbf{U}_\mathrm{x}^\mathsf{T} \tag{12}$$

$$\mathbf{E} = \mathbf{E_0} - \mathbf{U}_\mathrm{o}(\mathbf{R_V} + \mathbf{Q})^{-1}\mathbf{U}_\mathrm{u}^\mathsf{T} \tag{13}$$

where $\mathbf{R}_\mathrm{V} \in \mathbb{R}^{7\times7}$ is a diagonal matrix, which contains the resistances of the variable resistors given by $b\,VR_2$ for the bass control, $\frac{2(1-m)}{2-(1-m)}\,VR_3$ and $\frac{2m}{2-m}\,VR_3$ for the mid control, and $(1-t)\,VR_1$ and $t\,VR_1$ for the treble control. The coefficients $t \in [0,1]$, $m \in [0,1]$, and $b \in [0,1]$ denote the tone stack potentiometer settings. The resistances of the volume potentiometer VR_4 are derived in the same way as in the case of VR_3. However, they are left static and define a load for the tone stack circuit. The schematic of the modelled circuit with numbered nodes is shown in Fig. 2.

To adapt the model to be used in a deep learning framework, we can easily define trainable coefficients α_{R_i}, α_{VR_i} and α_{C_i} to optimise the component values from Table 2 during the model training similarly to [16]. We do this because component values taken directly from the schematic do not precisely correspond to the values of real components. Additionally, we use a scaled sigmoid function defined by $f(\alpha) = t_1 + \sigma(\alpha)t_2$, where t_1 and t_2 are fixed to limit the range in which the component values are adjusted. We set $t_1 = 0.8$ and $t_2 = 0.4$ to achieve ±20% tolerance.

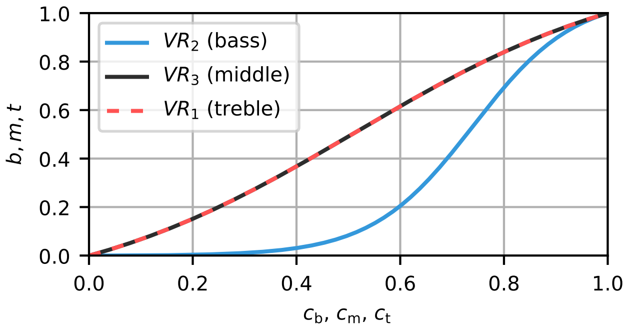

Figure 3: *Nonlinear potentiometer tapers.*

Table 2: *Component values of the JVM tone stack.*

Name	Value	Name	Value	Name	Value
R_1	33 kΩ	VR_2	1 MΩ	C_1	470 pF
R_2	39 kΩ	VR_3	20 kΩ	C_2	22 nF
VR_1	200 kΩ	VR_4	1 MΩ	C_3	22 nF

The resistances of the variable resistors in $\mathbf{R}_\mathrm{V}$ are also dependent on conditioning vectors $\mathbf{c}_\mathrm{b}$, $\mathbf{c}_\mathrm{m}$, and $\mathbf{c}_\mathrm{t}$, which describe how the controls were set on the amplifier. The number of elements in a single conditioning vector is given by the number of audio segments in the training batch of size s. The vectors $\mathbf{c}_\mathrm{b}$, $\mathbf{c}_\mathrm{m}$, and $\mathbf{c}_\mathrm{t}$ cannot be used to directly alter the variable resistances. Thus, we add nonlinear trainable tapers for potentiometers VR_1, VR_2, and VR_3, shown in Fig. 3. The same tapers were used in [16] to form a two layer neural network defined by $f(x) = w_1\tanh(w_2x+b_2)+b_1$, where $\tanh$ is a hyperbolic function, x is the layer input, and $w_{1\text{–}2}$ and $b_{1\text{–}2}$ are trainable weights and biases of the respective layers. We found that leaving w_1 and b_2 unrestricted does not assure that the mapping output will stay in the range of $[0,1]$, which is needed to compute the variable resistances correctly. To solve this, we compute

$$w_1 = 1/[\tanh(w_2 + b_1) - \tanh(b_1)], \tag{14}$$

$$b_2 = -w_1\tanh(b_1), \tag{15}$$

which results in only two free parameters w_2 and b_1 as described in [36]. The same work also specifies $w_{1\text{–}2}$ and $b_{1\text{–}2}$ parameter values for fitting either linear or logarithmic potentiometer tapers that we use to initialise the parameters of the mapping neural networks.

Since we recorded the audio signals from the amplifier with various settings of controls, each audio segment in a training batch has different conditioning values assigned to it due to dataset shuffling. As a result, the number of state-space representations we need to compute is also equal to s. To achieve this, we exploit the tensor broadcasting semantics of PyTorch[1] and adapt the DK-method to derive multiple state-space filters efficiently.

If we return to the equations (10)–(13), it can be seen that the state-space matrices will be different each time the variable resistances in $\mathbf{R}_\mathrm{V}$ change. Since each audio segment in the training

[1] https://pytorch.org/docs/stable/notes/broadcasting.html

batch needs to be filtered differently according to the conditioning vectors $\mathbf{c}_\mathrm{B}$, $\mathbf{c}_\mathrm{M}$, and $\mathbf{c}_\mathrm{T}$, we can create $\mathbf{R}_\mathrm{V} \in \mathbb{R}^{s\times 7\times 7}$, which is a tensor of variable resistances where the first dimension is equal to the batch size s. In other words, it is a tensor containing the variable resistances corresponding to the conditioning data for each item in the training batch, which are in turn used to obtain different state-space filters for each audio segment. If we use this tensor in computation of (10)–(13), we get tensors $\mathbf{A} \in \mathbb{R}^{s\times 3\times 3}$, $\mathbf{B} \in \mathbb{R}^{s\times 3\times 1}$, $\mathbf{D} \in \mathbb{R}^{s\times 1\times 3}$, and $\mathbf{E} \in \mathbb{R}^{s\times 1\times 1}$ as a result. This is relatively simple to implement using PyTorch, as the tensors will be correctly broadcasted as long as the remaining dimensions are compatible. Considering that we need to recompute the filters each time the neural network model parameters are updated, this approach is much less computationally expensive than deriving multiple state-space representations sequentially.

3.2. Frequency Sampling of the Tone Stack Filter

Several related works have already described frequency sampling of infinite impulse response (IIR) filters for deep learning purposes [37, 38, 39, 40]. Although IIR filters can be applied recursively in the time domain [41], the finite impulse response (FIR) approximation by frequency sampling allows for much faster training times [38]. Previous works investigated mainly frequency sampling of second-order sections (biquads). Nevertheless, we can do the same with state-space filters. Let us consider a single-input single-output (SISO) discrete system with scalar input $u[n]$ and output $y[n]$, that is defined by

$$\mathbf{x}[n+1] = \mathbf{A}\mathbf{x}[n] + \mathbf{B}u[n], \tag{16}$$

$$y[n] = \mathbf{D}\mathbf{x}[n] + \mathbf{E}u[n], \tag{17}$$

where $\mathbf{A}$ is the system matrix, $\mathbf{B}$ is the input matrix, $\mathbf{D}$ is the output matrix, $\mathbf{E}$ is the feedthrough matrix, and $\mathbf{x}$ is the state vector. To obtain the transfer function of a linear state-space filter, first, one has to take the $\mathcal{Z}$-transform of (16), which yields

$$z\mathbf{X}(z) - z\mathbf{x}[0] = \mathbf{A}\mathbf{X}(z) + \mathbf{B}U(z), \tag{18}$$

where $\mathbf{x}[0]$ represents the initial conditions. Then, the transformed state vector is given by

$$\mathbf{X}(z) = (z\mathbf{I} - \mathbf{A})^{-1}z\mathbf{x}[0] + (z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}U(z). \tag{19}$$

The transformed filter output (17) is equal to

$$\begin{aligned} Y(z) &= \mathbf{D}\mathbf{X}(z) + \mathbf{E}U(z) \\ &= \mathbf{D}(z\mathbf{I} - \mathbf{A})^{-1}z\mathbf{x}[0] + [\mathbf{D}(z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{E}]U(z). \end{aligned} \tag{20}$$

If we assume zero initial conditions ($\mathbf{x}[0] = 0$), (20) can be rearranged to obtain equation for the transfer function

$$H(z) = \frac{Y(z)}{U(z)} = \mathbf{D}(z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{E}, \tag{21}$$

which can be reformulated to

$$H(z) = \frac{\det(z\mathbf{I} - \mathbf{A} + \mathbf{B}\mathbf{D}) + \det(z\mathbf{I} - \mathbf{A})\mathbf{E}}{\det(z\mathbf{I} - \mathbf{A})}, \tag{22}$$

where det denotes the matrix determinant. Then, we derive a polynomial-form transfer function from (22) defined by

$$H(z) = \frac{b_0 z^m + b_1 z^{m-1} + \cdots + b_{m-1}z + b_m}{a_0 z^m + a_1 z^{m-1} + \cdots + a_{m-1}z + a_m}, \tag{23}$$

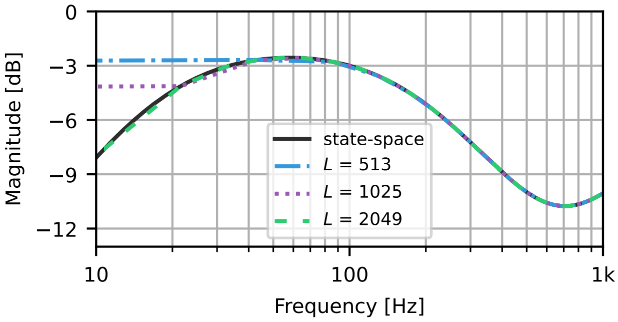

Figure 4: *Comparison of the ideal tone stack response with frequency sampled responses using a sampling vector of different lengths L, showing differences at low frequencies.*

where m is equal to the order of the system, and $b_{0\text{–}m}$ and $a_{0\text{–}m}$ are the numerator and denominator polynomial coefficients.

To frequency sample a linear state-space filter, first, we consider its frequency response denoted as $H(e^{j\omega})$, which can be frequency sampled at angular frequencies $\omega_k = 2\pi k/N$ where $k = 0, \ldots, \lfloor N/2 \rfloor$, and N is the length of the discrete Fourier transform (DFT). If we set z to $e^{j\omega_k}$, then we can easily derive $H(e^{j\omega_k})$ from the transfer function $H(z)$. Similarly as in [37], we can frequency sample $H(z)$ by combining frequency sampled m-sample delays $(z^{-m})_N \in \mathbb{C}^{\lfloor N/2 \rfloor + 1}$ and computing

$$H(e^{j\omega_k}) = H_N[k] = \frac{\sum_{m=0}^{M} b_m (z^{-m})_N[k]}{\sum_{m=0}^{M} a_m (z^{-m})_N[k]}. \tag{24}$$

A transfer function and frequency response of a linear state-space model can be computed using MATLAB or SciPy Python library functions called `ss2tf` and `freqz`, respectively. However, we need to compute several responses of state-space representations on a graphics processing unit (GPU) in parallel. To overcome this issue, we implemented differentiable methods for the PyTorch tone stack model to derive transfer functions and frequency responses for any number of SISO state-space filters simultaneously.

A few precautions must be taken when frequency sampling IIR filters. First of all, a sufficient sampling vector length L must be used to maintain the fidelity of the frequency response, especially at low frequencies. We found that $L = 2049$ is satisfactory for the tone stack model despite slight inaccuracies below 30 Hz as shown in Fig. 4. Increasing L above 2049 makes it possible to get even closer to the ideal response, but it is not necessary for model training purposes.

A less obvious problem arises from the numerical precision chosen for deriving the frequency responses. Deep learning libraries often use single-precision floating-point numbers for all computations, which can result in incorrect frequency responses due to rounding errors. This problem is not straightforward to detect as, in the case of the tone stack model, the responses are miscalculated only with particular settings of the virtual potentiometers. A simple solution is to use double-precision numbers when deriving the polynomial coefficients of the state-space transfer function $H(z)$. The purple dash-dotted line in Fig. 5 represents an incorrect response of the tone stack model with the controls set to $c_\mathrm{b} = 1$, $c_\mathrm{m} = 0$, and $c_\mathrm{t} = 1$ when using single-precision.

3.3. Frequency Domain Tone Stack Filtering

To significantly speed up the training process, we use a similar procedure to [23], where a one-pole IIR filter was applied in the fre-

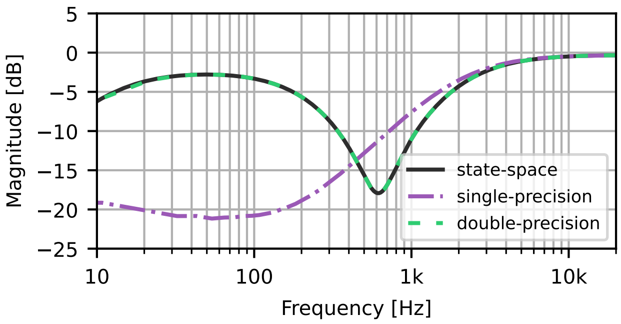

Figure 5: *Comparison of the ideal tone stack response with frequency sampled responses using single- and double-precision.*

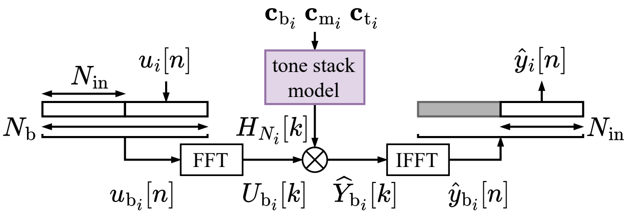

Figure 6: *Frequency domain filtering of an audio sub-segment with a tone stack frequency response.*

quency domain. We use truncated backpropagation through time (TBPTT) [42] for updating trainable parameters multiple times when processing longer audio segments. The TBPTT length determines N_{in}, which is the number of samples of the audio sub-segments used for training. Each TBPTT batch consists of s audio sub-segments $u_i[n]$ that get processed simultaneously during a single training step. The filtering process for a single audio sub-segment is depicted in Fig. 6.

First, an audio sub-segment $u_i[n]$ of length N_{in} is inserted into a buffer of length $N_{\mathrm{b}} = 2N_{\mathrm{in}}$. Then, we perform a fast Fourier transform (FFT) on the audio samples $u_{\mathrm{b}_i}[n]$ stored in the buffer, and discard the negative frequency components, resulting in $\lfloor N_{\mathrm{b}}/2 \rfloor + 1$ complex frequency coefficients $U_{\mathrm{b}_i}[k]$. Next, we supply the conditioning values $\mathbf{c}_{\mathrm{b}_i}$, $\mathbf{c}_{\mathrm{m}_i}$, and $\mathbf{c}_{\mathrm{t}_i}$ which affect the variable resistances of the tone stack model. After all virtual components of the tone stack model are updated, we can derive the state-space representation using the DK-method.

The tone stack transfer function $H_i(z)$ and frequency sampled response $H_{N_i}[k]$ is then computed. This is used to perform an element-wise multiplication of $H_{N_i}[k]$ with $U_{\mathrm{b}_i}[k]$, which results in filtered complex frequency coefficients $\hat{Y}_{\mathrm{b}_i}[k]$. Finally, we take the inverse FFT of $\hat{Y}_{\mathrm{b}_i}[k]$. Since the beginning of the filtered buffer $\hat{y}_{\mathrm{b}_i}[n]$ contains the starting transient, we only keep the last N_{in} samples, which allows for processing longer audio sequences without producing discontinuities in the filtered signal. As a result, there is no need to apply windowing functions during the filtering process. Note that we set the buffer size $N_{\mathrm{b}} = 4096$ in order to obtain 2049 complex coefficients after performing the FFT so it matches the length of the frequency sampled response $H_{N_i}[k]$.

4. MODEL STRUCTURE AND TRAINING

The proposed grey-box model is composed of three blocks, as shown in Figure 7. The first block is a long short-term memory

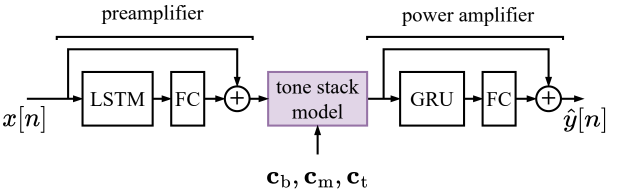
preamplifier
power amplifier
$x[n]$
LSTM
FC
tone stack model
GRU
FC
$\hat{y}[n]$
$\mathbf{c}_{\mathrm{b}}, \mathbf{c}_{\mathrm{m}}, \mathbf{c}_{\mathrm{t}}$

Figure 7: *Block diagram of the proposed grey-box model, which uses neural network models for the pre- and power- amplifier sections and a linear white-box model for the tone stack circuit.*

(LSTM) RNN [43] followed by the differentiable state-space tone stack model described in Sec. 3, and a gated recurrent unit (GRU) RNN [44]. The RNNs are used for modelling the preamplifier and power amplifier, respectively. Fully connected (FC) layers are added after each RNN to transform the hidden state vectors into single audio samples. We provide a reference PyTorch implementation, the dataset, and listening examples at[2].

The architecture of the RNNs is identical to a previously proposed black-box guitar amplifier model [28]. The hidden state size determines the accuracy of these models. An extensive hyperparameter search was conducted in previous works [28, 29] to assess the ideal hidden state sizes. We set the hidden size to 40 for the LSTM and to 8 in the case of the GRU. These hyperparameters were set empirically after initial training experiments on the dataset presented in Sec. 2.1.

Note that our grey-box model assumes that the preamplifier, tone stack, and power amplifier are decoupled in terms of their interaction. The Marshall JVM 410H has a feedback connection from the output transformer to the phase splitter, controlled by the “resonance” and “presence”. We did not investigate how this feedback connection affects the interaction between the amplifier sections.

4.1. Objective Metrics

All models in this work were trained to minimise the Error-To-Signal (ESR) loss, which has been used extensively for modelling nonlinear audio circuits [26, 27, 28, 29]. The ESR is given by

$$\mathcal{E}_{\mathrm{ESR}} = \frac{\sum_{n=0}^{N-1} |y[n] - \hat{y}[n]|^2}{\sum_{n=0}^{N-1} |y[n]|^2}, \tag{25}$$

where $y[n]$ is the target signal, $\hat{y}[n]$ is the predicted signal, and N is the length of the training segment.

Furthermore, we use a frequency domain error metric based on short-time Fourier transform (STFT) from [45] denoted as $\mathcal{E}_{\mathrm{STFT}}$ solely for validation purposes. It is a linear combination of spectral convergence and log-scale STFT-magnitude error. Contrary to the ESR, it discards the phase information and provides insight into how well the models perform regarding spectral similarity. We utilise this metric because phase differences between the model output and the target signal can result in high ESR with the model still performing well perceptually [21].

4.2. Training Hyperparameters

To train the models, we use similar hyperparameters to those proposed in [29]. The training dataset was split into 0.5 s audio segments. The first 1000 samples of each segment are processed without updating the parameters of the network, which allows for the

[2] https://stepanmk.github.io/grey-box-amp

Table 3: *Number of trainable parameters, duration of a training step on a GPU, and a real-time factor (RTF) at a sampling rate of 44.1 kHz for the baseline and proposed models.*

Model Type	Params.	Train. Step (s)	RTF
RNN (baseline [28])	10417	0.16	21.28
TS (tone stack from [16])	7208	26.59	11.49
TS (proposed recursive)	7208	14.17	19.61
TS (proposed freq. domain)	7208	0.45	19.61

initialisation of the RNN states and the tone stack buffer. The rest of the samples were processed with TBPTT being applied every 2048 samples. The training batch size was set to $s = 80$. We calculated the validation loss every 2 epochs, and the maximum number of training epochs was set to 350. We applied early stopping with a patience of 15 epochs whilst monitoring the validation loss. The models were trained with the Adam optimiser with an initial learning rate of 2×10^{-3}. The learning rate was decreased by a factor of 0.5 if the validation loss did not improve for 10 consecutive epochs. The training time of the models before early stopping varied but generally took approximately 10 to 40 min on a GPU, depending on the size of the training dataset.

5. EXPERIMENTS

We hypothesise that the white-box tone stack will greatly improve the ability of the model to generalise to the unseen values described in Sec. 2.1, especially when only a small number of parameter values are seen during training. To test this we train models using different subsets of the training dataset.

As a baseline, we use a fully black-box RNN model from [28] consisting of an LSTM of hidden size 48 followed by a FC layer. We use a larger hidden size for the baseline model to compensate for the fact that our proposed model includes an additional RNN stage after the tone stack model. The black-box baseline RNN model receives the conditioning values as additional input channels, and as such has an input size of 4.

Four different datasets were used for training, with the number of unique permutations of conditioning values varying from 1 to 21. The smallest dataset contains a single permutation, with all the tone stack controls set to the midpoint (5). The second dataset has 2 additional targets, where all the tone stack controls are set to either 0 or 10. The third dataset includes all the targets from the second dataset, in addition to cases where each tone stack control is set to either 0, 2, 8, or 10, whilst the remaining controls are set to 5. This results in a total of 15 tone stack parameter permutations in the third dataset. Finally, the last dataset includes the second dataset, as well cases where each tone stack control is varied in turn to either 0, 2, 4, 6, 8 or 10, whilst leaving the rest of the controls set to 5. This results in a total of 21 unique permutations for the fourth dataset. Note that we left the nonlinear potentiometer tapers non-trainable in the case of our TS1 model, as only a single conditioning value was presented during training.

6. RESULTS

The proposed model has less trainable parameters than the baseline, however, as shown in Table 3, it is slightly slower to train. Frequency sampling of the tone stack filter significantly improves the training times on a GPU. The difference in an average training step duration is even more pronounced when compared to a previously proposed differentiable tone stack model from [16], where the filtering is applied in the time domain. In addition, the previous approach uses a different discretisation scheme and does not allow direct derivation of the transfer function needed for frequency sampling.

Table 4: *Objective results for the baseline and proposed models. Bold indicates best-performing model. The numbering of the model names corresponds to the number of unique permutations of conditioning seen during training.*

	Test		Test Unseen		Train. Dataset
Model	$\mathcal{E}_{\mathrm{ESR}}$	$\mathcal{E}_{\mathrm{STFT}}$	$\mathcal{E}_{\mathrm{ESR}}$	$\mathcal{E}_{\mathrm{STFT}}$	Duration (min)
RNN1	0.020	0.793	1.748	2.573	4
RNN3	0.025	0.933	1.446	2.511	12
RNN15	0.017	0.743	0.040	0.953	60
RNN21	0.015	0.693	**0.039**	**0.893**	84
TS1	0.011	0.760	0.048	0.854	4
TS3	0.020	0.764	**0.023**	**0.715**	12
TS15	0.029	0.780	0.038	0.805	60
TS21	0.028	0.924	0.039	0.955	84

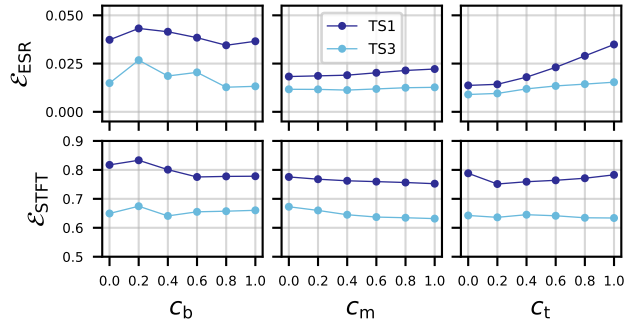

Figure 8: *Further evaluation of the proposed TS1 and TS3 models on unseen tone stack permutations: (top) time domain and (bottom) spectral errors for (left) bass, (center) middle, and (right) treble controls.*

Furthermore, we measured how long it takes to process 1 s of audio with a custom C++ implementation of our model expressed as a real-time factor (RTF) computed according to [18]. An RTF larger than 1 means the model can process the signal faster than in real time. The baseline model was found to be negligibly faster than the proposed model, as shown in Table 3.

Objective metrics for all trained models are shown in Table 4. In the case of the baseline RNN models, it is clear that the more tone stack permutations the models see during training, the better they perform on unseen permutations. This is evident from both the time and frequency domain metrics. The RNN21 that was trained on 84 min of data performed the best in comparison to other baseline models. In contrast to this, the proposed models performed well even when trained on very small datasets. The TS1 model, trained on just a single tone stack setting (4-min dataset), outperformed the best-performing baseline model in terms of STFT error computed on unseen parameter values. The TS3 performed the best of all the models, producing the smallest error on unseen data, for both metrics.

As the TS1 and TS3 models were only trained on 1 and 3 tone stack settings, we may also consider the rest of the training dataset as unseen, and use this to further evaluate the performance of these models, as shown in Fig. 8. It was observed during training that for the models TS15 and TS21, the additional data seemed to impact the model training negatively. An abrupt increase in the validation loss occurs after a few epochs, and this behaviour was observed consistently across different training runs. One possible explanation could be that the trainable potentiometer tapers used in the tone stack model might be sensitive to errors caused by manually setting the tone stack controls on the real amplifier when recording the dataset.

6.1. Listening Tests

A MUSHRA [46] listening test was conducted to evaluate the perceptual quality of the models. In each trial, participants were presented with a reference clip that was processed by the guitar amplifier being modelled. Participants were asked to rate seven test conditions on a scale of 0 to 100, based on perceived similarity to the reference. The test conditions included five neural network models: 3 black-box RNN models, which were trained using either 1, 3, or 21 unique permutations of tone stack parameters, and 2 versions of our proposed grey-box model, trained with either 1 or 3 permutations of tone stack parameters. These were selected based on the objective results of the previous section. Additionally an anchor, created by processing the input with a $\tanh$ nonlinearity, and a hidden reference, were included in the test.

Fourteen participants completed the listening tests. Two participants identified as female and twelve as male. All participants had experience completing listening tests, and their mean age was 28.5 years. One participant was excluded from the results as they rated the hidden reference below 90 in more than 15% of the trials.

The listening test was conducted for two different tone stack settings that the models had not seen during training. Results of both test scenarios are shown in Fig. 9. First, the tone stack parameters were set to $c_\mathrm{b} = 0.1$, $c_\mathrm{m} = 0.3$, and $c_\mathrm{t} = 0.7$. In this case, the conditioning values were in-between the steps used to capture the training dataset. This represents a case that should easier for the model to predict accurately. The RNN1 and RNN3 baseline models were rated as Poor and Fair, respectively. Interestingly the RNN3 model, which was trained on more data than RNN1, performed worse. The RNN21 model, on the other hand, was rated as excellent as it was trained on a substantially larger dataset. Our proposed TS1 and TS3 models were both rated as Excellent, showing that adding the tone stack model greatly improves generalisation, even though the models were trained only on 4 and 12 min data, respectively.

In the second scenario, the tone stack parameters were set to $c_\mathrm{b} = 1$, $c_\mathrm{m} = 0$, and $c_\mathrm{t} = 0$. In this case, the RNN models trained on the small datasets were rated the worst, as in the first scenario. The RNN21 model trained on the largest dataset was only rated as Fair, contrary to how it was rated in the first scenario. This shows that the RNN models must be supplied with additional permutations of the tone stack parameters to generalise better. Conversely, the TS1 and TS3 models with the tone stack included were again rated as Excellent as shown in the bottom half of Fig. 9. The TS3 model trained on 12 min of data was rated slightly worse than the TS1 model, in contrast to the first test scenario. This also goes against the objective results, however informal listening tests confirm only slight differences in the high frequencies. We encourage readers to evaluate the models for themselves by listening to the sound examples provided on the demo page.

7. CONCLUSIONS

In this work we present a grey-box approach for guitar amplifier modelling, which allows for the inclusion of user parameters whilst requiring minimal training data. The proposed approach can be applied to many popular guitar amplifiers, as the modelled tone stack circuit is ubiquitous in the industry. We also demonstrated how a state-space model of a linear parametric circuit can be implemented using the frequency sampling method, allowing for efficient training within a deep learning framework. A subjective and objective evaluation of our proposed method demonstrates excellent generalisation to unseen data and excellent perceptual quality. Future work should validate our approach on other devices as our study was limited to a single amplifier. Using the DK-method combined with frequency sampling should also allow for incorporating other controllable linear circuits into neural network models of guitar pedals and various analogue effects.

We acknowledge that modelling different guitar amplifiers, which produce very high amounts of distortion, could result in the need to use RNNs with larger hidden sizes, thus making the model more expensive to run in real time. However, recent work [47] has shown that it is possible to prune the trainable weights of black-box RNN guitar amplifier models, resulting in a smaller effective hidden size whilst slightly improving the perceptual modelling quality. It is likely that this method could also be applied to our grey-box model.

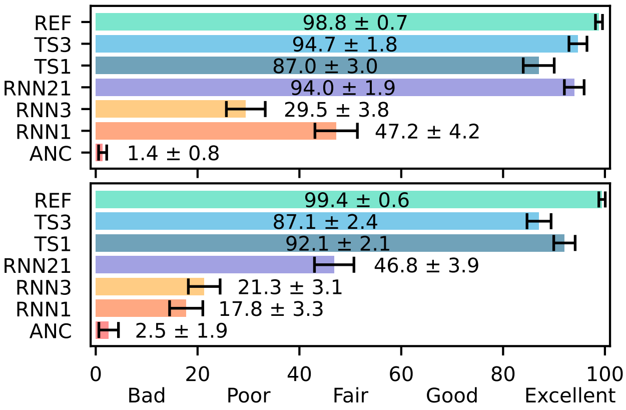

Figure 9: *MUSHRA scores with 95% confidence intervals for the tone stack settings of* $c_\mathrm{b} = 0.1$*,* $c_\mathrm{m} = 0.3$*, and* $c_\mathrm{t} = 0.7$ *(top) and* $c_\mathrm{b} = 1$*,* $c_\mathrm{m} = 0$*, and* $c_\mathrm{t} = 0$ *(bottom).*

8. REFERENCES

[1] V. Välimäki, S. Bilbao, J. O. Smith, J. Pakarinen, and D. Berners, *DAFX: Digital Audio Effects*, chapter "Virtual analog effects", pp. 473–522, Wiley, Chichester, UK, second edition, 2011.

[2] K. J. Werner, *Virtual Analog Modeling of Audio Circuitry Using Wave Digital Filters*, Ph.D. thesis, Stanford University, 2016.

[3] S. D'Angelo, "Lightweight virtual analog modeling," in *Proc. 22nd Colloquium on Music Informatics*, Udine, Italy, 2018, pp. 20–23.

[4] F. Eichas and U. Zölzer, "Black-box modeling of distortion circuits with block-oriented models," in *Proc. Int. Conf. Digital Audio Effects (DAFx-16)*, Brno, Czech Republic, Sep. 2016.

[5] D. T. Yeh, J. S. Abel, and J. O. Smith III, "Automated physical modeling of nonlinear audio circuits for real-time audio effects—part I: Theoretical development," *IEEE/ACM Trans. Audio Speech Lang. Process.*, vol. 18, no. 4, pp. 728–737, Oct. 2010.

[6] M. Holters and U. Zölzer, "A generalized method for the derivation of non-linear state-space models from circuit schematics," in *Proc. 23rd European Signal Processing Conf.*, Nice, France, Sep. 2015.

[7] K. Dempwolf, M. Holters, and U. Zölzer, "Discretization of parametric analog circuits for real-time simulations," in *Proc. Int. Conf. Digital Audio Effects (DAFx-10)*, Graz, Austria, Sep. 2010.

[8] G. De Sanctis and A. Sarti, "Virtual analog modeling in the wave-digital domain," *IEEE Trans. Audio Speech Lang. Process.*, vol. 18, no. 4, pp. 715–727, May 2010.

[9] K. J. Werner, A. Bernardini, J. O. Smith, and A. Sarti, "Modeling circuits with arbitrary topologies and active linear multiports using wave digital filters," *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 65, no. 12, pp. 4233–4246, Dec. 2018.

[10] J. Macak, J. Schimmel, and M. Holters, "Simulation of Fender type guitar preamp using approximation and state-space model," in *Proc. Int. Conf. Digital Audio Effects (DAFx-12)*, York, UK, Sep. 2012.

[11] F. Eichas, M. Fink, M. Holters, and U. Zölzer, "Physical modeling of the MXR Phase 90 guitar effect pedal," in *Proc. Int. Conf. Digital Audio Effects (DAFx)*, Erlangen, Germany, Sep. 2014.

[12] M. Holters and U. Zölzer, "Physical modelling of a wah-wah effect pedal as a case study for application of the nodal DK method to circuits with variable parts," in *Proc. Int. Conf. Digital Audio Effects (DAFx-11)*, Paris, France, Sep. 2011.

[13] K. Dempwolf and U. Zölzer, "A physically-motivated triode model for circuit simulations," in *Proc. Int. Conf. Digital Audio Effects (DAFx-11)*, Paris, France, Sep. 2011.

[14] B. Holmes and M. Van Walstijn, "Physical model parameter optimisation for calibrated emulation of the dallas rangemaster treble booster guitar pedal," in *Proc. Int. Conf. Digital Audio Effects (DAFx-16)*, Brno, Czech Republic, Sep. 2016.

[15] J. H. Engel, L. Hantrakul, C. Gu, and A. Roberts, "DDSP: Differentiable digital signal processing," *arXiv preprint*, 2020, arXiv:2001.04643.

[16] F. Esqueda, B. Kuznetsov, and J. D. Parker, "Differentiable white-box virtual analog modeling," in *Proc. Int. Conf. Digital Audio Effects (DAFx20in21)*, Vienna, Austria, Sep. 2021.

[17] J. Chowdhury and C. J. Clarke, "Emulating diode circuits with differentiable wave digital filters," in *Proc. Int. Sound and Music Computing Conf. (SMC-22)*, Saint-Étienne, France, Jun 2022, pp. 332–339.

[18] C. C. Darabundit, D. Roosenburg, and J. O. Smith III, "Neural net tube models for wave digital filters," in *Proc. Int. Conf. Digital Audio Effects (DAFx20in22)*, Vienna, Austria, Sep. 2022.

[19] J. Pakarinen and D. T. Yeh, "A review of digital techniques for modeling vacuum-tube guitar amplifiers," *Computer Music J.*, vol. 33, no. 2, pp. 85–100, 2009.

[20] J. D. Parker, F. Esqueda, and A. Bergner, "Modelling of nonlinear state-space systems using a deep neural network," in *Proc. Int. Conf. Digital Audio Effects (DAFx-19)*, Birmingham, UK, Sep. 2019.

[21] F. Eichas and U. Zölzer, "Gray-box modeling of guitar amplifiers," *J. Audio Eng. Soc.*, vol. 66, no. 12, pp. 1006–1015, Dec. 2018.

[22] R. Kiiski, F. Esqueda, and V. Välimäki, "Time-variant gray-box modeling of a phaser pedal," in *Proc. Int. Conf. Digital Audio Effects (DAFx-16)*, Brno, Czech Republic, Sep. 2016.

[23] A. Wright and V. Välimäki, "Grey-box modelling of dynamic range compression," in *Proc. Int. Conf. Digital Audio Effects (DAFx20in22)*, Vienna, Austria, Sep. 2022.

[24] Z. Zhang, E. Olbrych, J. Bruchalski, T. J. McCormick, and D. L. Livingston, "A vacuum-tube guitar amplifier model using long/short-term memory networks," in *Proc. IEEE SoutheastCon*, Saint Petersburg, FL, April 2018.

[25] T. Schmitz and J.-J. Embrechts, "Nonlinear real-time emulation of a tube amplifier with a long short time memory neural-network," in *Proc. Audio Eng. Soc. 144th Conv.*, Milan, Italy, May 2018.

[26] E.-P. Damskägg, L. Juvela, and V. Välimäki, "Real-time modeling of audio distortion circuits with deep learning," in *Proc. Int. Sound and Music Computing Conf. (SMC-19)*, Malaga, Spain, May 2019, pp. 332–339.

[27] E.-P. Damskägg, L. Juvela, E. Thuillier, and V. Välimäki, "Deep learning for tube amplifier emulation," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP19)*, Brighton, UK, May 2019.

[28] A. Wright, E.-P. Damskägg, L. Juvela, and V. Välimäki, "Real-time guitar amplifier emulation with deep learning," *Appl. Sci.*, vol. 10, no. 3, Jan. 2020.

[29] A. Wright, E.-P. Damskägg, and V. Välimäki, "Real-time black-box modelling with recurrent neural networks," in *Proc. Int. Conf. Digital Audio Effects (DAFx-19)*, Birmingham, UK, Sep. 2019.

[30] L. Juvela et al., "End-to-end amp modeling: from data to controllable guitar amplifier models," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP23)*, Rhodes Island, Greece, Jun 2023.

[31] D. T. Yeh and J. O. Smith, "Discretization of the '59 Fender Bassman tone stack," in *Proc. Int. Conf. Digital Audio Effects (DAFx-06)*, Montreal, Canada, Sep. 2006.

[32] J. Abeßer, P. Kramer, C. Dittmar, and G. Schuller, "Parametric audio coding of bass guitar recordings using a tuned physical modeling algorithm," in *Proc. Int. Conf. Digital Audio Effects (DAFx-13)*, Maynooth, Ireland, Sep. 2013.

[33] C. Kehling, J. Abeßer, C. Dittmar, and G. Schuller, "Automatic tablature transcription of electric guitar recordings by estimation of score- and instrument-related parameters," in *Proc. Int. Conf. Digital Audio Effects (DAFx-14)*, Erlangen, Germany, Sep. 2014.

[34] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," *Adv. Neural Inf. Process. Syst.*, vol. 32, pp. 8024–8035, 2019.

[35] D. T. Yeh, "Automated physical modeling of nonlinear audio circuits for real-time audio effects—part II: BJT and vacuum tube examples," *IEEE/ACM Trans. Audio Speech Lang. Process.*, vol. 20, no. 4, pp. 1207–1216, May 2012.

[36] B. Holmes and M. van Walstijn, "Potentiometer law modelling and identification for application in physics-based virtual analogue circuits," in *Proc. Int. Conf. Digital Audio Effects (DAFx-19)*, Birmingham, UK, Sep. 2019.

[37] S. Nercessian, "Neural parametric equalizer matching using differentiable biquads," in *Proc. Int. Conf. Digital Audio Effects (DAFx-20)*, Vienna, Austria, Sep. 2020.

[38] S. Nercessian, A. Sarroff, and K. J. Werner, "Lightweight and interpretable neural modeling of an audio distortion effect using hyperconditioned differentiable biquads," in *Proc. ICASSP21*, Toronto, Canada, June 2021, pp. 890–894.

[39] S. Lee, H. Choi, and K. Lee, "Differentiable artificial reverberation," *IEEE/ACM Trans. Audio Speech Lang. Process.*, vol. 30, pp. 2541–2556, July 2022.

[40] C. J. Steinmetz, N. J. Bryan, and J. D. Reiss, "Style transfer of audio effects with differentiable signal processing," *J. Audio Eng. Soc.*, vol. 70, no. 9, pp. 708–721, Sept. 2022.

[41] B. Kuznetsov, J. D. Parker, and F. Esqueda, "Differentiable IIR filters for machine learning applications," in *Proc. Int. Conf. Digital Audio Effects (DAFx-20in21)*, Vienna, Austria, Sep. 2020.

[42] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[43] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Dec. 1997.

[44] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint*, 2014, arXiv:1412.3555 [cs.NE].

[45] R. Yamamoto, E. Song, and J. Kim, "Probability density distillation with generative adversarial networks for high-quality parallel waveform generation," *arXiv preprint*, 2019, arXiv:1904.04472v2.

[46] ITU, "BS.1534: Method for the subjective assessment of intermediate quality levels of coding systems," Geneva, Switzerland, 2015.

[47] D. Südholt, A. Wright, C. Erkut, and V. Välimäki, "Pruning deep neural network models of guitar distortion effects," *IEEE/ACM Trans. Audio Speech Lang. Process.*, vol. 31, pp. 256–264, 2023.