ANTIALIASED BLACK-BOX MODELING OF AUDIO DISTORTION CIRCUITS USING REAL LINEAR RECURRENT UNITS

Fabián Esqueda and Shogo Murai

KORG Inc.
Tokyo, Japan
`fabian@korg.berlin`
`murai@korg.co.jp`

ABSTRACT

In this paper, we propose the use of real-valued Linear Recurrent Units (LRUs) for black-box modeling of audio circuits. A network architecture composed of real LRU blocks interleaved with nonlinear processing stages is proposed. Two case studies are presented, a second-order diode clipper and an overdrive distortion pedal. Furthermore, we show how to integrate the antiderivative antialiaisng technique into the proposed method, effectively lowering oversampling requirements. Our experiments show that the proposed method generates models that accurately capture the nonlinear dynamics of the examined devices and are highly efficient, which makes them suitable for real-time operation inside Digital Audio Workstations.

1. INTRODUCTION

Data-driven virtual analog (VA) modeling of audio circuits, commonly known as "black-box" modeling, is an established research area in musical signal processing. In this approach, the internal mechanisms of the device under study remain unknown; instead, the method focuses on fitting parameters within a predefined structure to replicate measured input–output relationships. In contrast, "white-box" modeling focuses on analyzing circuit schematics to derive the underlying system equations, which are then discretized. Although effective, white-box modeling is typically laborious, which has motivated substantial research into automated modeling frameworks that streamline this process [1–3]. In scenarios where schematics are unavailable, black-box techniques remain the only viable option.

With the advent of machine learning (ML), recent research in black-box modeling has increasingly leveraged different neural network (NN) architectures. In [4], for example, a Convolutional Neural Network (CNN) based on WaveNet was used to emulate a tube amplifier. Meanwhile, a substantial body of research has focused on Recurrent Neural Networks (RNNs) for circuit modeling. Because RNNs are designed to handle sequential data with inherent memory and nonlinear relationships, they are a compelling choice for VA. In [5] two well-known RNN architectures–the Long Short-Term Memory (LSTM) and the Gated Recurrent Unit (GRU)–are compared for modeling a guitar distortion circuit and a tube amplifier, demonstrating performance comparable to earlier CNN-based approaches at lower computational costs during inference.

Despite these advantages, RNNs present significant training challenges, primarily due to vanishing and exploding gradient problems [6]. Furthermore, their recursive nature precludes parallelization across the time dimension, hindering the use of arbitrary sequence lengths during training. These constraints have motivated research into alternative ML architectures, notably Transformers with their attention mechanisms [7] and *state-space models* (SSMs) [8], which offer improved parallel processing capabilities while maintaining temporal coherence.

SSMs originate from control theory, where they are used to describe dynamical systems through ODEs that govern the evolution of a set of hidden states over time. It is important to point out that, while state-space formulations can accommodate nonlinear dynamics, this work (like most ML literature on SSMs) focuses only on linear SSMs. This represents a key difference with traditional RNNs, which are inherently nonlinear. Defining SSMs as purely linear processing blocks enables parallelization across time and prevents vanishing or exploding gradients. In previous research, [9] proposed an approach in which the trajectories of the system states are learned through a residual network trained on actual state data, which in the case of audio circuits corresponds mainly to the voltages across memory-storing elements such as inductors and capacitors. While highly effective, this approach is also relatively laborious, as it requires direct access to the internal states of the device.

In [10], Orvieto *et al.* introduced a structure called the Linear Recurrent Unit (LRU), a SSM specifically designed for tasks with long memory dependencies. The LRU achieves its performance through several key characteristics, including a diagonal structure that enables efficient training on arbitrary sequence lengths and an exponential parameterization that ensures model stability. Since LRUs are linear time-invariant (LTI) units, the authors propose interleaving them with standard ML nonlinear blocks–such as Multi-Layer Perceptrons (MLPs) or Gated Linear Units (GLUs)–to capture any nonlinear relationships in the data. Although this decoupling of linear and nonlinear components precludes Turing completeness, Orvieto *et al.* demonstrate that stacking multiple LRUs with nonlinear elements provides a high level of expressivity perfectly capable of emulating any nonlinear dynamics seen in the data, provided the network has the right size [10].

In this work, we propose an application of LRUs for black-box modeling of analog audio distortion circuits. We investigate the adaptation of Orvieto *et al.*'s original complex-valued formulation to a purely real-valued one, while examining how network hyperparameters–including state size, hidden layer size, and network depth–influence modeling accuracy. As case studies, we model two guitar pedals: a distortion and an overdrive circuit. Lastly, we show how to integrate antialiasing during inference.

The use of LRUs for VA modeling was first proposed in [11], where the authors conducted a comprehensive comparative analysis of five distinct ML architectures across multiple audio effects. This work demonstrated the suitability of LRUs for VA tasks, as they outperformed RNNs and other SSM architectures for some of the effects evaluated. A fundamental distinction between this prior research and our work lies in the implementation of the LRU blocks and in the overall NN architecture. In [11], the LRUs are implemented across the so-called "feature" dimension as opposed to the temporal one. To do so, the authors defined a fixed window size of 64 input samples which are then compressed into the hidden size before being processed by the LRU matrices. This approach introduces a 64-sample processing latency, as the network generates a single output sample for every 64-sample window. In contrast, our approach applies LRUs directly along the time dimension, avoiding latency, supporting parallelization during training, and enabling antialiasing through a time-domain method. We also experiment with deeper structures than those discussed in [11], highlighting the significance of this implementation aspect.

This paper is organized as follows. Section 2 presents the theoretical background on SSMs and LRUs. Section 3 introduces our network architecture, and Section 4 explains the integration of antialiasing into our method. Section 5 outlines our experimental methodology, while Section 6 details our results, including real-time network run times. Finally, Section 7 offers concluding remarks and directions for future research.

2. BACKGROUND

The state-space formulation is a mathematical representation used to describe the behavior of a system through a set of states that evolve over time in response to a driving input and to the system's internal dynamics [12]. In the continuous-time domain, a system can be written in state-space form as:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= A\mathbf{x}(t) + B\mathbf{u}(t)\\ \mathbf{y}(t) &= C\mathbf{x}(t) + D\mathbf{u}(t)\end{aligned} \tag{1}$$

where $\mathbf{u}$, $\mathbf{y}$ and $\mathbf{x}$ are vectors representing the inputs, outputs and states of the system, respectively, and t is time. A, B, C and D are matrices that describe the behavior of the system. Matrix A, in particular, determines its stability, as it describes the evolution of the states over time.

In the discrete domain, assuming a constant time step (i.e. constant sampling rate), system (1) is written as:

$$\mathbf{x}_{n+1} = \widetilde{A}\mathbf{x}_n + \widetilde{B}\mathbf{u}_n \tag{2}$$

$$\mathbf{y}_n = \widetilde{C}\mathbf{x}_n + \widetilde{D}\mathbf{u}_n \tag{3}$$

where n is the discrete-time sample index ($\mathbf{x}_n \equiv \mathbf{x}[n]$), and matrices $\widetilde{A}$–$\widetilde{D}$ are discretized forms of A–D. The relationship between A–D and $\widetilde{A}$–$\widetilde{D}$ will depend on the discretization scheme used [10]. For sequence modeling tasks, matrices $\widetilde{A}$–$\widetilde{D}$ can be seen as the parameters to be learned via gradient descent.

As shown by (2), state-space systems are recursive. The current state vector is needed to compute the next one and so on. RNNs operate in the same recursive manner, with the exception that the relationship between successive states is nonlinear. This recursion forces sequential processing during training, leading to longer training times compared to architectures such as CNN that can be parallelized across time.

To understand how linear SSMs tackle this computational constraint, we observe what happens when we evaluate the first three steps of the general discrete state-space model. Assuming $\mathbf{x}_0 = 0$ and rewriting $\widetilde{A}$–$\widetilde{D}$ as A–D for simplicity, we get:

$$\begin{array}{lll}\mathbf{x}_1 = B\mathbf{u}_0 & \mathbf{x}_2 = A\mathbf{x}_1 + B\mathbf{u}_1 & \mathbf{x}_3 = A\mathbf{x}_2 + B\mathbf{u}_2\\ \mathbf{y}_0 = D\mathbf{u}_0, & \mathbf{y}_1 = C\mathbf{x}_1 + D\mathbf{u}_1, & \mathbf{y}_2 = C\mathbf{x}_2 + D\mathbf{u}_2.\end{array}$$

Focusing only on the state evaluation, we can further 'unroll' the recursion to derive an expression for $\mathbf{x}_n$ in terms of $\mathbf{u}$ exclusively:

$$\begin{aligned}\mathbf{x}_1 &= B\mathbf{u}_0\\ \mathbf{x}_2 &= AB\mathbf{u}_0 + B\mathbf{u}_1\\ \mathbf{x}_3 &= A^2B\mathbf{u}_0 + AB\mathbf{u}_1 + B\mathbf{u}_2\\ &\vdots\\ \mathbf{x}_n &= A^{n-1}B\mathbf{u}_0 + A^{n-2}B\mathbf{u}_1 + \ldots + B\mathbf{u}_{n-1}\end{aligned} \tag{4}$$

At this point a pattern emerges and we conclude that

$$\mathbf{x}_n = \sum_{k=0}^{n-1} A^k B\mathbf{u}_{n-k}. \tag{5}$$

Therefore, by unrolling the recursion the evaluation of the state vector $\mathbf{x}$ can be reformulated as a feedforward operation. This enables us to parallelize the evaluation of the state vector across the time dimension using techniques such as the *parallel scan* algorithm [13, 14]. Furthermore, by examining (5), we observe that computing $\mathbf{x}$ requires evaluating powers of the A matrix. This introduces a new computational bottleneck, as the complexity will once again scale exponentially with sequence length. To address this challenge, Gu *et al.* proposed constraining A to be diagonal [8], effectively replacing the expensive process of computing repeated matrix powers with the more efficient operation of computing powers of individual diagonal elements.

It is essential to point out that the form (5) is used exclusively during training. For inference, we revert back to the original recursive form (2), which is capable of processing data one sample at a time and is suitable for latency-free real-time implementations.

One final general observation regarding deep learning architectures that incorporate SSMs is their connection with Koopman-based models. In Koopman theory, nonlinear dynamical systems are transformed into infinite-dimensional linear systems in the space of observables [15]. This perspective allows nonlinear systems to be analyzed using linear operator theory. As will be exemplified later, the learned state dimensionality of SSM-based neural networks is typically higher than that of the underlying systems. From a theoretical perspective, this can be interpreted as an attempt to learn a finite-dimensional approximation of the Koopman operator. This is discussed, for instance, in Appendix E of [10]. A comparison between SSMs and Koopman-based models in the context distributed acoustic systems can be found in [16].

2.1. Linear Recurrent Units

Building on the work presented in [8], Orvieto *et al.* introduced in [10] the LRU, a diagonal SSM that can be written directly in discrete time as:

$$\mathbf{x}_{n+1} = \Lambda\mathbf{x}_n + B\mathbf{u}_n \tag{6}$$

$$\mathbf{y}_n = C\mathbf{x}_n + D\mathbf{u}_n \tag{7}$$

where $\Lambda = \text{diag}(\lambda_1, \lambda_2, ..., \lambda_N) \in \mathbb{C}^{N\times N}$, $B \in \mathbb{C}^{N\times H}$, $C \in \mathbb{C}^{H\times N}$ and $D \in \mathbb{R}^{H\times H}$. Parameters N and H denote the size of the state vector $\mathbf{x}$, and the input/output vectors $\mathbf{u}$ and $\mathbf{y}$, respectively. As explained by the authors, the use of complex numbers emerges based on the fact that an arbitrary dense matrix $A \in \mathbb{R}^{N\times N}$ with linearly independent eigenvectors can be diagonalized into a diagonal complex matrix. Defining the state matrix to be complex diagonal means each element will represent a system pole and system stability can therefore be guaranteed by ensuring these remain inside the unit circle during both inference and training. Additionally, it allows the network to learn oscillatory behavior.

In addition to the discussed discrete-domain definition, LRUs have two main features. The first is an exponential parameterization where each element in the state matrix is defined as $\lambda_j := \exp(-\exp(\nu_j^{\log}) + i\theta_j)$, where $\boldsymbol{\nu}^{\log}$ is the vector of parameters to be optimized and is initialized as $\nu_j^{\log} := \log(\nu_j)$ for $\nu_j \in [0, 1]$. Parameter θ determines the angle of the complex number and is initialized between $[0, 2\pi]$. The purpose of this exponential parametrization is twofold, to help ensure the stability of the system and to add more resolution to the area near the unit circle [10].

A second important feature of LRUs is the use of a normalization strategy, which modifies the state update as

$$\mathbf{x}_{n+1} = \Lambda\mathbf{x}_n + \exp(\boldsymbol{\gamma}^{\log}) \odot B\mathbf{u}_n, \tag{8}$$

where $\boldsymbol{\gamma}^{\log} \in \mathbb{R}^N$ is a trainable parameter vector initialized via the state matrix $\gamma_j^{\log} := \log(\sqrt{1 - |\lambda_j|^2})$. For a more detailed description of the features LRUs the reader is referred to [10].

2.2. Proposed Modifications

In this work, we propose the use of LRUs as part of a larger NN architecture designed to model distortion circuits. Our approach adheres, for the most part, to the framework established by Orvieto *et al.*, with one significant modification regarding the LRU definition. In [10], the authors emphasize the importance of initializing the entries of Λ to lie close to the unit circle and in the vicinity of the real axis (i.e. highly resonant and low frequency modes). During our initial experiments we were able to confirm this, and in fact observed that the best results were obtained when θ was initialized to be very close to zero. This empirical finding, together with recent research on diagonal SSMs that explores the use of purely real-valued matrices [17]–albeit in different applications and architectures–motivated us to constrain Λ to be purely real. Although this design decision precludes the network from learning oscillatory behavior, such as that of self-oscillating synthesizer filters, it proved to be suitable for the systems examined in this study. Moreover, it has the added advantage of significantly reducing the complexity of the implementation across different platforms, as some systems might not provide native support for complex numbers. From a system dynamics point of view this design choice means the modes in our system are characterized exclusively by exponentially decaying dynamics.

Following this modification, our LRU block is still defined by the equations (7) and (8), but with matrices $\Lambda \in \mathbb{R}^N$, $B \in \mathbb{R}^{N\times H}$, $C \in \mathbb{R}^{H\times N}$ and $D \in \mathbb{R}^H$. Since we continue to use the two LRU properties described in the previous section, namely the exponential parameterization and the normalization strategy, as well as several of the implementation details originally presented by Orvieto *et al.* (see Sec. 3 and Sec. 5), we simply refer to this model as a Real Linear Recurrent Unit.

3. NETWORK ARCHITECTURE

Figure 1 shows a block diagram representation of the proposed architecture as used during training. This design interleaves the LTI real LRU stages with nonlinear processing elements to capture the nonlinear relationships present in the data, as is standard in SSM-based ML architectures. For example, prior works such as [8] and [10] combine diagonal SSMs with MLPs and GLUs. For each nonlinear stage we chose a streamlined approach consisting of a nonlinear activation function followed by a single fully connected (FC) layer. This design is similar to the one in [11], but in reverse order. We chose this reversal–and also omitted an additional FC layer before the activation function–because either option would have resulted in two consecutive linear operations. Since the input to the nonlinear block is computed by a linear operation, a pre-activation FC layer can be seen as redundant, with its weights effectively absorbable into those of the LRU.

The proposed network expects an input tensor with dimensions $(B, L, 1)$ where B is batch size and L is sequence length. The last dimension is kept at 1 for monophonic audio input. The first component of the network (highlighted in red) is an FC layer that projects the input tensor to the internal hidden dimension H. Biases are disabled for this layer. The next stage is the real LRU block, which maps the input tensor to the state size N and maps it back to H via matrices C and D following the state update. The output of the LRU block is then passed through a series of nonlinear saturating activations applied element-wise, followed by an additional FC layer. Section 4 discusses the choice of activation function. For simplicity, the width of this nonlinear stage is also defined as H. Lastly, the output of the nonlinear stage is mixed with the input to the real LRU block via a skip connection. This combined pair–comprised of the real LRU and the nonlinear stage with skip connection–is repeated D times, determining the *depth* of the network. Finally, the output of the D^{th} block is mapped back to the original input dimension through an FC layer (also highlighted in red) with biases disabled.

An interesting observation that can be made about the proposed structure is that it is reminiscent of block-oriented structures such as Wiener-Hammerstein and related models, which have in the past been used for black-box VA modeling tasks [18]. Indeed, one can view this proposed structure as a multi-dimensional, deep, and highly optimized Wiener–Hammerstein style model. However, a thorough examination of the equivalences between SSMs and block-oriented structures lies outside the scope of this study.

4. ANTIALIASING

As previously mentioned, one key advantage of decoupling the linear and nonlinear components inside the NN is that we can employ established antialiasing techniques, such as antiderivative antialiasing (ADAA), with relative ease. First introduced by Parker *et al.* [19], ADAA is a technique to mitigate aliasing in memoryless nonlinearities of the form $y_n = f(x_n)$, where $f(\cdot)$ is a nonlinear mapping. In its first-order form, ADAA is defined as

$$y_n = \frac{F(x_n) - F(x_{n-1})}{x_n - x_{n-1}}, \tag{9}$$

where $F(\cdot)$ is the first antiderivative of $f(\cdot)$. When implemented in a finite-precision context, (9) becomes ill-conditioned if $x_n \approx x_{n-1}$. To avoid this, we can replace (9) with the averaged evalua-

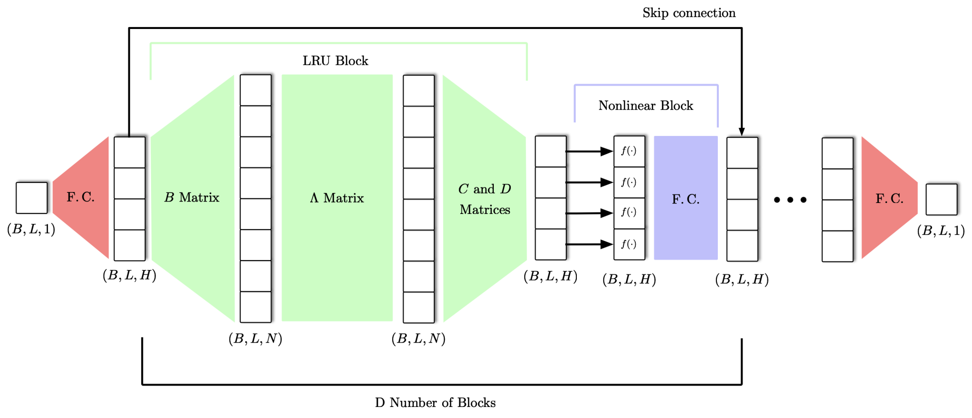

Figure 1: *Block diagram representation of the proposed network architecture. As an example, we use a network with state size $N = 8$ and hidden size $H = 4$. The abbreviation "F. C." denotes fully connected, or dense, layers*

tion $y_n = f((x_n + x_{n-1})/2)$ whenever $\epsilon > |x_n - x_{n-1}|$, where ϵ is a predetermined threshold.

Now, our proposed architecture features a nonlinear processing stage after each real LRU block (see Fig. 1). Each of these stages consists of an element-wise activation function followed by an FC layer. Since the processing blocks in the proposed structure are memoryless, they lend themselves naturally to ADAA. A typical choice for the activation function in VA modeling tasks is the hyperbolic function $\tanh(\cdot)$. Substituting these nonlinear activations with antialiased forms is in fact somewhat of a trivial process; however, a key limitation arises due to the black-box and multi-dimensional nature of neural networks, again relating to ill-conditioning.

In standard ADAA, parameter ϵ is set according to the expected input signal range, which is typically known. In a ML context, however, the distribution of signals at various points in the network–both during training and inference–is generally not known, making it difficult to select a suitable value for ϵ. This challenge is compounded by the fact that the total number of nonlinearities, given by $H \times D$, can be large, and there is no guarantee that thresholds can be shared across different parts of the network. Although normalization strategies may partially alleviate this problem, they can also influence the outcome of training, making them an unsuitable solution as we would like to decouple this behavior. During our experiments, we were unable to find suitable ill-conditioning thresholds that ensured the correct operation of the network for arbitrary input.

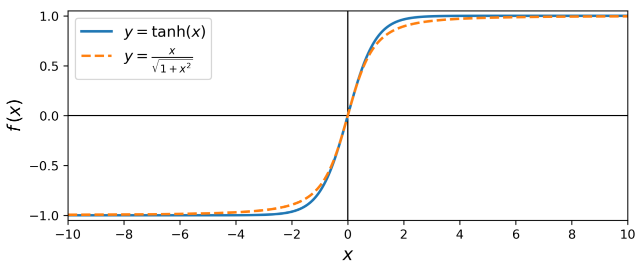

Figure 2: *Comparison between the classic* $\tanh(x)$ *activation function and its algebraic approximation.*

Rather than devising a procedure to find optimal ill-conditioning thresholds, we pursue an alternative that eliminates the need for them altogether. We replace the standard $\tanh(\cdot)$ activation with the saturating function

$$f(x) = \sin(\arctan(x)), \tag{10}$$

which can be written algebraically as

$$f(x) = \frac{x}{\sqrt{1 + x^2}}. \tag{11}$$

Figure 2 shows the input–output relation of this function plotted against $\tanh(\cdot)$. Although not a perfect one-to-one match, we can observe that both functions exhibit the same general saturating behavior and converge to ± 1 at the respective ends of their input range.

Equation (11) is exhibits a desirable property when used in conjunction with ADAA. Its antiderivative appears in closed form as

$$F(x) = \sqrt{1 + x^2}. \tag{12}$$

Substituting (12) into (9) yields the antialiased form

$$y_n = \frac{\sqrt{1 + x_n^2} - \sqrt{1 + x_{n-1}^2}}{x_n - x_{n-1}}, \tag{13}$$

which can be simplified via rationalization, giving us:

$$y_n = \frac{x_n + x_{n-1}}{\sqrt{1 + x_n^2} + \sqrt{1 + x_{n-1}^2}}. \tag{14}$$

Notably, the denominator here will always be ≥ 1, ensuring that this form does not suffer from ill-conditioning[1]. This means we can safely replace every nonlinear activation in our network (assuming (11) was used during training) with this antialiased form without concern for undefined behavior.

Lastly, we note that the first-order ADAA introduces a delay of half a sample, which must be compensated for in the skip path via a two-point average operator, $y_n = (x_n + x_{n-1})/2$.

[1] To the best of our knowledge, Julian Parker was the first to identify this property [20] following the publication of [19]. It was later discovered independently by Martin Vicanek and documented in a recent self-published note [21].

5. EXPERIMENTAL SETUP

This section details the experiments performed, including a description of the examined devices, measurement signal and recording setup, as well as training process.

5.1. Modeled Devices

The proposed approach was tested using two guitar distortion circuits as case studies. The first circuit is a variant of the well-known second-order diode clipper, which has been the subject of numerous VA papers (see, e.g., [9] and [22]). Specifically, we used the version found in the MXR Distortion+[2] pedal, which features an op-amp driving circuit preceding the diode clipper stage. The second circuit modeled was the Ibanez Tube Screamer[3] overdrive pedal [23]. This circuit has two stages: an active non-inverting overdrive stage and a tone section.

Due to time and scope limitations, our experiments were constrained to the static parameter case, commonly known as a *profile*. Perspectives on incorporating parameter data into the proposed structure are provided in Sec. 7. For the diode clipper, we set the gain of the driving stage to minimum to bypass any potential clipping from the op-amp, instead driving the circuit via the level of the input signal. For the overdrive pedal, saturation was set to maximum while the tone control was set to 50%.

5.2. Measurement Signal

Given the musical nature of the examined circuits, the measurement signal consisted of a mixture of guitar and bass guitar recordings. Additionally, following the recommendations in [9], we included white noise band-limited to 22 kHz and a series of 10-second logarithmic sine sweeps with varying peak levels. Each sweep spanned from a minimum frequency of 20 Hz to a maximum frequency of 10 kHz. The total duration of the measurement signal was approximately 5 minutes and 30 seconds.

5.3. Measurement Setup

The two pedals were measured using an RME Fireface UCX II audio interface as follows. The measurement signal was sent to a stereo output on the interface. The left output channel was routed back into the interface via one of the inputs, while the right output was connected to the pedal. The pedal's output was then routed to a second input on the interface. This loopback configuration captures a reference copy of the input signal "as seen" by the audio circuit. This reference is essential, as it ensures that both input and output recordings are perfectly time-aligned, thereby accounting for any latency introduced by the measurement setup. Such configuration is standard practice in black-box modeling, as documented in [18]. All measurements were performed at a sample rate of 96 kHz. This sample rate was chosen as it provided a good level of aliasing suppression when combined with first-order ADAA, as will be shown in Sec. 6, thus removing the need to operate at a higher sample rate.

5.4. Network Initialization

During our experiments, we observed that the distribution of the elements of Λ tended to converge to values in the range $[0.8, 1.0)$, i.e., close to the unit circle. Therefore, following [10] we initialized our matrix Λ with values within this range by setting our trainable vector

$$\boldsymbol{\nu}^{\log} = \log\left(-\frac{1}{2}\left(\boldsymbol{\alpha}(1-0.8^2)+1\right)\right), \tag{15}$$

where $\boldsymbol{\alpha}$ is a vector sampled from a normal distribution. The normalization vector $\boldsymbol{\gamma}^{\log}$ was then initialized as

$$\boldsymbol{\gamma}^{\log} = \log\left(\sqrt{1-\left|\exp\left(-\exp(\boldsymbol{\nu}^{\log})\right)\right|}\right). \tag{16}$$

Matrices B, C, and D were initialized as described in Appendix A of [10] by simply ignoring the complex part.

5.5. Training Details

Models were trained in PyTorch for a total of 300 epochs using the AdamW optimizer with the default weight decay of 0.01. We employed a cyclic cosine annealing scheduler with a learning rate varying from a minimum of 10^{-4} to a maximum of 5×10^{-4}, where each cycle consisted of 100 epochs. A warmup phase of 20 epochs was also applied. To train the real LRU blocks, we used the open-source parallel scan implementation "mamba.py" [24]. Trainings were done on an Nvidia Tesla V100 Graphics Processing Unit (GPU), with all of them lasting under an hour each.

Two loss functions were used. The primary loss was the Error-to-Signal Ratio (ESR), which has been previously employed in black-box VA modeling applications [5]:

$$\mathrm{E}_{\mathrm{ESR}} = \frac{\sum_{n=0}^{L-1}\left(Y_n - \hat{Y}_n\right)^2}{\sum_{n=0}^{L-1} Y_n^2}, \tag{17}$$

where Y is the target signal and $\hat{Y}$ is the output of the proposed network, for a sequence of L samples.

A second frequency-domain loss was introduced to emphasize the frequency regions of interest. For this purpose, we employed a Mel-spectrogram-based loss, which converts both the target and predicted signals to the frequency domain using a Short-Time Fourier Transform (STFT) and maps the resulting frequencies to the Mel scale, which approximates human pitch perception. We used an implementation similar to that provided by Steinmetz and Reiss in the *auraloss* repository[4] [25].

The measured audio data was normalized to unit variance and split 80–20 for training and validation, respectively. The validation dataset consisted of guitar and bass guitar recordings exclusively. A dedicated third dataset for testing was not used since the models were also tested in real-time. Audio was segmented into sequences of 4096 samples, and a fixed batch size of 128 was used across all trainings. These two parameters were chosen to achieve a good balance between the number of batches per epoch, frequency resolution, and training time. At a sample rate of 96 kHz, the chosen sequence length (approx. 43 ms) can, at least in theory, capture frequency content from approximately 23 Hz. We experimented with longer sequence lengths and found that while the network could handle them, the reduction in the number of batches resulted in fewer parameter updates per epoch, which negatively affected the training outcome. Ultimately, we settled on these parameters empirically, as they provided a good balance on the number of batches

[2] `www.electrosmash.com/mxr-distortion-plus-analysis`
[3] `www.electrosmash.com/tube-screamer-analysis`
[4] `https://github.com/csteinmetz1/auraloss`

Device	N	H	D	Num. of Params	Train Loss	Val. Loss
DC	1	1	1	9	0.1422	0.0362
DC	2	2	1	24	0.0231	**0.0097**
DC	2	2	3	64	0.0114	**0.0095**
DC	4	4	1	72	0.0106	**0.0095**
DC	4	4	3	200	**0.0093**	**0.0083**
OD	8	4	1	112	0.1290	0.0789
OD	8	4	6	632	**0.0084**	**0.0040**
OD	12	6	1	228	0.1651	0.0857
OD	12	6	3	660	0.0116	**0.0061**
OD	16	8	1	384	0.1077	0.0714
OD	16	8	3	1120	0.0144	**0.0044**
OD	32	12	3	3024	**0.0082**	**0.0040**
OD	32	12	6	6024	**0.0056**	**0.0031**

Table 1: Total parameter count, and training and validation loss values for different network sizes trained on the diode clipper (DC) and overdrive (OD) data.

per epoch given the amount of data available. However, for modeling tasks involving more data (e.g. when parametric data is also available), we recommend increasing sequence length.

The LRU blocks in the proposed structure require a few samples of audio processing to "stabilize". This is normal behavior in recursive systems. In [5], for instance, the authors propose running their RNNs for 1000 samples prior to the start of the parameter updates. In our case, we found it sufficient to ignore the first 100 samples of each sequence before evaluating the loss functions.

6. RESULTS

In this section we present the outcome of our trainings. Additionally, we validate the antialiasing approach used and provide reference measurements regarding the time performance of the proposed model both during training and in a real-time inference scenario. Supporting materials, including, sound example can be found in the accompanying repository[5].

6.1. Training Results

Table 1 presents the results of various training experiments conducted on the two circuits. We explored different combinations of the network hyperparameters N, H and D to assess their impact on the total network parameter count and loss values. The last two columns on the table report the training and validation loss values, respectively, averaged over the final 10 epochs of each training. Note that for the validation loss, only the ESR loss is reported, which explains why these values are generally lower. We have highlighted loss values below < 0.01 as these correspond to a loss of less than 1% for the case of the validation loss (i.e. over 99% accuracy).

For the diode clipper circuit, these results indicate that networks with state sizes $N \geq 2$ perform well, which is consistent with expectations for a second-order structure. We observe how, as expected, increasing the network size results in lower loss values. In particular, the $4{\times}4{\times}3$ model (i.e. $N{\times}H{\times}D$) achieves both training and validation loss values below 0.01. This finding supports the notion that the learned state size can be (and usually is) significantly larger than the real state size. Informal listening tests further confirmed that all models, except for the baseline $1 \times 1 \times 1$ model, produced outputs that were perceptually very close to the reference data.

While the proposed structure successfully replicates the behavior of the target diode clipper circuit with relative ease, we acknowledge that the filtering effects of the specific unit we modeled may not be as pronounced as those of other diode clipper variants. During this work we did not trace or perform any kind of circuit analysis of the circuits under study. However, as a simple case study we believe these results to provide a useful baseline for our method.

Moving on to the overdrive circuit, the training results tell a different story. For this circuit, we were unable to generate accurate models for state sizes lower than $N = 8$, a significant difference w.r.t. the diode clipper. This result is supported by the knowledge that the overdrive pedal is larger and provides more filtering than the clipper, which results in a more dynamic behavior. Nevertheless, it is once again evident that the learned state size exceeds the assumed physical state size. A particularly interesting result is the fact that model $8 \times 4 \times 6$, with only 632 parameters, performed just as well as model $32 \times 12 \times 3$, which is nearly 5 times as large. This result highlights the expressive power contributed by the depth parameter, as described in [10]. As with the previous circuit, models with validation loss values < 0.01 were found to sound good during informal listening tests and real-time evaluations. These results serve as proof-of-concept that the proposed architecture is capable of learning the nonlinear dynamics of analog distortion circuits.

6.2. Antialiasing

Figure 3 shows spectrograms for a 10-second unit-gain linear sine sweep ranging from 20 to 20,kHz processed by the 8×4×6 overdrive model with and without first-order ADAA (14). As shown, the signal processed without ADAA suffers from high levels of aliasing distortion, with the aliases that fall below the fundamental frequencies being particularly problematic. In contrast, the bottom spectrogram demonstrates that the proposed method effectively reduces the overall level of aliasing. The remaining aliases could be further attenuated via higher-order ADAA. However, this is not explored further in this work.

We further examine the effect of ADAA by observing the magnitude spectra shown in Fig. 4. Here, we display the spectra of a 4186 kHz sine wave (the highest fundamental frequency on the piano) processed both with and without first-order ADAA. The direct implementation exhibits a significant concentration of audible aliasing below approximately 1 kHz, with one alias notably near the –20,dB mark. In contrast, the antialiased version shows a much cleaner spectrum, with most aliasing components falling below approximately –60,dB. During informal listening tests, we confirmed–using the sine sweep signal depicted in Fig. 3–that the proposed overdrive model does not exhibit perceivable aliasing up to fundamental frequencies of at least 5 kHz.

[5]`https://github.com/fabianesqueda/dafx25-va-modeling-lrus`

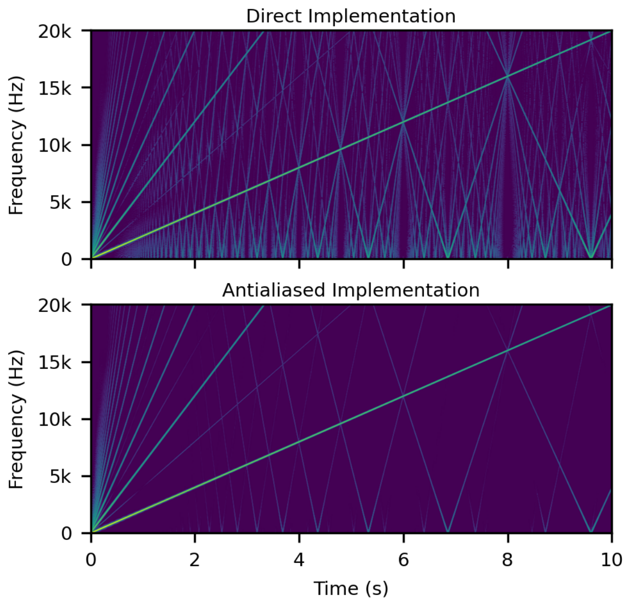

Figure 3: *Spectrogram for a linear sine sweep processed by the overdrive model (top) directly and (bottom) with first-order ADAA. Lowest visible amplitude is –80 dBFS.*

6.3. Time Evaluation

Lastly, we examine the time performance of the method during training inside PyTorch and during inference in a real-time C++ environment. For the first test, we measured the time it takes for different network sizes to implement a forward pass on a single batch of data. This provides a rough estimate of how the network hyperparameters N, H and D influence training times. All measurements were conducted under identical conditions, using a fixed batch size of 128. The results are presented in Table 2, with recorded times averaged over 10 executions.

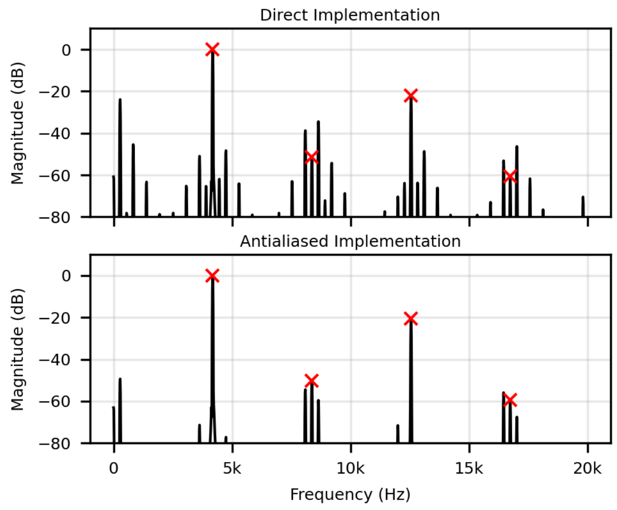

Figure 4: *Magnitude spectrum of a single 4186-Hz sinewave processed by the proposed* 8×4×6 *overdrive model implemented (top) directly and (bottom) using first-order ADAA. Red crosses indicate non-aliased components.*

N	H	D	Parameters	L	Time (ms)
32	12	1	1024	4096	6.70
32	12	3	3024	4096	15.9
64	12	1	1856	4096	6.10
64	12	3	5520	4096	15.0
64	12	6	11016	4096	29.4
64	12	6	11016	9600	36.1
64	12	6	11016	96000	45.7

Table 2: Average processing times during training for a single batch of data for different network sizes and sequence lengths.

These results show that the batch processing times for the proposed architecture scale linearly with the hidden size H and depth D, while changes in N appear to have minimal impact on performance when doubled. Next, we observe the effect of increasing sequence length to 100 ms (9600 samples) and 1 s (96000 samples). As shown by these measurements, increasing the sequence length by 234% only increases the batch processing time by 23%. Further increasing the sequence length by 10 times only increases processing time by approx. 27%. These results demonstrate how the LRU block benefits from the unrolling approach discussed in Sec. 2 and the use of the parallel scan function. In contrast, if the LRU block were to be trained sequentially, batch processing times would scale linearly with sequence length.

It should be noted that these numbers are provided for reference only, with the primary goal of illustrating how the performance of the proposed architecture scales with hyperparameters N, H, D and L. In practice, processing times will depend greatly on the hardware used and may also vary across different ML frameworks.

Lastly, the peformance of the proposed architecture was evaluated in a real-time scenario. The models (with ADAA) were implemented in C++ using the JUCE[6] framework, which allows us to build and deploy our models as VST plugins. The Eigen library was employed to implement the matrix-vector operations in a vectorized manner. All measurements were performed inside Ableton Live on a 2023 MacBook Pro with an Apple M2 Pro processor using JUCE's integrated "Performance Counter". During our initial experiments on real-time deployment we considered using an off-the-shelf ML inference engine such as ONNX Runtime [7]. However, we ultimately decided against this option and opted for a custom implementation due to real-time safety concerns [26].

Table 3 presents the processing times (in ms) for a single audio buffer for four different network sizes. Times were averaged over 100 executions. A fixed buffer size of 128 samples was used across all measurements. The last column shows the processing cost of the proposed networks relative to the available budget per second, which is a function of both the buffer size and the sample rate. As shown in this table, all models are highly efficient and suitable for real-time operation; even the largest model considered in this study (32×12×6) exhibits a relative processing usage of under 5%. It is worth highlighting that these figures only reflect audio processing costs and do not account for additional processes.

[6] `juce.com`
[7] `onnxruntime.ai/`

N	H	D	Parameters	Avg. Proc. Time (ms)	CPU Usage
8	4	6	632	0.012	0.9%
16	8	3	1120	0.018	1.35%
32	12	3	3024	0.030	2.26%
32	12	6	6024	0.059	4.40%

Table 3: Measured real-time processing times and CPU usage for a 128-sample buffer of audio for four network architectures.

7. CONCLUSION

In this work, we proposed the use of real LRUs to model audio distortion circuits. LRUs are an attractive choice for this task because, like most SSM models, they can be efficiently trained using parallelization techniques such as the parallel scan algorithm. Building upon the work of [10], we developed a neural network architecture composed of real LRU stages interleaved with streamlined nonlinear processing blocks. We showed that the proposed approach yields accurate models through two case studies involving two guitar distortion circuits. Moreover, the decoupling of the linear and nonlinear components in our structure naturally facilitates the integration of the ADAA technique. This approach helps reduce the oversampling requirements of the system and lowers computational overheads. The proposed method yields models that can be run in real-time inside modern digital production environments.

Regarding future work, an immediate open question is how to incorporate parameter data into the modeling structure. For example, [11] proposes using Feature-Wise Linear Modulation (FiLM) conditioning. Although we experimented with this approach to varying degrees of success, we found that FiLM conditioning substantially increased the hidden state and depth requirements of the network, leading us to question its overall efficacy. We hypothesize that integrating the parameter data directly into the LRU matrices could provide a more accurate representation of the circuit's internal behavior, where parameter changes directly affect the system matrices. However, due to the limited scope of this work, we leave this investigation for future research.

8. ACKNOWLEDGMENTS

We would like to acknowledge previous work done in collaboration with Julian Parker, Boris Kuznetsov, and Jason Naradowsky on a different (and as yet unpublished) research project that ultimately inspired the ideas presented in this work.

9. REFERENCES

[1] D. T. Yeh, J. S. Abel, and J. O. Smith, "Automated physical modeling of nonlinear audio circuits for real-time audio effects—Part I: Theoretical development," *IEEE Trans. Audio, Speech Lang. Process.*, vol. 18, no. 4, May 2010.

[2] M. Holters and U. Zölzer, "A generalized method for the derivation of non-linear state-space models from circuit schematics," in *Proc. European Signal Processing Conference (EUSIPCO-15)*, 2015.

[3] K. J. Werner, *Virtual analog modeling of audio circuitry using wave digital filters*, Ph.D. thesis, Stanford University, Stanford, CA, USA, Dec. 2016.

[4] 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. (ICASSP)*, Brighton, UK, May 2019.

[5] A. Wright, E.-P. Damskägg, and V. Välimäki, "Real-time black-box modelling with recurrent neural networks," in *Proc. 22nd Int. Conf. Digital Audio Effects (DAFx-17)*, Birmingham, UK, Sept. 2019.

[6] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proc. 30th Int. Conf. on Machine Learning (ICML-13)*, 2023.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. 31st Conf. on Neural Information Processing Systems (NIPS 2017)*, 2017.

[8] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," in *Proc. 10th Int. Conf. on Learning Representations (ICLR 2022)*, 2022.

[9] J. D. Parker, F. Esqueda, and A. Bergner, "Modelling of nonlinear state-space systems using a deep neural network," in *Proc. 22nd Int. Conf. Digital Audio Effects (DAFx-19)*, Birmingham, UK, 2019.

[10] A. Orvieto, S. L. Smith, A. Gu, A. Fernando, C. Gulcehre, R. Pascanu, and S. De, "Resurrecting recurrent neural networks for long sequences," in *Proc. 40th Int. Conf. on Machine Learning (ICML-23)*, 2023.

[11] R. Simionato and S. Fasciani, "Comparative study of state-based neural networks for virtual analog audio effects modeling," arXiv preprint arXiv:2405.04124, 2024.

[12] R. C. Dorf and R. H. Bishop, *Modern Control Systems*, Prentice Hall, 2001.

[13] Guy E. Blelloch, "Prefix sums and their applications," Tech. report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1990.

[14] E. Martin and C. Cundy, "Parallelizing linear recurrent neural nets over sequence length," in *Proc. 6th Int. Conf. on Learning Representations (ICLR 2018)*, 2018.

[15] B. O. Koopman and J. v. Neumann., "Dynamical systems of continuous spectra," in *Proc. of the National Academy of Sciences*, 1932.

[16] R. Diaz, C. de la Vega Martin, and M. Sandler, "Towards efficient modelling of string dynamics: A comparison of state space and Koopman based deep learning methods," in *Proc. 27th Int. Conf. Digital Audio Effects (DAFx-42)*, Guildford, UK, 2024.

[17] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," arXiv preprint arXiv:2312.00752, 2024.

[18] F. Eichas, S. Möller, and U. Zölzer", "Block-oriented gray box modeling of guitar amplifiers," in *Proc. 20th Int. Conf. Digital Audio Effects (DAFx-17)*, Edinburgh, UK, Sept. 2017.

[19] J. D. Parker, V. Zavalishin, and E. Le Bivic, "Reducing the aliasing of nonlinear waveshaping using continuous-time convolution," in *Proc. 19th Int. Conf. Digital Audio Effects (DAFx-16)*, Brno, Czech Republic, Sept. 2016.

[20] J. D. Parker, "Personal communication," Dec. 2024.

[21] M. Vicanek, "Note on alias suppression in digital distortion," [Online] `http://www.vicanek.de/articles/AADistortion.pdf`, Accessed April 2nd, 2025.

[22] J. Wilczek, A. Wright, V. Välimäki, and E. A. P. Habets, "Virtual analog modeling of distortion circuits using neural ordinary differential equations," in *Proc. 25th Int. Conf. Digital Audio Effects (DAFx-22)*, Vienna, Austria, 2022.

[23] D. T. Yeh, J. S. Abel, and J. O. Smith, "Simplified, physically-informed models of distortion and overdrive guitar effects pedals," in *Proc. 10th Int. Conf. Digital Audio Effects (DAFx-07)*, Bordeaux, France, 2007.

[24] A. Torres-Leguet, "mamba.py: A simple, hackable and efficient Mamba implementation in pure PyTorch and MLX.," [Online] `https://github.com/alxndrTL/mamba.py`, Accessed April 2nd, 2025.

[25] C. J. Steinmetz and J. D. Reiss, "auraloss: Audio focused loss functions in PyTorch," in *Digital Music Research Network One-day Workshop (DMRN+15)*, 2020.

[26] D. Stefani, S. Peroni, and L. Turchet, "A comparison of deep learning inference engines for embedded real-time audio classification," in *Proc. 25th Int. Conf. Digital Audio Effects (DAFx-22)*, Vienna, Austria, 2022.