UNIVERSITY OF ZAGREB

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 975

EMULATION OF GUITAR EFFECTS USING MACHINE LEARNING

Luka Ivanković

Zagreb, July 2025

UNIVERSITY OF ZAGREB

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 975

EMULATION OF GUITAR EFFECTS USING MACHINE LEARNING

Luka Ivanković

Zagreb, July 2025

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Zagreb, 03 March 2025

MASTER THESIS ASSIGNMENT No. 975

Student:	**Luka Ivanković (0036524531)**
Study:	Computing
Profile:	Computer Science
Mentor:	assoc. prof. Domagoj Vlah, PhD

Title: **Emulation of Guitar Effects Using Machine Learning**

Description:

In this thesis, it is necessary to investigate machine learning methods to implement a model capable of imitating guitar pedals and other sound effects. The fundamental idea is to train a model using data in which the input is the sound of a guitar without effects, while the desired output from the model is the sound of a guitar with an effect. The model should correctly learn the sound distortion produced by an individual pedal. Firstly, it is necessary to create a software framework for the easy creation of a new dataset for a particular pedal. Afterwards, several approaches to solving the problem should be tested. The first approach is to explore how deep networks (for example, the WaveNet architecture, RNN-based architectures, etc.) perform on this problem and to identify any potential shortcomings they might have. It is assumed that such deep models will exhibit certain limitations. The second approach would be to utilize a range of existing pedals (e.g., distortion, compression, reverb, delay, etc.) that depend on a number of parameters. The idea here is to approximate the original pedal (the one with which the dataset was created) by adjusting the parameters of these known pedals. Essentially, it is necessary to solve an optimization problem in the parameter space of all the pedals. Additionally, a decision must be made on which loss function to use for comparing sound signals. Since gradient-based methods will not be feasible for the optimization, the focus will be on approaches using genetic algorithms.

Submission date: 04 July 2025

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 3. ožujka 2025.

DIPLOMSKI ZADATAK br. 975

Pristupnik:	**Luka Ivanković (0036524531)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	izv. prof. dr. sc. Domagoj Vlah

Zadatak: **Emulacija gitarskih efekata primjenom strojnog učenja**

Opis zadatka:

U ovom radu potrebno je istražiti metode strojnog učenja za ostvarenje modela koji bi bio sposoban imitirati gitarske pedale i druge zvučne efekte. Temeljna ideja je trenirati model koristeći podatke koji su na ulazu zvuk gitare bez efekata, dok je željeni izlaz iz modela zvuk gitare s efektom. Model bi trebao ispravno naučiti izobličenje zvuka koje pojedina pedala ostvaruje. Prvo je potrebno napraviti softverski okvir za jednostavno kreiranje novog skupa podataka za pojedinu pedalu. Nakon toga je potrebno je isprobati više pristupa u rješavanju problema. Prvi pristup je istražiti kako na ovom problemu rade duboke mreže (recimo arhitektura WaveNet, arhitekture na bazi RNN-ova, itd.) te koje nedostatke one potencijalno imaju. Pretpostavlja se da će takvi duboki modeli imati određene nedostatke. Drugi pristup bi bio uzeti niz postojećih pedala (npr. distorzija, kompresija, reverb, delay...) koje ovise o nekom broju parametara. Ideja ovog pristupa bi bila originalnu pedalu (onu s kojom je stvoren skup podataka) aproksimirati podešavanjem tih parametara poznatih pedala. Esencijalno treba riješiti optimizacijski problem u prostoru parametara svih pedala. Treba odlučiti i koju funkciju gubitka koristiti za uspoređivanje zvučnih signala. Za optimizaciju neće biti moguće koristiti gradijente metode pa ćemo se fokusirati na pristupe pomoću genetičkih algoritama.

Rok za predaju rada: 4. srpnja 2025.

Thanks to all the giants, I did my best not to slip off the shoulders.

Contents

1 Introduction

Guitar players have used pedals for a long time for more expressiveness in their songs. Effect pedals offer multiple ways to alter audio signals and change guitar tone [1]. Traditionally, these effects are achieved through analog circuitry or digital signal processing (DSP), but designing them can be complex and time-consuming. Recently, machine learning has emerged as a promising alternative, offering the potential to emulate the sound and behavior of various guitar effects with high accuracy and less manual tuning. This thesis explores the use of machine learning techniques to model and emulate guitar effects, with the goal of achieving realistic results.

Modeling guitar effects is a technically challenging problem due to the nonlinear, dynamic, and time-dependent nature of audio transformations. Effects like distortion exhibit strong nonlinearity, while others like delay and reverb involve memory and temporal structure.

There are three ways to approach modeling guitar effects: black-box, white-box and gray-box modeling. Black box modeling assume nothing about the internal structure of guitar effects and focus on learning end-to-end mapping straight from input to output. In contrast, white-box modeling means that every analog component needs to be meticulously measured and digitally simulated, so that the model not only produces the sounds of the modeled effect, but also allows adjusting the controls like on the modeled effect. Gray-box modeling does something in between, it tries to assume some internal structure of the modeled effect, but still relies on optimization methods to fill out unknown parameters or relationships that cannot be directly observed or derived from first principles.

This thesis explores two of the three approaches to modeling guitar effects: black-box learning and gray-box modeling. For black-box models, we will first try out the basic fully connected network and start upgrading from there. We will introduce memory via LSTMs and later tackle more complex architectures like WaveNet, TCN, Structured state space sequence modeling. These models are highly expressive and flexible but can be quite data-hungry and memory expensive, that's why the second part of this thesis will focus on gray-box optimization, more specifically, we will try out the genetic algorithm and differentiable digital signal processing and compare them to white-box modeling.

Models will be evaluated on two types of effect pedals: distortion and reverb. We will evaluate a few different loss functions and answer why training these models on a dataset without guitar recordings may also work.

2 Dataset

Sound is a mechanical vibration that propagates as an acoustic wave through a medium such as air. These vibrations can be either heard by our ears which we then perceive as sound, or captured and represented digitally using audio file formats such as WAV. A standard sampling rate of 44.1 kHz means the audio signal is sampled 44,100 times per second. Why exactly 44.1 kHz is used explains Nyquist theorem, which states that to accurately reconstruct a signal, it must be sampled at least twice the highest frequency present. Since the upper limit of human hearing is approximately 20 kHz, a sampling rate just above 40 kHz is sufficient. Each sample represents the amplitude of the sound wave at a specific point in time.

The figure below 2.1 illustrates a typical guitar signal chain, where the raw input from the guitar passes through an effects pedal before reaching the output (such as an amplifier or audio interface).

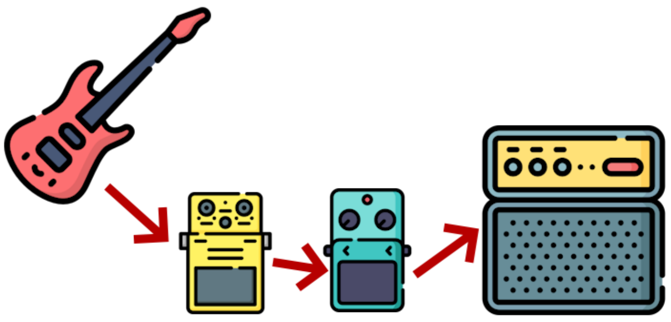

Figure 2.1: Typical guitar setup

In this case, all the signals are analog, even the amplifier. The figure below 2.2 illustrates how the same sound would be represented if captured digitally, as WAV files, before and after the pedal. These waveform transformations are what our models aim to capture and learn: modeling how the effects pedal alters the raw guitar signal in both subtle and significant ways.

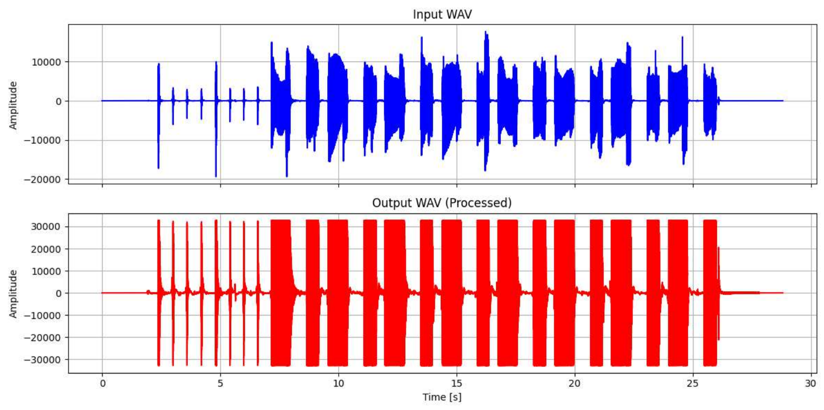

Figure 2.2: Audio waveform before (blue) and after (red) distortion

2.1 Dataset Source

The dataset used for this thesis is the "Musical Instruments Sound Dataset" by Soumendra Prasad, available on KaggleHub[1]. The original dataset contains 700 recordings of guitar, drums and violin each and 528 recordings of piano with various lengths. The test set contains a total of 80 audio files, 20 from each class.

Only guitar recordings were used to train the models. Audio files in the dataset are between 1.44 and 82 seconds long. As shown in Figure 2.3, the majority of the files are shorter than 5 seconds. I have converted all mono files to stereo, so every file now has 2 channels. All files use 44.1 kHz sampling rate.

Since the original dataset only contains unprocessed audio files, I have applied effects on all audio files using Spotify's library PedalBoard, thus creating pairs of unprocessed and processed audio files. In the next chapter we will look into details of which exact effects were chosen and why.

2.2 Effects

Two types of effects were used in this study: **distortion** and **reverb**.

- **Distortion** pedal alters an electric guitar's sound by adding gain and clipping the

[1] `https://www.kaggle.com/datasets/soumendraprasad/musical-instruments-sound-dataset`

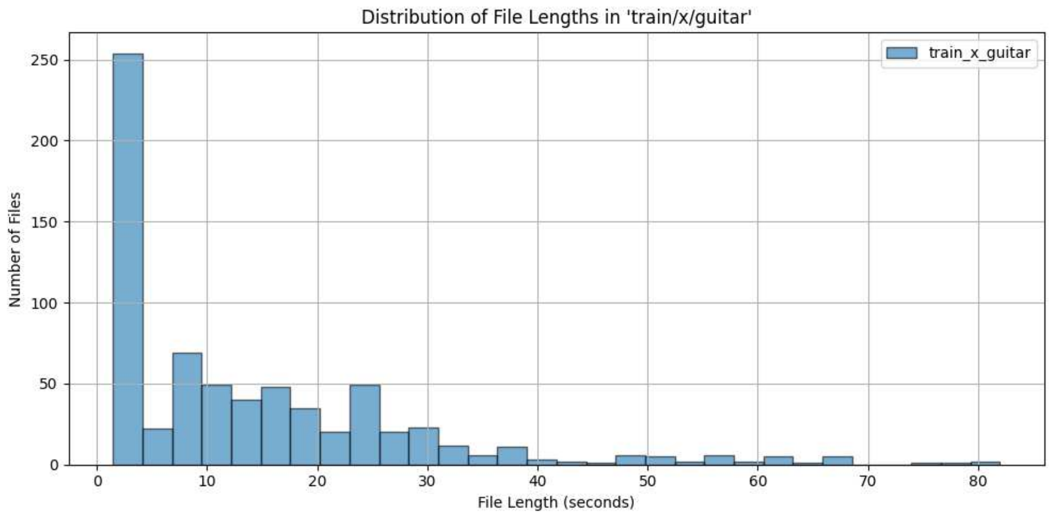

Figure 2.3: Distribution of file lengths

signal, resulting in a heavily overdriven, saturated tone, often described as "dirty" or "gritty". It's commonly used for rock and metal music to achieve a heavy, sustained sound.

- **Reverb** pedal simulates the natural sound reflections that occur in physical spaces, adding depth and ambience to a sound. It essentially creates the effect of playing in a room, hall, or other environment by emulating the way sound waves bounce off surfaces and decay over time.

Impulse Response Time

Impulse response time refers to the duration it takes for a system's output to settle down after being subjected to a brief, sudden input (an impulse). It will be useful to measure the impulse response time for guitar effects, as it correlates with how time-dependent the pedals are.

2.2.1 Distortion

Memoryless distortion refers to a type of distortion in a system where the output at any given time depends only on the input at that same time, without any influence from past input values. This means the system has no memory of past inputs and its behavior is solely determined by the instantaneous input. This system corresponds to an impulse response time of 0 samples.

For simple memoryless nonlinear distortions, one can model the transformation using static nonlinear functions. We can apply a simple **hard-clipping** distortion to an audio by multiplying the original signal by a constant g (that we call *gain*), and then applying the following nonlinearity:

$$y(t) = \begin{cases} -1 & \text{if } g \cdot x(t) < -1 \\ g \cdot x(t) & \text{if } -1 \leq g \cdot x(t) \leq 1 \\ 1 & \text{if } g \cdot x(t) > 1 \end{cases} \tag{2.1}$$

Simmilarily, we can model a **soft-clipping** distortion by applying $tanh$ nonlinear function after the gain:

$$y(t) = \tanh\left(g \cdot x(t)\right) \tag{2.2}$$

As shown in Figure 2.2, the waveform becomes visibly altered after applying soft-clipping with a high gain. The peaks of the waveform are clipped off and compressed toward the maximum and minimum amplitude limits, resulting in a more square-like appearance compared to the original signal.

In contrast to memoryless distortion, systems with memory can have output that depends on past input values. For example, a filter with a frequency response that is not flat (i.e., it affects different frequencies differently) has memory. These systems correspond to an impulse response longer than one sample.

Analog vs Digital Distortion

Analog distortion circuits rely on physical components like diodes, tubes, and transistors. These components are temperature-sensitive and may behave differently depending on humidity, power supply variation, and even the wear of the components. For instance, germanium transistors are known to sound different at different ambient temperatures.

While such small deviations are difficult to capture precisely, they generally do not significantly affect the perceptual quality.

Some pedals add non-deterministic elements, such as analog crackling or gray noise, that cannot be modeled accurately with deterministic ML systems. For example, some vintage analog fuzz pedals introduce intentional noise artifacts through unstable circuitry. Modeling such effects is theoretically impossible without modeling a stochastic source.

Distortion in my Dataset

In this dataset, we applied two different types of audio distortion:

- **Simple Distortion:**
  This is a deterministic, memoryless digital distortion implemented using *soft-clipping*, available in the Spotify `pedalboard` library. The transformation uses the soft-clipping nonlinearity defined in Equation 2.2, where g is the pre-gain applied to the input signal $x(t)$. For this dataset, we set the pre-gain to 25dB. This distortion is static and has no memory, making it relatively simple to model.

- **Raging Demon:**
  The second distortion type is the *Raging Demon*, a VST plugin[2] that we loaded using the `load_plugin` function available in the Spotify `pedalboard` library. To add extra harmonic content, we chained this plugin along with the same *soft-clipping* distortion from the Spotify `pedalboard` library used earlier, but with pre-gain set to 4dB this time. Unlike the soft-clipping distortion, the Raging Demon plugin introduces **nonlinear distortion with memory**, having an estimated impulse response duration of approximately **0.188 seconds**. This temporal behavior makes the modeling task significantly more complex, as the output depends on both the current and past inputs.

2.2.2 Reverb

Reverb is a time-domain effect that simulates the natural reflections of sound in a physical space like room or hall. Unlike distortion, reverb is often linear or quasi-linear and is typically implemented as a convolution of the input signal.

[2]`https://github.com/shayangheidi/theragingdemon/`

Reverb is a long-memory effect, and commonly has an impulse response time of more than a few seconds. Keeping in mind that each second of audio has 44100 samples, that means the model needs to have a receptive field of 100000 samples at the very least, only to capture long-term dependencies on 3 seconds of audio.

Analog Reverb

When digital modeling wasn't a thing, people were creative in using analog equipment to alter the sound. A good example is when they used actual springs to propagate sound. That resulted in a reverb-like effect, and it was later called spring reverb pedal and used among many guitar players. Its behavior is highly nonlinear due to the physical properties of the spring. To accurately model a spring reverb, a neural network would essentially need to approximate the behavior of a full mechanical physics engine, a task that is extremely difficult without access to the underlying physical state.

Reverb in My Dataset

In addition to distortion effects, the dataset also includes reverberation applied using VST plugins loaded with the Spotify `pedalboard` library. Both reverbs used in my dataset are from the Dragonfly plugin suite [3]

- **Dragonfly Plate Reverb:**
  Measured impulse response analysis showed that this effect introduces a reverberation tail lasting **2.844 seconds**. This adds a moderate amount of temporal dependencies for the models to learn.

- **Dragonfly Room Reverb:**
  This reverb has a significantly longer tail, with a measured impulse response duration of **6.738 seconds**. This adds an even greater amount of temporal dependencies, simulating a large and reflective room environment.

2.2.3 Other effects

There are also modulation effects like **chorus** and **phaser**, they are typically based on dynamic time-varying parameters (e.g., low-frequency oscillators). Similarly, **compres-**

[3] `https://michaelwillis.github.io/dragonfly-reverb/`

sion involves threshold-based dynamic range processing. These effects were chosen to be excluded from the research.

2.2.4 Applying the effects

After applying distortion and reverb effects to the original clean recordings, the final dataset contains one processed pair per effect type for each clean file. This results in four input-output pairs per clean file: one for each of the soft-clipping distortion, Raging Demon distortion, Plate reverb, and Room reverb effects.

Table 2.1 summarizes the audio effects used in this study, along with their estimated impulse response durations, sample lengths, and qualitative memory characteristics. These values reflect the temporal extent of each effect and were measured using Python by analyzing the decay envelope of each impulse response.

Table 2.1: Impulse Response Characteristics of Effects in the Dataset

Effect	**Estimated IR Duration**	**Samples**	**Memory Type**
Simple Distortion	0 ms	0	Memoryless
Raging Demon Distortion	0.188 s	8271	Short-Memory
Dragonfly Plate Reverb	2.844 s	125413	Long-Memory
Dragonfly Room Reverb	6.738 s	297165	Very Long-Memory

This table highlights the increasing complexity of modeling each effect. Memoryless effects such as soft-clipping are purely static and easier to approximate using feedforward models. In contrast, time-dependent effects like the Raging Demon and long-tail reverbs introduce significant temporal dependencies, requiring neural architectures with extended receptive fields and memory capabilities.

Figure 2.4 visualizes the total audio length per effect category. Effects with non-zero impulse responses, such as the reverbs and Raging Demon, result in longer output signals due to the natural decay or tail introduced by their temporal characteristics. Each processed audio file is extended by an amount equal to the effect's impulse response duration. This leads to a dataset that is significantly longer (by tens or hundreds of thousands of samples) compared to the original clean recordings.

To address the mismatch in length between clean inputs and processed outputs, the data loading pipeline automatically pads the shorter signal (usually the input) with zeros.

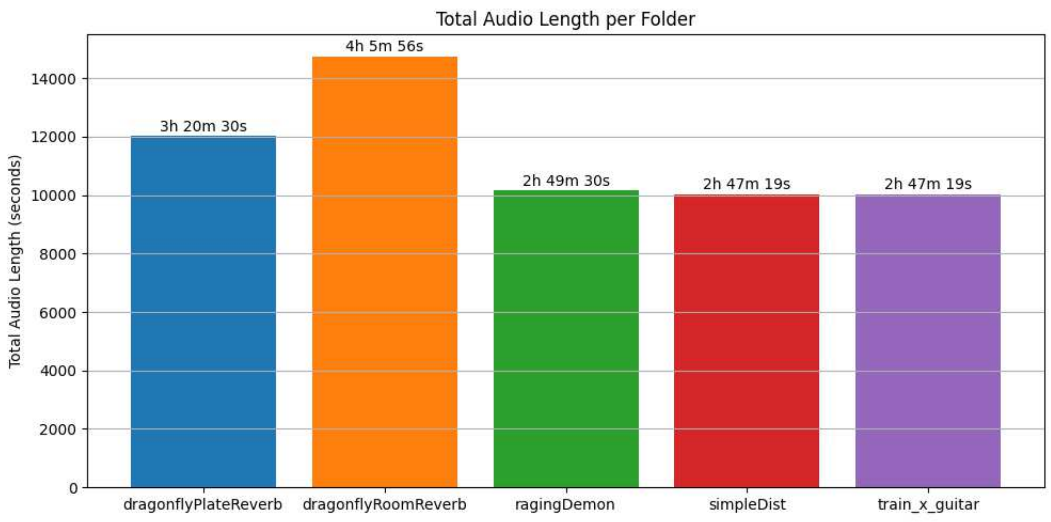

Figure 2.4: Distribution of dataset lengths

This ensures that all input-output pairs are temporally aligned and of consistent length for training. Notably, the purple bar (representing the soft-clipping distortion) appears nearly identical to the red bar (clean) in the figure, since this memoryless effect does not alter the length of the signal.

The following chapters will explore the architectural choices and training strategies used to capture these effects across varying levels of temporal complexity, from static nonlinearities to long-memory convolutional behavior.

2.3 Chunking and Smart Loading

To avoid bottleneck in loading new audio files from permenant memory to RAM, I implemented a custom dataset handler class using PyTorch's `Dataset` and `DataLoader` interfaces. The core strategy revolves around three key components: audio chunking, smart file access via a frequency-based cache, and normalization.

Chunking and Batching

Instead of loading all files at once into memory, we read files on-the-fly. A fixed-length sliding window of size N samples is then extracted from the audio files, acting as the input to the model. The target of the model is either the corresponding sample at the end of the chunk (for sample-wise prediction) or another chunk depending on the model architecture. This windowed approach allows training on longer sequences instead of training on individual samples.

Smart Caching via FrequencyQueue

To avoid reloading and reprocessing the same files repeatedly during training (especially when using small input windows that span across audio files), a smart caching mechanism is implemented using a custom `FrequencyQueue` class. This cache keeps track of the most frequently and recently accessed audio files and stores them in memory, up to a specified capacity.

Each file access updates both a frequency counter and a timestamp, ensuring that files which are both frequently and recently accessed are prioritized in memory. When the cache exceeds its capacity, the least recently/frequently used file is evicted.

The benefits become especially apparent when batch shuffling is enabled: since random chunks are sampled from the dataset, larger audio files (containing more samples) are naturally selected more often. Without an intelligent caching mechanism, this would result in frequent reloading of large files from disk, introducing significant I/O overhead. By employing the `FrequencyQueue`, we avoid redundant disk access by keeping the most frequently and recently accessed files in memory, greatly improving data loading speed and overall training throughput.

Normalization

After each file is loaded into memory, it is then normalized to ensure consistent dynamic range across the dataset. The normalization function scales each audio file so that its maximum absolute value (across both stereo channels) is within the range $[-1, 1]$. This speeds up convergence and stabilizes training. The model will focus more on modeling the input-output mappings rather than also trying to predict the scale or loudness variations between different recordings. Without normalization, amplitude differences between files could introduce unwanted bias, forcing the model to allocate capacity to learn trivial gain corrections instead of capturing the actual effect behavior.

To further improve the dataloader speed, normalization should have been done as part of the data preprocessing, allowing the model to load already-normalized audio files directly from disk, thereby reducing per-sample processing overhead during training.

2.4 Loss Function Selection

In audio effect modeling, the choice of loss function is critical. The suitability of a loss function depends primarily on the *type of effect* being emulated, rather than on *type of model* used.

MSE vs. MAE vs. ESR

The *Mean Squared Error* (MSE) is defined as:

$$\mathrm{MSE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_t (y_t - \hat{y}_t)^2, \tag{2.3}$$

penalizing absolute sample-wise amplitude differences without normalization.

The *Mean Absolute Error* (MAE), also known as the L_1 loss, is defined as:

$$\mathrm{MAE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_t |y_t - \hat{y}_t|, \tag{2.4}$$

penalizing absolute deviations linearly, making it less sensitive to large outliers than MSE.

In contrast, the *Error-to-Signal Ratio* (ESR) loss is defined as:

$$\mathrm{ESR}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\sum_t (y_t - \hat{y}_t)^2}{\sum_t y_t^2 + \varepsilon}, \tag{2.5}$$

where $\mathbf{y}$ is the target signal, $\hat{\mathbf{y}}$ the predicted signal, and ε is a small constant to avoid division by zero.

ESR can also be applied *after* a pre-emphasis filter:

$$x'_t = x_t - \alpha x_{t-1}, \quad 0.9 \le \alpha \le 1.0, \tag{2.6}$$

to boost high-frequency components and make the loss more sensitive to perceptually important details such as early reflections in reverberation.

Choosing Loss Functions for Different Audio Effects

The choice of loss function should reflect the perceptual and physical properties that define the audio effect [2] [3]. For distortion-based effects such as overdrive or fuzz, exact waveform amplitude and phase alignment is crucial, since even small deviations in amplitude can substantially alter the harmonic content. In these cases, the Mean Squared Error (MSE) is generally the most appropriate choice, as it penalizes absolute sample-wise differences without normalization.

For time-based effects like reverb, delay, or chorus, perceptual similarity is more important than precise waveform matching. These effects often produce dense, phase-rich outputs where many different waveforms can sound perceptually identical. Here, a scale-invariant measure such as the Error-to-Signal Ratio (ESR) is better suited, especially when combined with pre-emphasis filtering to highlight high-frequency details that carry spatial cues (e.g., early reflections). Alternatively, a spectral-domain loss can capture similarities in spectral envelope and decay characteristics without requiring exact phase matching.

In the case of equalization (EQ) and other filtering effects, the defining property is the accuracy of the frequency response. Since these effects are explicitly designed to shape magnitude spectra, an **STFT-magnitude MSE** directly optimizes for matching the target spectral profile while ignoring irrelevant phase differences.

Finally, for pitch-shifting and time-stretching effects, the perceived pitch and timing must be preserved, even if the waveform shape changes. In such cases, **perceptual spectral losses** [4], which compare representations such as mel-spectrograms, can better reflect human auditory sensitivity to pitch accuracy and temporal coherence.

2.4.1 Empirical Findings

When modeling a simple audio distortion effect with an LSTM, I empirically found that using **MSE** yielded better results than ESR. Because ESR is scale-invariant, it can underestimate the penalty for small amplitude mismatches, which are perceptually critical in distortion. In contrast, for effects such as reverb, ESR is generally expected to be more appropriate. In the rest of this work we will use both MSE and ESR.

3 Black-Box models

3.1 Fully Connected Networks

Fully Connected Networks (FCNs) are among the simplest types of neural architectures. Each neuron in a given layer is connected to all neurons in the subsequent layer. When applied to audio modeling, an FCN typically operates on short windows of the input signal.

FCNs can effectively model simple distortion effects with minimal memory, i.e., with a very short or negligible impulse response. This is because such effects are primarily nonlinear amplitude transformations, often describable by a static nonlinear function:

$$y[n] = f(x[n])$$

where $x[n]$ is the input signal and $f(\cdot)$ is the nonlinear transformation applied by the effect.

However, FCNs struggle with effects that involve longer memory, such as reverb or delay. These effects require a longer temporal context, and since FCNs do not inherently model time dependencies, they fail to capture these interactions.

In my experiments, I used a fully connected network with a single hidden layer of size 72 and ReLU activation function. After training for 5 epochs, it successfully learned to emulate only the simplest soft-clipping distortion, characterized by an impulse response length of zero samples. In this case, the network effectively learned the nonlinear transformation of the distortion, which corresponds to a simple tanh function.

Interestingly enough, when the activation function of the FCN is chosen to be sigmoid instead of ReLU, the network can directly model the smooth saturation curve of the distortion. This is because the hyperbolic tangent can be expressed in terms of the logistic sigmoid $\sigma(z)$ as:

$$\tanh(gx) = 2 \cdot \sigma(2gx) - 1,$$

where the gain parameter g is learned through the weights of the network. In this form, $\sigma(2gx)$ produces a smooth transition between 0 and 1, which is then scaled and shifted to match the -1 to 1 range of tanh.

Using `ReLU` activation makes the task significantly more challenging, as `ReLU` produces only piecewise-linear mappings. While a ReLU-based FCN can approximate $\tanh(gx)$ to arbitrary precision by increasing the number of hidden neurons or layers, the result will always be a segmented linear approximation rather than a smooth curve, requiring more parameters to achieve the same accuracy as smooth activations.

For the reverb task, the model ended up learning only the dry component of the signal, effectively minimizing the MSE by ignoring the reverberant tail. As a result, it failed to reproduce the characteristic decay and spatial texture of the reverb effect.

Table 3.1 demonstrates the limitations of FCNs when modeling effects with increasing temporal complexity.

Table 3.1: Performance of FCNs on Effects with Increasing Impulse Response

Effect Type	**Impulse Response Length**	**MSE Loss**	**Subjective Quality**
Simple Distortion	None	0.00095	Good
Raging Demon Distortion	Short	0.028	Moderate
Dragonfly Plate Reverb	Long	0.3537	Poor

3.2 Recurrent Networks (LSTM)

Recurrent Neural Networks (RNNs) [5] are designed to model sequential data by maintaining a hidden state that captures past information. Major improvements like the Long Short-Term Memory (LSTM) 3.1 architecture proposed by Hochreiter and Schmidhuber [6] enabled RNNs to effectively learn longer temporal dependencies by mitigating the vanishing and exploding gradient problems.

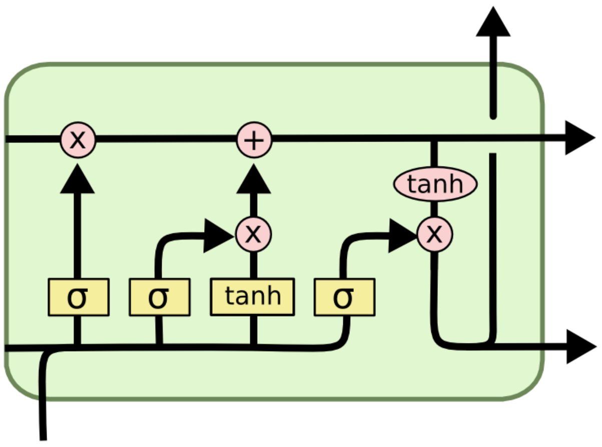

Figure 3.1: Representation of an LSTM cell

Because of their memory and gating mechanisms, LSTMs have been implemented to emulate a variety of distortion-type effects and even tube amplifiers [7] [8]. Unlike FCNs, RNN-based models can, in principle, model effects with theoretically infinite impulse responses, although in practice they often struggle to do so.

I used similar architecture as [7] and the Keith Bloemer's implementation available at [1]. As illustrated in Figure 3.2, the model first applies two 1D convolutional layers with 16 filters each to capture short-term temporal features from the input audio, followed by a single LSTM layer with 26 hidden units to model longer temporal dependencies. Input sequences of length 100 are processed in a sliding-window fashion, and the LSTM output at the last time step is passed through a fully connected layer to produce the predicted audio samples. The model was trained to minimize sample-wise MSE, aiming to learn both the nonlinear characteristics and the short-term temporal structure of the effect.

As shown in Table 3.2, the LSTM matches the FCN in performance for simple distortion, but performs significantly better on distortions with short impulse responses, such as Raging Demon Distortion. However, it still fails to fully capture the effect, since the impulse response is 8271 samples long, while LSTMs based on NLP tasks have been shown to handle sequences of roughly 500–1000 tokens. This corresponds to only about 10–100 ms (441–4410 samples) in audio modeling, which is shorter than the impulse response length.

[1]`https://github.com/GuitarML/GuitarLSTM`

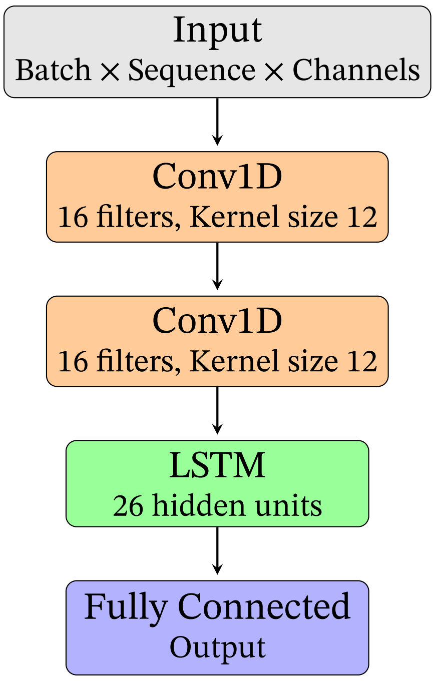
Input
Batch × Sequence × Channels
Conv1D
16 filters, Kernel size 12
Conv1D
16 filters, Kernel size 12
LSTM
26 hidden units
Fully Connected
Output

Figure 3.2: Architecture of the LSTM-based guitar effect model.

LSTM share the same problem with FCN in modeling reverb, it struggles to emulate it effectively and focuses on minimizing MSE by capturing only the dry part of the effect.

Table 3.2: Comparison of LSTM and FCN on Effects with Increasing Impulse Response

Effect Type	**Model**	**MSE Loss**	**Subjective Quality**
Simple Distortion	FCN	0.00095	Good
Simple Distortion	LSTM	0.00043	Great
Raging Demon Distortion	FCN	0.028	Moderate
Raging Demon Distortion	LSTM	0.0013	Good
Dragonfly Plate Reverb	FCN	0.3537	Poor
Dragonfly Plate Reverb	LSTM	0.0143	Poor

3.3 WaveNet

WaveNet, introduced by van den Oord et al. in 2016 [9], is a generative model for raw audio based on dilated causal convolutions 3.3. The model stacks layers of convolutions where the dilation factor increases exponentially, allowing the receptive field to grow without an explosion in parameters.

WaveNet is well-suited for short-term, nonlinear effects like distortion, where the output depends heavily on a small window of recent samples. However, reverb is a long-term effect with a very long impulse response, sometimes lasting several seconds, mean-

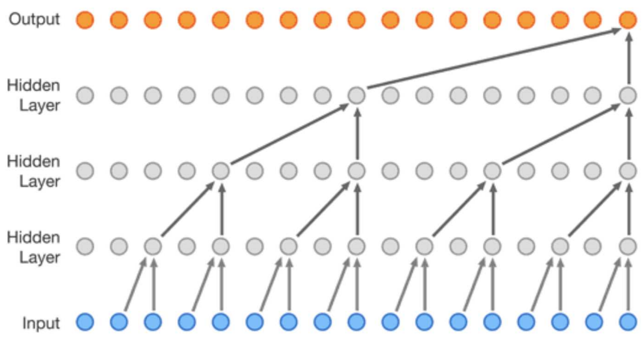

Figure 3.3: Wavenet dilated convolutions

ing the current output may depend on audio that occurred thousands or even millions of samples earlier. Because WaveNet has a finite receptive field (determined by its number of layers and dilation rates), it cannot capture such long-term dependencies effectively.

My implementation roughly followed the approach of Keith Bloemer[2] and the architectures proposed in [7, 8]. Concretely, a PyTorch WaveNet model was configured with stereo input (as opposed to Keith Bloemer's implementation that was mono), 32 residual channels, 64 skip channels, 3 blocks, 4 layers per block and kernel size of 2. Each block consists of a causal convolution followed by gated activations, with residual and skip connections aggregated across layers.

During training, sequences were fed in chunks using sliding windows, similar to the LSTM approach, and the model was optimized using sample-wise MSE loss.

Wavenet performs similarly to LSTMs on audio distortion emulation task as already shown in [8], and shares some of LSTMs limitations (struggles with effects that have longer temporal dependencies such as reverb), but also offers some upgrades such as faster convergence due to parallelization. Although, RNNs are shown to require less processing power to run [8]. In Table 3.3, we can see that Both LSTMs and Wavenet fail miserably for the longest room reverb effect. They manage to simulate only a very short part of the reverb tail (that is more than 6 seconds for the longest reverb).

[2]`https://github.com/GuitarML/PedalNetRT`

Table 3.3: Wavenet and LSTM struggling on Effects with very long impulse response

Effect Type	Model	MSE Loss	Subjective Quality
Dragonfly Plate Reverb	Wavenet	0.0178	Poor
Dragonfly Plate Reverb	LSTM	0.0143	Poor
Dragonfly Room Reverb	Wavenet	0.0933	Poor
Dragonfly Room Reverb	LSTM	0.0941	Poor

3.4 Temporal Convolutional Network (TCN)

Temporal Convolutional Networks [10] build upon WaveNet's idea but introduce residual blocks and normalization. TCNs are causal, use dilated convolutions, and can model long sequences efficiently. They have the advantage of parallel computation (unlike RNNs) and avoid the vanishing gradient issues common in recurrent architectures.

Empirically, TCNs often outperform LSTMs [11] in both convergence speed and final accuracy when modeling effects with long memory, while still retaining fast inference capabilities. They also outperform WaveNet in this task, because they do not limit receptive field to a few seconds like WaveNet does. They have been successfully applied to model both dynamic range effects such as compression [12] and nonlinear effects such as overdrive [13].

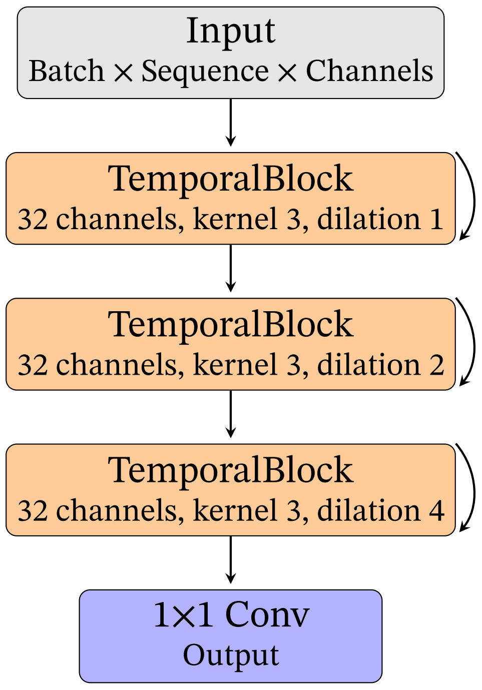
Input
Batch × Sequence × Channels
TemporalBlock
32 channels, kernel 3, dilation 1
TemporalBlock
32 channels, kernel 3, dilation 2
TemporalBlock
32 channels, kernel 3, dilation 4
1×1 Conv
Output

Figure 3.4: Architecture of the Temporal Convolutional Network (TCN) model.

As shown in Figure 3.4, the architecture consists of three `TemporalBlock` modules, each containing two 1D convolutional layers with ReLU activations, dropout, and causal padding removal via a `Chomp1d` operation. Residual connections facilitate gradient flow through the deep stack of blocks, while a 1×1 convolution is used for channel matching when required. The blocks employ dilation factors of 1, 2, and 4, each with 32 channels and a kernel size of 3. A final 1×1 convolution projects the output back to a single channel.

Training followed the same sliding-window batching approach used for the WaveNet, with sample-wise mean squared error (MSE) as the loss function. Empirically, the TCN converged faster than both LSTMs and WaveNet, and achieved lower loss on reverb tasks with long impulse responses (see Table 3.4).

Table 3.4: Performance of TCN vs. WaveNet and LSTM on long-memory reverb effects.

Effect Type	**Model**	**MSE Loss**	**ESR**
Dragonfly Plate Reverb	TCN	**0.0115**	1.607
Dragonfly Plate Reverb	Wavenet	0.0178	11.745
Dragonfly Plate Reverb	LSTM	0.0143	14.01
Dragonfly Room Reverb	TCN	**0.0452**	1.077
Dragonfly Room Reverb	Wavenet	0.0933	26.32
Dragonfly Room Reverb	LSTM	0.0941	33.18

3.5 Structured State Space Models (Mamba)

Structured State Space Sequence models [14], provide a flexible framework for general sequence modeling and have been shown to surpass traditional recurrent, convolutional, and Transformer-based architectures across a range of tasks. Each S4 layer can be interpreted as a differentiable realization of an infinite impulse response (IIR) system in state-space form, allowing it to maintain information over theoretically unlimited time horizons, much like recurrent networks. This property makes S4 particularly well-suited for modeling non-linear, time-dependent audio effects, such as those found in analog guitar pedals [15, 11, 16]. NablaFx [17] offers open source implementation of S4 for audio effects modeling tasks [3]

Mamba [18] is a recent implementation of SSMs designed for efficiency and ease of

[3] `https://github.com/mcomunita/nablafx`

integration in audio and control tasks. It leverages the mathematical structure of SSMs to reduce computation while preserving long-term memory, and it forms the backbone for the S4 architecture.

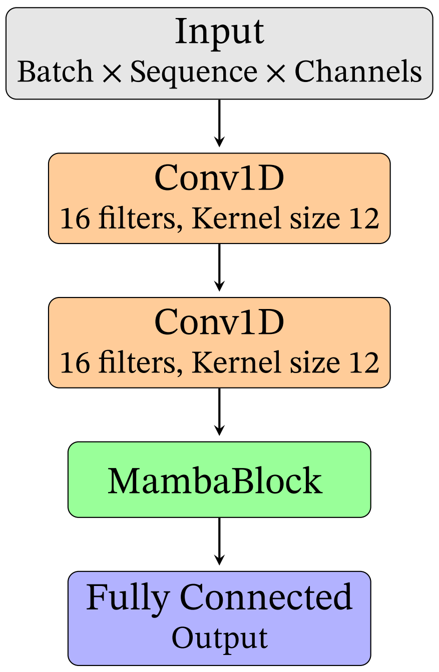
Input
Batch × Sequence × Channels
Conv1D
16 filters, Kernel size 12
Conv1D
16 filters, Kernel size 12
MambaBlock
Fully Connected
Output

Figure 3.5: Architecture of the Conv-Mamba guitar effect model.

In my experiments, I implement a Conv-Mamba architecture. As illustrated in Figure 3.5, the model consists of two 1D convolutional layers, each with kernel size 12 and 16 filters. The output features are passed through a single Mamba block, which expands the input dimension to 26, applies a placeholder state-space kernel, and projects back to the hidden dimension. A fully connected layer then maps the output of the last time step to two output channels. The model operates on input tensors of shape [batch_size, sequence_length, 2] and produces output tensors of shape [batch_size, 2], similar to my LSTM implementation.

Empirically, the Conv-Mamba model achieves lower mean squared error (MSE) than both WaveNet and TCN on tasks involving long-memory effects, such as plate and room reverb. Table 3.5 summarizes the results, showing that the model is able to capture more of the reverb tail and produce higher subjective audio quality.

Table 3.5: Performance of Conv-Mamba on long-memory reverb effects compared to TCN and WaveNet.

Effect Type	**Model**	**MSE Loss**	**ESR**
Dragonfly Plate Reverb	Conv-Mamba	**0.0089**	0.85
Dragonfly Plate Reverb	TCN	0.0115	1.607
Dragonfly Plate Reverb	Wavenet	0.0178	11.745
Dragonfly Room Reverb	Conv-Mamba	**0.0312**	0.62
Dragonfly Room Reverb	TCN	0.0452	1.077
Dragonfly Room Reverb	Wavenet	0.0933	26.32

3.6 Transformers

Transformers, introduced in 2017 [19], revolutionized NLP by modeling global context through self-attention, and have recently been implemented for a variety of audio processing tasks [20, 21].

While powerful, Transformers are computationally expensive and require large datasets for effective training. Their lack of inductive bias for time locality makes them less efficient for audio applications like guitar effects emulation. To my knowledge, they have not yet been implemented to emulate guitar effects.

Nevertheless, their global receptive field and flexibility make them a potential option for future work.

4 Gray-Box models

Gray-box models assume an existing digital signal processing chain and focus on tuning its parameters to match the behavior of a target effect. This model-based parameter optimization introduces a strong inductive bias, significantly reducing the space of possible solutions. While less flexible than black-box models, gray-box approaches are more interpretable, faster to train, and often sufficient for approximating a wide range of guitar effects.

I present two methods:

- A model based on a Genetic Algorithm (GA) that searches for optimal combinations of audio effects and parameters to match a reference signal.
- A differentiable DSP based model that learns interpretable parameters via standard gradient optimization methods.

4.1 Genetic Algorithm-Based Effect Estimation

Genetic algorithm-based audio modeling has been explored in several studies [22, 23]. Other non-differentiable optimization methods have been used for audio synthesis, such as in [24], where Gaussian Processes were employed for active learning of intuitive control knobs in synthesizers. However, to my knowledge, there have been no studies applying genetic algorithms specifically to guitar effect emulation. First we will go over a brief explanation of what genetic algorithm is, and then focus on my implementation in terms of guitar effect modeling.

4.1.1 Overview of Genetic Algorithms

Genetic Algorithms are optimization techniques inspired by biological evolution. A GA operates on a population of candidate solutions, which evolve over successive generations through:

- **Selection**: Favoring individuals with higher fitness (better solutions).
- **Crossover (Mating)**: Combining two parents to produce a new individual.
- **Mutation**: Introducing small random changes to maintain diversity.
- **Elitism**: Preserving a few top-performing individuals into the next generation.

In each generation, individuals are evaluated using a *fitness function*, and the best solutions are carried forward while new candidates are generated through crossover and mutation. This process continues until convergence or a stopping criterion is reached.

4.1.2 Effect Chain Encoding and Pedalboard Integration

We use the `pedalboard` library by Spotify to simulate common audio effects: `Compressor`, `Distortion`, `Chorus`, `Reverb`, and `Delay`. Each individual in the GA population represents a fixed sequence of these effects, with tunable parameters as shown in Figure 4.1. All individuals have exactly 5 of these effects.

Each effect is encoded with:

- A dictionary of effect-specific parameters (e.g., threshold, ratio, damping).
- A `dry/wet mix` coefficient $\alpha \in [0, 1]$, which balances unprocessed and processed audio:

$$\text{output} = (1 - \alpha) \cdot \text{dry} + \alpha \cdot \text{wet}$$

Including the dry/wet mix greatly improves flexibility, allowing the algorithm to use subtle versions of effects.

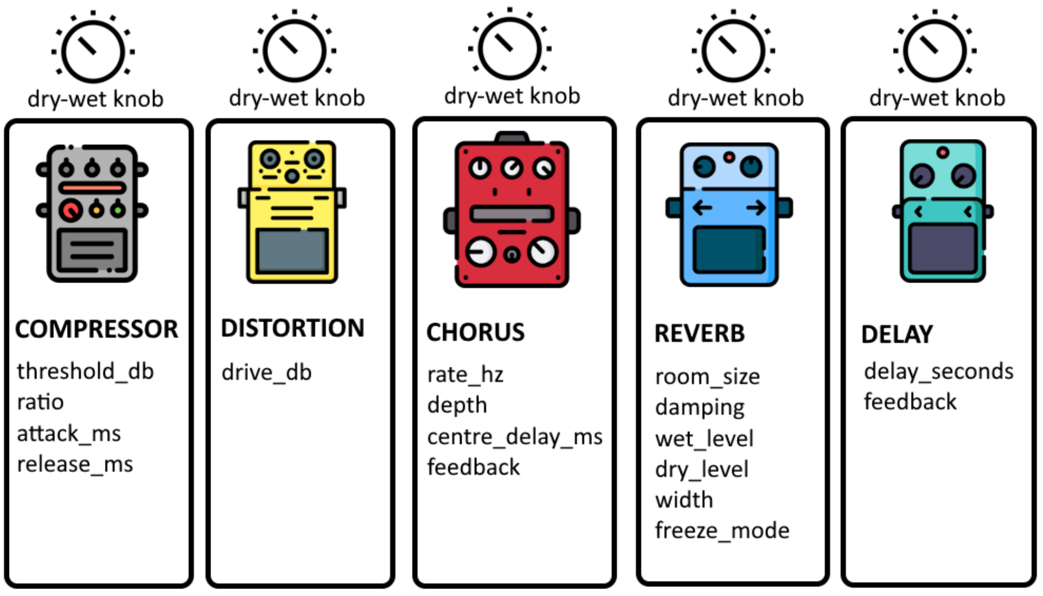

Figure 4.1: A signal chain containing effects and it's parameters. Along effect-specific parameters, each effect also contains a dry-wet mix parameter

4.1.3 Fitness Evaluation and Evolutionary Process

Each candidate effect chain is evaluated by comparing the audio it produces (by processing dry input) to a given target audio. This is done chunk-by-chunk (typically 1-second segments), and the loss is computed as the mean squared error. Fitness function is thus inversely proportional to the loss function.

The population evolves according to the following steps:

1. **Selection**: Tournament selection is used to choose parents. A subset of individuals compete from the old population (typically 4), and the best one is selected with high probability. Once both parents are chosen via this process, they mate and the offspring is added to the new population. The whole process repeats until the new population is as long as the old one.

2. **Crossover**: Each child inherits one effect from either parent at each position in the chain.

3. **Mutation**: Each effect has a chance to mutate. This may modify a parameter slightly or adjust the dry/wet mix.

4. **Elitism**: Best performing individuals (by fitness) are preserved directly into the

next generation.

The genetic algorithm described in Algorithm 1 runs iteratively, logging the average loss and exporting the best solution's audio output after each generation for monitoring. In my experiment, I've set the number of individuals in the population to $N = 50$, the tournament size to $T = 4$, and the elitism rate to $E = 0.02$, and let the algorithm run for 30 generations.

Algorithm 1 Genetic Algorithm Implementation

```
1:  Initialize population of $N$ individuals with random effect chains
2:  for each individual in population do
3:      Evaluate fitness by processing dry audio and computing error to target audio
4:  end for
5:  while stopping criterion not met do
6:      Sort population by fitness (lower error = higher fitness)
7:      Preserve top $E \cdot N$ individuals as elites
8:      Initialize new population with elites
9:      while new population size $< N$ do
10:         Select two parents using tournament selection of size $T$
11:         Produce child: for each effect, randomly select from one parent
12:         Mutate child: for each effect, slightly perturb a parameter or dry/wet mix
13:         Evaluate child fitness
14:         Add child to new population
15:     end while
16:     Replace old population with new population
17:     Output an effected wav file of the best individual
18:     Output MSE of the best individual
19: end while
```

4.1.4 Results and Discussion

While this solution offers interpretability, it still fails to outperform even the basic black-box model like LSTM, and convergence is very slow due to stochastic nature of genetic algorithm. Why it fails to outperform black-box models lies in a fact that we introduced a strong inductive bias and thus limited expressivity of a model. This model can't learn the task completely unless the targeted pedal is in fact a combination of effects used in the algorithm.

4.2 Gradient-Based Optimization (DASP)

Building on the idea from the previous chapter - an effect chain designed to emulate a target guitar pedal - we now replace the GA-based parameter search with standard gradient-based optimization methods such as gradient descent. This change greatly accelerates convergence, as the model no longer relies on purely stochastic exploration of the parameter space.

While the spotify Pedalboard library used in previous chapter offers common audio effects and allows to define signal chains, these are not differentiable. Differentiable Audio Signal Processing (DASP)[1] provides a differentiable implementation of common guitar effects inspired by [25, 26, 27, 28].

Using `DASP`, I represented a sequence of effects - distortion, compression, and reverb, with each effect having differentiable parametrs. By minimizing a mean squared error (MSE) between the chain's output and a target audio signal, the chain learns interpretable parameters that approximate the desired tone.

DASP learns faster than genetic algorithms because its differentiable blocks lead it directly toward a solution, rather than stochastically wandering through the parameter space like GA. Its parameter space is the same as the genetic model (not exactly the same, but very similar, since the pedal implementations inside the effect chain are not identical), which means it is still limited in expressivity, but it is good enough for approximating some pedals.

4.2.1 Limitations of Gray-Box Models

Both the GA-based approach and the DASP-based approach share a fundamental limitation: they can only converge to target pedals that lie within the expressivity of the model, i.e., the space spanned by the effects included in the chain. If a target pedal implements

[1]`https://github.com/csteinmetz1/dasp-pytorch`

Table 4.1: Performance comparison of Gray-Box (GA, DASP) vs. Black-Box (WaveNet, LSTM) models on pedal emulation.

Effect Type	**Model**	**MSE Loss**
Simple Distortion	GA	0.0852
Simple Distortion	DASP	0.0714
Simple Distortion	LSTM	0.00043
Simple Distortion	WaveNet	0.00028
Plate Reverb	GA	0.123
Plate Reverb	DASP	0.109
Plate Reverb	LSTM	0.0143
Plate Reverb	WaveNet	0.0112

a non-linear behavior, unique modulation, or interaction between parameters that cannot be represented as a linear or differentiable combination of the available effects, then both models will fail to fully replicate it.

It can be observed from Table 4.1 that both gray-box models perform worse than the black-box models, due to their lack of expressivity.

Recent studies, such as [17], have suggested combining gray-box and black-box optimization to address this limitation. In such hybrid approaches, a differentiable chain captures the bulk of the effect behavior, while a black-box neural network component models the residual non-linearities or dynamics that the differentiable chain cannot represent. This allows the model to retain interpretability in the primary effect parameters while gaining the expressivity needed to approximate more complex pedals. This hybrid approach offers a promising direction for future research.

5 Additional experiments

5.1 Modeling knobs

In this thesis, the pedals I attempted to emulate had "locked-in" knob settings. Unlike real pedals, where adjusting knobs in real time can create a wide variety of tones, the pedals in this study were fixed. Modeling the control knobs was not the focus in my research.

In many differentiable audio modeling frameworks, conditioning mechanisms such as FiLM and TFiLM [17] are used to allow neural networks to dynamically adapt to different knob settings. These layers modulate the network's internal activations based on external parameters (e.g., knob positions), enabling the model to produce a continuous range of outputs corresponding to different pedal settings. Conditioning improves interpretability and flexibility of a model.

For physical analog devices, capturing knob-dependent behavior requires synchronizing control knob positions with recordings of both dry input and wet output audio. Manually collecting such data is challenging, as exhaustively sampling the continuous control space of an amplifier or effects pedal is time-consuming and prone to inconsistency. Recent work [29] has addressed this issue through robotic automation. Robotic systems can accurately and consistently adjust knobs while recording paired input-output data.

5.2 Modeling LFO-based effects (Phaser, Flanger)

Modeling Low-Frequency Oscillator (LFO) modulated, time-varying audio effects such as phasers and flangers may be feasable, but very time consuming using methods described in this thesis.

Some works offer a clever *gray-box* neural network approach using (LSTM) [30], the network's inputs include both the unprocessed audio signal and the corresponding LFO signal. The explicit inclusion of the LFO signal as an input greatly improves both model performance and interpretability.

5.3 Guitarless Dataset

As a final part of my thesis, I tried out training LSTM on a dataset with no guitar recordings. The idea is that guitar pedals although designed to change the guitar tone, can work on any audio signal including human voice or drums. Guitar pedals don't have idea to which type of signal is fed into their input.

After some experiments, I found out LSTM was indeed able to emulate a distortion guitar pedal successfully while trained on input-output pairs with no guitar recordings. When trained exclusively on guitar recordings, the model reached an MSE of 4.3×10^{-4} and ESR of 14.01. When trained on non-guitar recordings, it still achieved almost identical performance, with an MSE of 4.7×10^{-4} and ESR of 14.32. This confirms that the LSTM generalizes well to guitar signals even when trained on unrelated audio.

6 Conclusion

While the goal of this work was to provide a general model that would be good at emulating all types of guitar pedals, no such model exists with current technologies. Although some models work better on some types of effects, other models work better on other types of effects. There are tradeoffs to be made when learning a new effect, and without domain knowledge of which type of effect we are trying to emulate, it is very hard to predict which model should we choose to generalize well on all types of effects.

Some complex effects might even be impossible to model with any of these methods, as they either require too much data to even approximate them well, or can't be learnt at all because of modeling complexity.

References

[1] T. Wilmering, D. Moffat, A. Milo, and M. Sandler, “A history of audio effects,” *Applied Sciences*, vol. 10, p. 791, 01 2020. https://doi.org/10.3390/app10030791

[2] S. Braun and I. Tashev, “A consolidated view of loss functions for supervised deep learning-based speech enhancement,” 2020. [Online]. Available: https://arxiv.org/abs/2009.12286

[3] R. Simionato and S. Fasciani, “Comparative study of state-based neural networks for virtual analog audio effects modeling,” 2025. https://doi.org/https://doi.org/10.1186/s13636-025-00416-3

[4] J. Byun, S. Shin, Y.-C. Park, J. Sung, and S.-W. Beack, “Development of a psychoacoustic loss function for the deep neural network (dnn)-based speech coder,” pp. 1694–1698, 2021. https://doi.org/10.21437/interspeech.2021-2151

[5] R. M. Schmidt, “Recurrent neural networks (rnns): A gentle introduction and overview,” 2019. [Online]. Available: https://arxiv.org/abs/1912.05911

[6] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, pp. 1735–1780, 11 1997. https://doi.org/10.1162/neco.1997.9.8.1735

[7] A. Wright, E.-P. Damskägg, L. Juvela, and V. Välimäki, “Real-time guitar amplifier emulation with deep learning,” *Applied Sciences*, vol. 10, no. 3, 2020. https://doi.org/10.3390/app10030766

[8] A. Wright, E.-P. Damskägg, and V. Välimäki, “Real-time black-box modelling with recurrent neural networks,” 09 2019.

[9] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, “Wavenet: A generative model for raw audio,” 2016. [Online]. Available: https://arxiv.org/abs/1609.03499

[10] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, “Temporal convolutional networks for action segmentation and detection,” 2016. [Online]. Available: https://arxiv.org/abs/1611.05267

[11] S. Bai, J. Z. Kolter, and V. Koltun, “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling,” *arXiv preprint arXiv:1803.01271*, 2018.

[12] C. J. Steinmetz and J. D. Reiss, “Efficient neural networks for real-time modeling of analog dynamic range compression,” 2022. [Online]. Available: https://arxiv.org/abs/2102.06200

[13] Y.-T. Yeh, W.-Y. Hsiao, and Y.-H. Yang, “Hyper recurrent neural network: Condition mechanisms for black-box audio effect modeling,” 2024. [Online]. Available: https://arxiv.org/abs/2408.04829

[14] A. Gu, K. Goel, and C. Ré, “Efficiently modeling long sequences with structured state spaces,” 2022. [Online]. Available: https://arxiv.org/abs/2111.00396

[15] H. Yin, G. Cheng, C. J. Steinmetz, R. Yuan, R. M. Stern, and R. B. Dannenberg, “Modeling analog dynamic range compressors using deep learning and state-space models,” 2024. [Online]. Available: https://arxiv.org/abs/2403.16331

[16] R. Simionato and S. Fasciani, “Modeling time-variant responses of optical compressors with selective state space models,” 2025. [Online]. Available: https://arxiv.org/abs/2408.12549

[17] M. Comunità, C. J. Steinmetz, and J. D. Reiss, “Nablafx: A framework for differentiable black-box and gray-box modeling of audio effects,” 2025. [Online]. Available: https://arxiv.org/abs/2502.11668

[18] A. Gu and T. Dao, “Mamba: Linear-time sequence modeling with selective state spaces,” 2024. [Online]. Available: https://arxiv.org/abs/2312.00752

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[20] P. Verma and C. Chafe, "A generative model for raw audio using transformer architectures," 2021. [Online]. Available: https://arxiv.org/abs/2106.16036

[21] W.-T. Lu, J.-C. Wang, M. Won, K. Choi, and X. Song, "Spectnt: a time-frequency transformer for music audio," 2021. [Online]. Available: https://arxiv.org/abs/2110.09127

[22] H. A. Zubi, "Gascar: Genetically automated synthesizer configuration for audio replication," 2022. [Online]. Available: https://www.cs.ru.nl/bachelors-theses/2022/Hamzah_Al_Zubi___1047768___GASCAR_-_Genetically_Automated_Synthesizer_Configuration_for_Audio_Replication.pdf

[23] A. Johnson, "Sound resynthesis with a genetic algorithm," Imperial College London, Final Year Project, 2011. [Online]. Available: https://www.doc.ic.ac.uk/teaching/distinguished-projects/2011/a.johnson.pdf

[24] C. E. Perez, S. Dixon, and D. Stowell, "Active learning of intuitive control knobs for synthesizers using gaussian processes," in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, 2019, pp. 123–128.

[25] C. J. Steinmetz, N. J. Bryan, and J. D. Reiss, "Style transfer of audio effects with differentiable signal processing," *arXiv preprint arXiv:2207.08759*, 2022.

[26] C. J. Steinmetz, V. K. Ithapu, and P. Calamia, "Filtered noise shaping for time domain room impulse response estimation from reverberant speech," in *WASPAA*. IEEE, 2021.

[27] S. Nercessian, "Neural parametric equalizer matching using differentiable biquads," in *DAFx*, 2020.

[28] J. T. Colonel, C. J. Steinmetz, M. Michelen, and J. D. Reiss, "Direct design of biquad filter cascades with deep learning by sampling random polynomials," in *ICASSP*. IEEE, 2022.

[29] L. Juvela, E.-P. Damskägg, A. Peussa, J. Mäkinen, T. Sherson, S. I. Mimilakis, and A. Gotsopoulos, “End-to-end amp modeling: From data to controllable guitar amplifier models,” 2024. [Online]. Available: https://arxiv.org/abs/2403.08559

[30] A. Wright and V. Välimäki, “Neural modeling of phaser and flanging effects,” *Journal of the Audio Engineering Society*, vol. 69, pp. 517–529, 07 2021. https://doi.org/10.17743/jaes.2021.0029

Abstract

Emulation of Guitar Effects Using Machine Learning

Luka Ivanković

This thesis investigates machine learning approaches for modeling and emulating guitar effects pedals. We explore black-box and gray-box methods to capture the non-linear, dynamic, and time-dependent behavior of effects such as distortion and reverb. Black-box models, including LSTMs and WaveNet, learn end-to-end mappings from input to output without assuming nothing about the internal structure of guitar effects, while gray-box methods leverage partial knowledge of the effects structure combined with optimization techniques like genetic algorithm and gradient descent. Models are evaluated on guitar and non-guitar audio, demonstrating that machine learning can effectively emulate guitar pedals.

Keywords: Guitar Pedals, Machine Learning, Black-box optimization, Gray-box optimization

Sažetak

Emulacija gitarskih efekata primjenom strojnog učenja

Luka Ivanković

Ovaj rad istražuje pristupe strojnog učenja za modeliranje i emulaciju efekata gitarskih pedala. Istražujemo black-box i gray-box metode kako bismo uhvatili nelinearno, dinamično i vremenski ovisno ponašanje efekata poput distorzije i reverba. Black-box modeli, uključujući LSTM i WaveNet, uče preslikavanje pedale koju pokušavamo emulirati bez pretpostavki o unutarnjoj strukturi gitarskih efekata, dok gray-box metode koriste djelomično znanje o strukturi efekta kombinirano s tehnikama optimizacije poput genetskih algoritama i gradijentnog spusta. Modeli se evaluiraju na gitarskom i negitarskom zvuku, pokazujući da strojno učenje može učinkovito reproducirati ponašanje pedale.

Ključne riječi: Gitarske pedale, Strojno Učenje, Crna-kutija, Siva-kutija

Appendix A: The Code

As part of this research, I provide the original codebase that was used to create and analyze the dataset, train and evaluate the models, and measure the impulse responses of the audio effects. The structure of the code is organized as follows:

Core Files

- **Response.py** — Responsible for loading VST3 plugins, creating the dataset, and measuring the impulse response lengths of the effects.
- **dataset.ipynb** — A Jupyter notebook used for dataset analysis and exploratory evaluation.
- **Main.py** — The main script used to train the black-box models.

Model Implementations

- **models/** — Contains all black-box architectures used in this research.
- **grey_box_models/** — Contains grey-box models, including DASP-based and genetic approaches.

Utilities

- **helper/** — Contains various loss functions used in the experiments, helper utilities for saving and processing audio (`.wav`) files, and training/evaluation loops.
- **datasets/** — Includes the dataset handling code, such as chunking audio for training, as well as the custom `FrequencyQueueDataset` class for efficient data loading.

Excluded Folders

Due to their size, the `data/` and `plugins/` folders are not included in this repository. Their structure is as follows:

Data Folder

The dataset is split into three subfolders: `train`, `valid`, and `test`. Each subset is further divided into:

- **x/** — Contains unprocessed audio files, organized into `guitar/` and `other/` subfolders.
- **y/** — Contains processed audio files, also organized into `guitar/` and `other/` subfolders. Each of these is further divided by effect type:
  - `dragonflyPlateReverb`
  - `dragonflyRoomReverb`
  - `ragingDemon`
  - `simpleDist`

Plugins Folder

This folder contains the original VST3 plugins used to generate the dataset.