Joel A. Jaffe

Department of Music

University of California, Santa Barbara

June 2024

Closing the Gap Between Open-Source and Proprietary Amplifier Modeling

Abstract

This project investigates digital amplifier modeling, a family of techniques for emulating the sound of analog guitar amplifiers with digital computers. The author argues that free, open-source alternatives to proprietary modeling software offer equivalent sonic qualities, but suffer from a lack of support on intuitive hardware. A solution is proposed in the form of a hybrid system comprised of free, open-source software running on proprietary (but inexpensive) hardware. The author notes ways in which the hybrid system at present lacks parity with proprietary systems, developing software that both extends the system's functionality and establishes a framework for further work.

Background

In June of 2000, Fender Musical Instruments Corporation filed a patent application for a "simulated tone stack," a system for emulating the sound of analog guitar amplifiers using a digital computer. The authors of U.S. Patent No. 6222110 state that "An amplifier for an electric guitar has low fidelity [that] contributes to the 'voice' of the guitar to such an extent that the guitar and the amplifier together are the instrument, not the guitar alone."[1] In practice, this means

[1] *Curtis, Chapman, and Adams, Simulated tone stack for electric guitar.*

that a single electric guitar by means of different amplifiers can produce a near infinitude of timbres. The authors of Patent 6222110 sought to extend the generality of the digital computer as a synthesis instrument to the domain of signal processing, proposing a system that renders a single computer capable of reproducing existing signal processing for guitar while also introducing novel sounds.

The processing of musical signals with digital computers is a sub-domain of digital signal processing (DSP), a field that has many applications beyond audio signals. The field was and continues to be pioneered by researchers in telecommunications, particularly those working at Bell Labs. The underlying theory is that continuous variations in a quantity (i.e. sound pressure as registered by a microphone) can be measured as discrete values (known as samples) and stored as binary code. Indistinguishable playback of the original signal can be achieved by converting these discrete samples back into continuous signals using a digital-to-analog converter (DAC).

Despite the capabilities of digital signal processing, strong preferences for analog (specifically vacuum tube) processing for guitar have persisted into the 21st century.[2] These preferences are multifaceted, encompassing purely sonic qualities but also extending to attributes of form factor such as user interface (UI) and aesthetic. Commercial production of analog guitar amplifiers and effects processors developed in conversation with consumer taste throughout the 20th century, weaving a tapestry of innovation documented sonically in the recorded music of the era.

In the processing of musical signals, key differences between analog and digital are found in the connected concepts of fidelity and linearity. In signal processing generally, fidelity is a

[2] *Pakarinen and Yeh, "A Review of Digital Techniques for Modeling Vacuum-Tube Guitar Amplifiers."*

measure of the relationship between an input signal and the resultant output when that signal is fed through a processor. In analog processors, signal from electric instruments is passed through circuits whose componentry are affected by conditions of their environment such as temperature, humidity, electrostatic charge, magnetic fields, and more. Compounded with this environmental sensitivity is the fact that, like physical musical instruments, no two serial numbers are built exactly alike. These factors all contribute to the "low fidelity" described in U.S. Patent No. 6222110.

A perfectly linear response, the theoretical highest fidelity, results when a system can recreate exactly the signal that was input to it. Due to the laws of thermodynamics, a truly linear response is impossible in analog systems. The extreme low fidelity of early guitar amplification, a result of these limitations, led to a creative turn in the field: guitarists, sound engineers and amplifier manufacturers all turned their efforts not towards the most perfect recreation of input, but instead towards pleasing distortions. Musician/Scholar Robert Poss writes:

"The closest thing to pure electric guitar tone—when it is recorded directly from the pickup's output or when it is amplified using an amplifier virtually free of audible distortion—is a lifeless, one-dimensional sound. It lacks resonance and sustain; its rich overtones are muffled or inaudible; it has only the most vague and rudimentary sonic personality; and when unplugged, the solid-body electric is an instrument virtually without a voice. Yet when properly amplified and/or processed, it becomes the most varied, versatile and character-laden instrument imaginable, with both the ability to sustain tones like a bowed violin or cello and the dynamic, percussive range of a piano."[3]

[3] Poss, "Distortion Is Truth."

The term ‘distortion’ in the broader field of signal processing describes any discrepancy between an input and output signal due to processing. In the lexicon of electric guitarists, the word distortion is colloquially used to describe saturation, a type of distortion that occurs when the amplitude of an input signal exceeds the acceptable range of a processor, resulting in its zeniths of and azimuths being ‘clipped’ off (see figures 1 & 2).

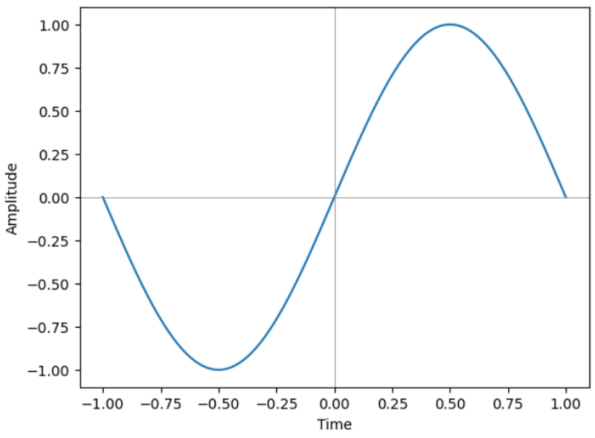

Figure 1: signal rendered with linear response

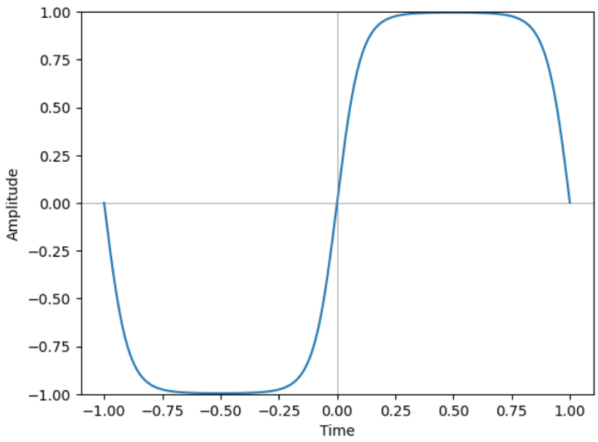

Figure 2: signal “clipped” by a limiting function

Early examples of distorted electric guitar timbres can be heard in the music of Bob Dunn, placing the inception of these timbres as artistic choice in the 1930s.[4] Over the decades that followed, amplifiers became increasingly capable of cleanly reproducing guitar signal, but a taste for distorted guitar had developed and notable guitarists experimented with creating increasingly ‘dirty’ tones. Jimi Hendrix popularized an effect colloquially known as fuzz, which clips a signal to such extent that resulting intermodulation distortion populates much of the signal’s spectral content with noise. Notable in Hendrix’s fuzz effect is that it was applied to his

[4] Millard, *The Electric Guitar*.

signal prior to the amplifier via a toggleable device, embodying the fact that this distortion was intentional.

Figure 3: Bob Dunn with a "combo" amp[5]

Figure 4:The Fuzz Face effects unit, popularized by Jimi Hendrix[6]

Modern tube amplifiers incorporate this preference for controllable distortion in their design. An amp's circuitry can be divided into the common yet somewhat confusing labels of 'preamp' and 'power amp' sections. The preamp circuitry generally amplifies the input signal by a factor of 30-50x with small 'preamp tubes,' whose limited capacity lends itself to overloading for the production of clip-distortion.

The power amp section takes the signal from the preamp section and amplifies it to a voltage suitable for driving loudspeakers. Its 'power tubes' are orders of magnitude more powerful than preamp tubes, allowing them to more cleanly reproduce signals at higher

[5] https://birthplaceofwesternswing.com/images/BobDunn1.png
[6] https://upload.wikimedia.org/wikipedia/commons/4/46/FuzzFace_Effect_Pedal.jpg

amplitudes. Because coloration of the signal in the power amp section is negligible compared to the preamp section, many amplifiers pair tube preamp sections with transistorized power amp circuits, which are smaller, lighter, cheaper, and offer much greater power efficiency.

Power amps typically output their signal to speaker cabinets, where electrical signal from the amplifier is converted back to physical motion by a loudspeaker. Purpose-built guitar cabinets are low fidelity by intention, designed to attenuate the noisy, abrasive high frequency information added to a signal during amplification. At every stage, signal processing for electric guitar is about transformation, not reproduction.

Figure 5: A “half-stack” consisting of a Marshall DSL100H “head” and Chute “cab”

Enter the digital computer. When audio signals are stored as binary code, the data is impervious to the elements and perfectly reproducible, in stark contrast to analog mediums such as magnetic tape or vinyl records. Digital audio offers genuine fidelity and linearity, and has resultantly become ubiquitous in recording and playback, areas where fidelity is desirable.

In the digital domain, signal processing occurs through mathematical transformations of the data, which offers extremely fine-grain control and nearly limitless possibilities. The

downside is that, while the nonlinear processing of analog is entirely possible in digital systems, it is much harder to implement than linear algorithms.

Software programs that contribute to the digital emulation of analog gear are collectively referred to as 'modeling software' and have become increasingly popular since 2000. In the early days of amplifier modeling, progress took the form of clever math that emulates the saturation of overloaded preamp tubes through 'wave-shaping' algorithms[7] that map each input sample to a corresponding output sample based on a function. After 2010, convolution became an increasingly popular operation in modeling algorithms, particularly for modeling the response of speaker cabinets. Convolution involves the generation of a multi-sample signal for every input sample, and was explored theoretically for application in digital audio as early as 1975[8] but wasn't deployed in commercial products until the 2010s due to the prohibitive amount of computing power that it requires.[9]

Modeling algorithms of any complexity are just mathematical theory until made usable for musicians. Many modeling techniques are implemented in 'plugin' software formats that allow them to run inside larger applications for audio editing (DAWS[10]), which run on general-purpose computers. While DAWs are a ubiquitous tool for music producers and audio engineers, the general-purpose computer is not an ideal form-factor for a gigging guitarist.

A counterpart to the general-purpose computer is the embedded system, a computer whose operation is constrained to the optimized execution of a limited range of programs. The first commercially successful digital audio processors were embedded systems made by Eventide

---

[7] For a survey of techniques in this family, see *Pakarinen and Yeh*.
[8] Rabiner and Gold, *Theory and Application of Digital Signal Processing*.
[9] Time-domain convolution requires N^2 calculations, where N is the length in samples of the longest of two signals being convolved.
[10] DAW stands for Digital Audio Workstation, a computer program that digitizes music mixing and mastering processes by emulating the role of a traditional recording console. Popular DAWS include Pro Tools, Logic Pro, and Live.

Inc in the 1970s. These devices leveraged digital recording and playback capabilities to produce a variety of time-domain effects, and were sold in a rackmount form-factor that integrated seamlessly into standard production workflows.[11]

Starting in the 1990s, guitar-specific devices called “floor modelers” began to circulate in the consumer market. Floor modelers are embedded systems that package a computer optimized for guitar-oriented DSP in a form factor familiar to guitarists- a pedalboard.[12] Early examples include the Boss GT-5 and Line 6 POD, modern examples include the NeuralDSP Quad Cortex and Line 6 Helix.

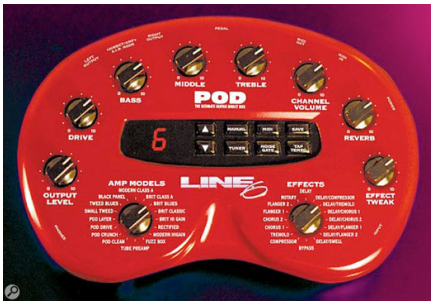

Figure 6: Line 6 POD circa 1998[13]

Figure 7: Line 6 Helix circa 2024[14]

When connected to a high-fidelity power amp and loudspeaker, one floor modeler can mimic the processing of a wide range of analog guitar amps and switch between sounds instantly. While capable of a wide range of DSP, floor modelers are generally constricted to the most popular sounds for guitar, with only a handful of parameters exposed to the user via foot-friendly buttons and knobs. Their major technical limitations are found in the engineering of

[11] https://www.eventideaudio.com/rackmount/ddl-1745/
[12] The pedalboard is a nearly ubiquitous feature of the modern guitar rig. Its most general form is a flat surface that holds an array of individual effects pedals, wired together via quarter-inch patch cords.
[13] https://dt7v1i9vyp3mf.cloudfront.net/styles/header/s3/imagelibrary/l/line6pod-header-0299-cbzFx4lBI_sd2yyPenZX9ZIbL0dC0tVA.jpg
[14] https://line6.com/data/6/0a020a41d6b56400ed40dec57/image/png

software that accurately mimics analog gear and does so within the computing limits of a floor modeler's processor.

Another popular form factor is the 'digital combo amp,' an embedded system that houses a DSP computer, power amplifier, and loudspeaker in one enclosure. Even more than floor modelers, digital combo amps embody analog look and feel in their design and user interface (UI), visible in the similarity between Figures 8 and 9. Early examples include the Line 6 AxSys 212 and Roland Cube 30, modern examples include the Fender Champion series and Boss Katana series.

Figure 8: Fender tube-powered combo amp[15]

Figure 9: Fender digital combo amp[16]

While the potential for analog user experience (UX) in embedded systems makes them ideal for amplifier modeling, the software that runs on them conforms to none of the standardization seen in plugin formats for general-purpose computers. This standardization allows for software developers to write one program that will work in a variety of DAWs, which

[15] https://www.fender.com/cdn-cgi/image/format=png,resize=height=auto,width=712/https://www.fmicassets.com/Damroot/ZoomJpg/10001/2172000000_amp_frt_001_nr.jpg

[16] https://www.fender.com/cdn-cgi/image/format=png,resize=height=auto,width=712/https://www.fmicassets.com/Damroot/ZoomJpg/10009/2330200000_amp_frt_001_nr.jpg

is conducive to the highly collaborative open-source ecosystem[17] (more on this later). Conversely, the highly proprietary world of embedded modelers pits manufacturers against each other and discourages collaboration in many ways.

For example, if Guitar Player A creates an excellent sound on his Brand X embedded modeler, there is no way for them to share their settings with Guitar Player B, because they own a Brand Y embedded modeler. In some cases, there is not even an interface for sharing settings between Brand X Serial 1 and Brand X Serial 2. In contrast, modeling algorithms implemented in plugin formats can almost always exchange settings via standardized filetypes.

Enter Neural Amp Modeler[18] (NAM), an open-source project started in 2022. NAM allows users to create digital models of their analog gear at home, and use these models in the context of plugin software. NAM users can store and share their models with others on a dedicated website[19] and test models without downloading them using a web implementation of the plugin.[20] While NAM has been successful in bringing crowdsourced, shareable modeling to the world of general-purpose computers, its implementation in embedded systems is nascent yet rapidly evolving.

The creation of a 'hardware host' for NAM mandates at least some level of proprietary technology, most inarguably in the manufacture and distribution of its physical componentry. Some companies offer complete embedded systems that utilize NAM, while others deal in components and software that allow hobbyists to convert general-purpose minicomputers (i.e. the highly popular Raspberry Pi) into digital modelers. The problem with the former is that it

[17] The open-source ecosystem refers to a decentralized community of computer programmers who share and collaborate on source code for free digital tools over the internet.
[18] "Neural Amp Modeler | Highly-Accurate Free and Open-Source Amp Modeling Plugin."
[19] "ToneHunt | Sound Better!"
[20] "Neural Amp Modeler Online."

suffers from gatekeeping similar to that found in other proprietary modelers, the problem with the latter is that requires a level of computing proficiency that is a barrier for many musicians.

The goal of this project is to present a prototype embedded system that extends an existing hardware/software combination with a more intuitive user interface. The prototype is intended to represent a preferrable future of embedded modelers that accept a standardized filetype, allowing guitarists to participate in a free, open-source ecosystem of digital amp modeling that will enfranchise even the least technically inclined players.

Design

In 2007, the authors of aforementioned U.S. Patent No. 6222110 produced another patent, U.S. Patent No. 2007/0234880, which describes an embedded system that runs amplifier modeling and effects software for electric guitar. This patent culminated in the Tone Master Pro; a floor modeler released by Fender Musical Instruments Corporation in 2023. The year prior saw the release of the notable Quad Cortex modeler, made by NeuralDSP, a company who previously only made modeling software for general purpose computers. They were both priced north of $1500 USD, a price point established by the Line 6 Helix modeler, released in 2015.

While these products represent valuable innovation, their retail cost is prohibitive to many, and arguably unwarranted. The majority of their function can be performed by hardware one tenth their cost, and highly accessible free software. This disconnect between production cost and retail price mirrors a central thread in the history of personal computing: the OS (operating system) wars. Living in the aftermath, most consumers today still pick between the same players as the 1980s: Apple (MacOS) and Microsoft (Windows). However, a key difference today is that there's a third option- and it's free. Technically inclined readers may

know already that I'm talking about Linux, a free operating system published on the web in the early 1990s. Author Neal Stephenson observed in a 1999 essay that "Apple…created a machine that discouraged hacking, while Microsoft…had created a vast, disorderly parts bazaar--a primordial soup that eventually self-assembled into Linux."[21]

Since then, Linux has grown into a family of distributions (distros), a term denoting the Linux operating system packaged with other software. Think of distributions like versions of Windows or MacOS- the differences between Windows 10/11 and MacOS Ventura/Sonoma. While versions of Windows and MacOS progress in a linear fashion, always incorporating more features, Linux development is characterized by the parallel evolution of specialized distributions, each tailored to a specific functionality.

An example of a highly specialized distribution can be found in PatchboxOS, a free Linux distro made by a Lithuanian company called Blokas. In their own words: "Patchbox OS is a custom Linux distribution specially designed for Raspberry Pi based audio projects.[22]" It's a natural choice of operating system for this project, which seeks to demonstrate the powerful combination of cheap hardware and free software.

The hardware in question is the Raspberry Pi, a pocket-sized single-board computer. Initially developed to promote education in digital computing, the Pi quickly found a huge audience in hobbyists following its initial release in 2012, and is even a component in some commercial products such as Korg's Wavestate, Modwave and Opsix digital synthesizers.[23] The Raspberry Pi 4B is the model of choice for this project, as it has the robust computing power necessary for real-time audio processing while also being relatively inexpensive (~$60). It's also

[21] Stephenson, *In the Beginning...Was the Command Line.*
[22] "Patchbox OS – Raspberry Pi OS for Audio Projects."
[23] Douglas, "Raspberry Pi Synthesizers - How the Pi Is Transforming Synths."

the model best supported by the PiSound- a device that allows for input and output of musical signals via MIDI or quarter-inch TRS cables.

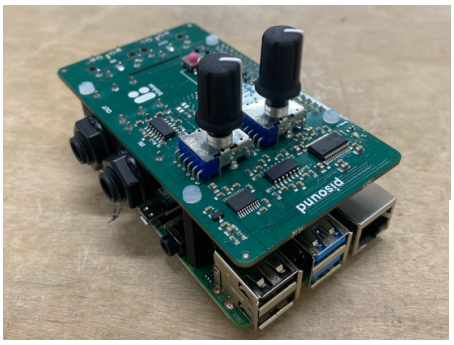

Figure 10: Raspberry Pi 4B with PiSound attached

For those interested in real-time effects processing using the Pi, PatchboxOS comes with a software package called MODEP, made by MOD Audio. MODEP is described as a "virtual pedalboard"[24] that allows users to pass their input signal through a series of digital audio plugins, including a specialized version of Neural Amp Modeler. The user can configure their pedalboard remotely via a web-based graphical user interface (GUI), one that strikingly resembles the GUI on the Fender Tone Master Pro.

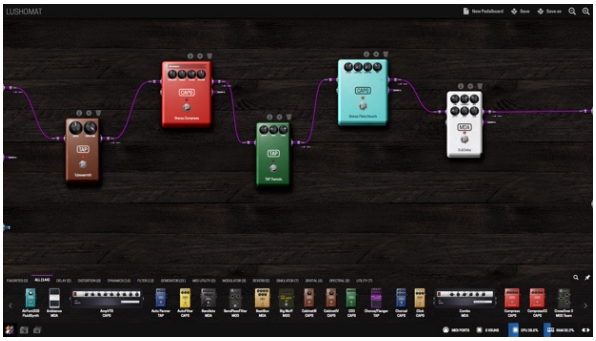

Figure 11: MODEP Web GUI[25]

Figure 12: Fender TMP Touchscreen GUI[26]

[24] "MODEP – Virtual Pedalboard for Raspberry Pi."
[25] https://community.blokas.io/uploads/default/original/1X/9d0fbb7e63ae0068432a0268d7500cac0a536c55.jpeg
[26] https://media.sweetwater.com/m/products/image/0791eed74fGkDfurKsNa71VnMrCs9fKRytnPiRPp.jpg?v=0791eed74f2d25a9

While these GUIs do a nice job capturing the look of a pedalboard, they don't mimic the feel- stomping on them won't do much other than damage whatever device is displaying them. To add foot-switchable functionality, MODEP must be extended by additional hardware controls (footswitches, knobs, etc.) and software that allows added hardware to communicate with MODEP. Extension is common in the open-source community, typically taking the form of programmers contributing functionalities to personal copies of existing freeware and asking the original author to incorporate their changes (more on this later). In the spirit of open-source, this project aims to extend the user interface of PatchboxOS/MODEP by developing and distributing free software, and also suggesting ways in which this software may be extended further.

The software extends the pairing of Neural Amp Modeler and MODEP, allowing users to load files from an external memory device without having to interact with the GUI. By enabling the user to load an amplifier model by simply attaching a USB drive to their Raspberry Pi, this software contributes to a more "analog" user interface, in hopes of inching open-source modeling towards a state of competition with proprietary modelers.

There are many possible uses. One example may be a case of multiple users, where an embedded system running MODEP/NAM needs to accommodate the sounds of many different guitarists over the course of a multi-artist show. Rather than each guitarist bringing their own amplifier, they can simply bring USB drives containing models of their amplifiers at home, or models downloaded from the internet. With further extension, this software could load an entire suite of effects, including amp model, from USB. Even further, the bypass state or settings of parameterizable effects (i.e. the TIME knob on a delay effect) could be modulated by external, intuitive hardware such as a buttons or dials connected to the Pi either physically by wires or through wireless protocols such as WI-FI and Bluetooth.

Virtually infinite extensions are possible. The impact of this software's limited function is dwarfed by its role as proof of concept for a general system of interaction with MODEP that bypasses the GUI. The core concept is such: the entirety of MODEP's function is parametrizable via text files on the host system's memory, and this text can be edited by automatable interactions with the computer's operating system.

The ubiquitous tool for automating interaction with a computer's filesystem is called a shell script. To understand them, it's important to first understand what a shell is. Before graphical user interfaces were popular, most computers were operated by a command line interface (CLI), which involves a user typing out text instructions to a computer. This text is not natural language, but instead a series of instructions fed to a program called a command line interpreter. A shell is a command line interpreter that works in the context of a computer's entire operating system, as opposed to a CLI for an application.

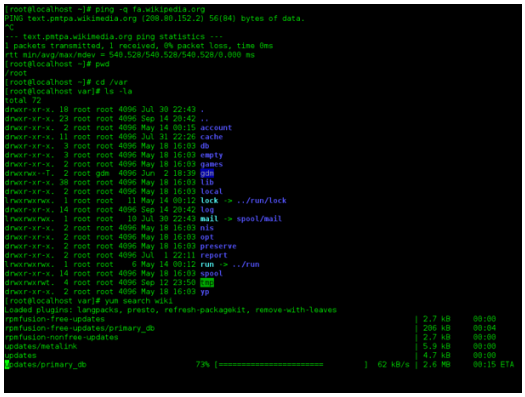

Figure 13: Example of a Linux CLI[27]

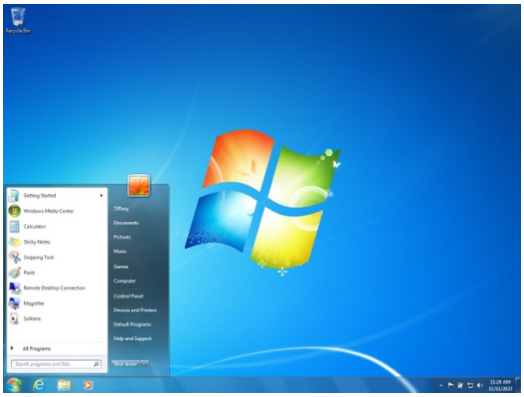

Figure 14: Windows 7 GUI[28]

[27] https://upload.wikimedia.org/wikipedia/commons/2/29/Linux_command-line._Bash._GNOME_Terminal._screenshot.png

[28] https://upload.wikimedia.org/wikipedia/en/thumb/5/50/Windows_7_SP1_screenshot.png/600px-Windows_7_SP1_screenshot.png

While learning the conventions of a given command line interpreter can be challenging, it is an extremely powerful tool due to the fact that it is highly automatable. Instead of giving commands to a shell one line at a time, a sequence of commands can be written into a text file, and the interpreter will read and perform them in order. This is known as shell scripting.

Thankfully, the Bourne-Again SHell (bash) is endemic to Linux development, so there is a more or less ubiquitous shell language spoken by nearly all Linux distros, including PatchboxOS. The software component of this project is a bash script, a shell script meant to be interpreted by the bash shell.

The first step of the script is to perform variable assignment, one of the most fundamental operations in programming languages. Variable assignment allows for the reference of a value via a unique character or group of characters. The graphic below demonstrates how numbers or the result of operations can be stored as variables and easily referenced later in a program's source code.

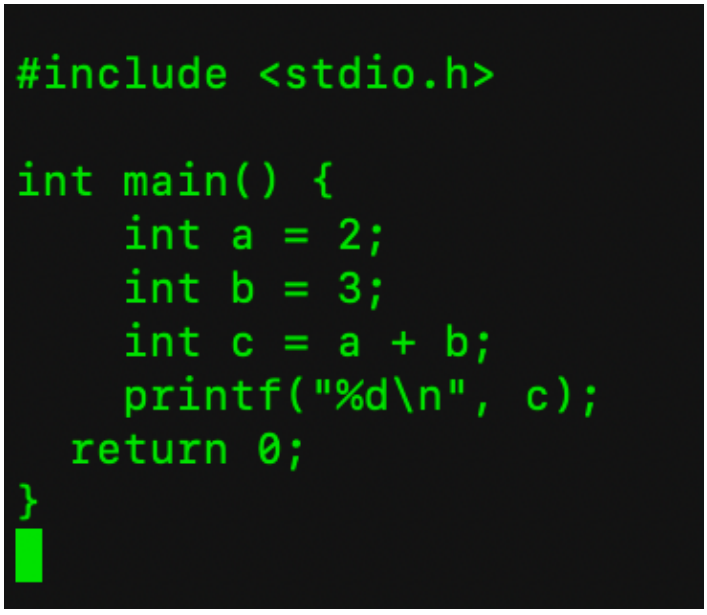

Figure 15: A simple C program demonstrating variable assignment

The variable assignment in the beginning of our shell script allows us to reference a file path (20+ characters) with a concise symbol. The first variable, `models_dir`, represents the directory (aka folder) where .nam files are visible to instances of the NAM plugin in MODEP. The second variable, `board_path`, represents the location of the directory that contains configuration files for a MODEP pedalboard. The remaining variables are used to reference specific files and subdirectories within `board_path`.

The next set of commands checks to see if an external memory device is attached to the Raspberry Pi. This is done by calling the command `lsblk`, which outputs the names of all memory devices of which the computer is aware. The output of this process is fed into the input of another process called `grep`, whose output is fed to a process called `tail`, whose output is fed to another process and so on. This method of using the output of one shell command as the input to another is called 'piping,' and is analogous to the serial processing of pedalboards. It allows for the creation of powerful algorithms by chaining together many simple ones. The output of this 'pipeline' of shell commands is assigned to a variable, `usb_part`, which is referenced in the following line. This demonstrates how variables make our code more 'human readable,' a desirable quality in software development generally and especially code intended for pedagogical use.

The next part of the shell script is an `if` statement, one of the most fundamental data structures in computer programming. The anatomy of an `if` statement can be divided into two parts: a conditional expression and a body. In the simplest form of an `if` statement, instructions in the body are only executed if the conditional expression is evaluated as `true`. In a more

advanced version, the `if/else` statement, there is alternative code in the body of the `else` statement that is executed if the conditional is `false`.

```
#include <stdio.h>

int main() {
    int age = 19;

    if (age < 21) {
        printf("User is below the legal drinking age");
    } else {
        printf("Let's party!");
    }
  return 0;
}
```

Figure 16: A simple C program demonstrating an `if/else` statement

Conditionals in `if` statements must ultimately boil down to a binary answer, `true` or `false`. This is known as Boolean logic, and is fundamental to digital computing, which at the hardware level deals with binary: the 1s and 0s (`True` and `False`) readers may be familiar with. In the case of evaluating mathematical conditionals (i.e. $x > 1$), numerical logic built into programming languages can assess the truth of expressions, assuming x is a real number.

The `if` statement in our shell script doesn't evaluate mathematical values however, instead operating on a "string" of letters. In the case of strings (a collection of letters), conditionals are assessed on exact equivalence- so ("string" == "stirng") would evaluate as `false`. Our `if` statement checks to see if the value of `usb_part` is anything other than an empty string (i.e. no external memory was found).

Because the remainder of the shell script exists inside the body of the `if` statement, the process is terminated if the conditional evaluates as `false`. This means that if no external

memory is attached, no edits will be made to any files. If the conditional is evaluated as `true`, the following code is executed:

If an external memory device is attached, its contents are made accessible via the Linux filesystem using a command called `mount`. The `mount` command exposes the contents of the memory device in a place that the user specifies, in the case of our shell script, the `/media` folder. `/media` was chosen because it is empty on a fresh install of PatchboxOS, like it is on many Linux distros. This minimizes the possibility of files unrelated to the shell script interacting with its processes.

Once the external memory is mounted at `/media`, the shell script searches it for a .nam file using alphabetical logic with the find command. If it finds one, its name is saved to the `model` variable, and it is copied to `models_dir`. After the file is copied, the external memory is 'ejected' with the `umount` command.

The shell script then copies the .nam file in `mdls_dir` to a directory within `board_path`, after checking for an existing .nam file and deleting it (`rm`) if one exists. The `effect_dir` directory holds the configuration files for the pedalboard's instance of neural amp modeler.

Finally, the script edits the text of a file within `effect_dir`, which configures the pedalboard's instance of the NAM plugin to use the new model for its processing.

An important question is when this program should be executed. Most operating systems have programs that run constantly or frequently at a regular interval, known as background applications or daemons. Because processing power is often a precious resource on embedded systems, and particularly in real-time audio devices, the author recommends this script run once every time the Raspberry Pi is turned on (at 'boot').

```
  GNU nano 5.4                                                    load-nam.sh
#!/bin/bash

# Shell script for my Senior Project - a hardware host for Neural Amp Modeler
# https://www.neuralampmodeler.com/

# designed for PatchboxOS with MODEP module
# https://blokas.io/patchbox-os/docs/

# Reads .nam file from an external memory device, copies to disk,
# and configures pedalboard to use it

# filepath/directory variables
models_dir=/var/modep/user-files/'NAM Models'
board_path=$(grep -o '[^"]*\.pedalboard' /var/modep/last.json)
board_name=$(basename -s .pedalboard $board_path)
effect_num=$(grep -A 1 "neural-amp-modeler-lv2" "$board_path/$board_name.ttl" | grep Number - | grep -oP " \K\d+")
effect_dir="$board_path/effect-$effect_num"

# search for external memory devices
usb_part=$(lsblk -n -o NAME -p | grep -v 'mmcblk0' | tail -n +2 | awk -F/ '{print $(NF-1) "/" $NF}' | sed 's|^|/|')

# check if external partition exists
if [ -n "$usb_part" ]; then

    # mount partition at /media
    sudo mount "$usb_part" /media

    # look for .nam model and assign name to a variable
    model=$(basename $(find /media -maxdepth 1 -name "*.nam"))

    # copy $model to /var/modep/user-files/'NAM Models'
    sudo cp "/media/$model" /var/modep/user-files/'NAM Models'

    # unmounts partition
    sudo umount /media

    # search $effect_dir for .nam model and assign to variable
    current_model=$(find "$effect_dir" -name "*.nam")

    # delete $current_model if it exists
    if [ -n "$current_model" ]; then
        sudo rm $current_model
        fi

    # copy .nam model to $effect_dir
    sudo cp "$models_dir/$model" "$effect_dir"

    # modify contents of effect.ttl
    sudo sed -i -E "s/([[:alnum:]_]+\.nam)/"$model"/g" "$effect_dir/effect.ttl"
```

Figure 17: The shell script, `load-nam.sh`

The way to schedule the execution of tasks on most Linux distros is `cron`, a utility that schedules the running of shell scripts or 'cron jobs'. `Cron` is configurable with a `crontab` file, who's text dictates when a cron job should be executed. Figure 18 depicts a `crontab` file. By stipulating that our shell script run at boot, our script is prevented from interfering with audio processing. The UI consequence is that a reboot is necessary to load a new model.

```
  GNU nano 5.4                                                /tmp/crontab.
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
@reboot /usr/local/bin/load-nam.sh
```

Figure 18: a `crontab` with `load-nam.sh` scheduled to run at "boot"

Development & Testing

Plugging a guitar into MODEP for the first time can be tremendously empowering. Loading up a blank pedalboard, the output signal is initially identical to the input- audible as the "lifeless, one-dimensional sound" Poss describes in his analysis of "pure" electric guitar tone.[29] While sonically lackluster, this purity represents the infinitude of possibilities endowed by digital audio, in turn suggesting to the user that any sound audible in their imagination can be generated by a $60 computer.

Dragging an instance of Neural Amp Modeler into the virtual pedalboard's web GUI, the user can now emulate the sounds of thousands of guitar amplifiers by simply downloading .nam

[29] Poss, "Distortion Is Truth."

files from ToneHunt.[30] Loading up a high-gain tone, the user will be met by the same noisy screech of a real tube amp with its gain cranked up to 11, but can quickly mitigate it by dragging a virtual noise gate in front of their NAM instance- had the amp been physical, the user would have to acquire a dedicated noise gate unit, which can cost $60 unto itself.

The above situation describes the key strength of digital audio: the ability of one device to chameleonically shift between disparate sounds at nearly the speed of light. This is the functionality described in U.S. Patent No. 6222110. The key weakness is user interface- to orchestrate these changes in sound with MODEP, the user would typically access the web GUI via another computer attached to the same WI-FI network as the Pi, or operate the Pi itself via a CLI by attaching a keyboard and monitor. This project demonstrates a third method: controlling the Pi via button-toggled shell scripts.

In software development, it is generally considered good practice to break large tasks into smaller ones. A benefit of this tactic is that it allows for unit testing, where each component of a program is tested individually as it is written and, as a result, trustworthy later in development. One common 'debugging' tool is console logging, in which the output of individual functions are 'printed' to the console, so that their output can be compared with expectations. Figure 19 shows a script made for the unit testing of variable assignment, and Figure 20 shows a console dialogue alternating between its modification (`sudo nano /test.sh`) and evaluation (`/test.sh`).

[30] "ToneHunt | Sound Better!"

```
  GNU nano 5.4                                    test.sh
#!/bin/bash

# filepath/directory variables
foo="bar"
models_dir=/var/modep/user-files/'NAM Models'
#board_path= $(grep -o '"[^"]*\.pedalboard"' "/var/modep/last.json")
#board_name= $(basename -s .pedalboard $board_path)
#effect_num= $ (grep -m1 -B2 "neural_amp_modeler_lv2/input" "$board_name.ttl" | grep -oP ":b\K\d

echo $foo
echo $models_dir
#echo $board_path
#echo $board_name
#echo $effect_num
```

Figure 19: A shell script that tests variable assignment

```
/test.sh: line 5: /var/modep/user-files/NAM Models: Is a directory


patch@patchbox:/ $ sudo nano test.sh
patch@patchbox:/ $ sudo nano test.sh
patch@patchbox:/ $ /test.sh
/test.sh: line 4: bar: command not found
/test.sh: line 5: /var/modep/user-files/NAM Models: Is a directory
$foo

patch@patchbox:/ $ sudo nano test.sh
patch@patchbox:/ $ /test.sh
/test.sh: line 4: bar: command not found

patch@patchbox:/ $ sudo nano test.sh
patch@patchbox:/ $ sudo nano test.sh
patch@patchbox:/ $ /test.sh
/test.sh: line 4: bar: command not found

patch@patchbox:/ $ sudo nano test.sh
patch@patchbox:/ $ /test.sh
bar
patch@patchbox:/ $ sudo nano test.sh
patch@patchbox:/ $ /test.sh
bar
patch@patchbox:/ $ sudo nano test.sh
patch@patchbox:/ $ /test.sh
/test.sh: line 4: bar: command not found
/test.sh: line 5: /var/modep/user-files/NAM Models: Is a directory


patch@patchbox:/ $ sudo nano test.sh
patch@patchbox:/ $ /test.sh
bar
/var/modep/user-files/NAM Models
patch@patchbox:/ $ sudo nano test.sh
patch@patchbox:/ $
```

Figure 20: Unit testing of `test.sh` using console logging

As the script grew in complexity, unit tests began encompassing the movement and editing of files, the primary operations of the script. In its final form, the script was confirmed to perform all intended functions, configuring NAM with models from external memory at boot without any other interaction.

It's important to state that this software needs further development and testing. While it has been verified as functional on the author's Raspberry Pi, there's no guarantee it'll work on every, or even any, other one. This is one of the ways in which the open-source community is so valuable: it provides a forum for crowdsourcing both testing and improvements from a worldwide userbase. This typically takes the form of a 'pull request' on GitHub, a free webservice that hosts copies of git repositories.

Git is a free software tool that is nearly ubiquitous among software engineers, and comes pre-installed with many operating systems. It was written and published by Linus Torvalds, the man behind Linux, in 2005. It's an example of version-control software, which allows users to save snapshots of their work. Instead of destructive saving like the kind seen in text editors like Microsoft Word, where the save feature overwrites the past version, git keeps track of every 'save' (called a 'commit') so that changes between commits can be evaluated and historic copies resurrected at any time. Git doesn't operate on single files however, but instead entire directories.

When a directory is tracked by git, it is called a git repository ('repo' for short). Git was designed for collaborative software projects, such as the maintaining of the Linux kernel. A key feature is the concept of a 'remote,' a public copy of a git repository that can be accessed via the internet. A user can install a copy of a remote repository by using the `git clone` command on a computer connected to the internet, and incorporate changes on their computer ('local' changes) to the remote using commands like `git push`.

On GitHub, remote repositories have an owner, who sets permissions regarding how other users can incorporate their changes. A pull request allows users to submit changes to a copy of the remote for review by the owner, who can incorporate the changes into their repo if they are deemed worthy. With further development, this project could be incorporated into the

PatchboxOS/MODEP repositories via a pull request. If the reader wishes to download and use or modify the shell script, they can access it at https://github.com/joeljaffesd/MODEP-Scripting.

Conclusion

The research presented in this article aims to demonstrate that the dominance of proprietary industry in amplifier modeling is contingent on embedded digital systems, and that open-source alternatives are capable of overturning this dominance with minor extension. While this project was successful in extending the functionality of the author's Raspberry Pi, its major goal is to provide a framework for extending the functionality of Neural Amp Modeler and MODEP for all users, paving the way for inexpensive embedded modelers that may promote a greater acceptance of digital modeling among electric guitarists.

The author invites readers to consider ways in which free and open-source tools are superior to proprietary options. The pairing of NAM and MODEP allows a single, inexpensive computer to provide a greater variety of electric guitar tones than any proprietary modeler on the market currently, and their open-source codebases invite users to both customize personal copies of the software and propose improvements affecting all users.

Given that this article was written only two years after NAM's initial development, the author has confidence NAM will continue to develop and attract more users, documented at present in daily updates to its codebase.[31] The author would like to thank NAM developer Steven Atkinson for creating and maintaining the project, and also for participating in an interview.

New versions of PatchboxOS and MODEP that will be compatible with the recently-released Raspberry Pi 5 are under development and will greatly extend possibilities for open-

[31] "The Code."

source embedded modeling. The author looks forward to more projects extending their user interface.

Lastly, the author would like to ask readers that they consider the adoption of more robust digital audio curriculum in university music education. Creating digital audio tools is one of the most valuable interactions possible between music academia and practicing musicians, and can transcend the problematic identity politics endemic to Eurocentric conservatory-style education and its adjacent music theory, musicology, and ethnomusicology disciplines. If the reader is interested in digital audio curriculum, the author invites them to visit https://github.com/joeljaffesd/Intro-to-Digital-Audio.

Works Cited

Curtis, Dale Vernon, Keith Lance Chapman, and Charles Clifford Adams. Simulated tone stack for electric guitar. United States US6222110B1, filed June 15, 2000, and issued April 24, 2001. https://patents.google.com/patent/US6222110B1/en.

Douglas, Adam. "Raspberry Pi Synthesizers - How the Pi Is Transforming Synths." gearnews.com, April 19, 2024. https://www.gearnews.com/raspberry-pi-synthesizers-how-the-pi-is-transforming-synths/.

Millard, A. J., ed. *The Electric Guitar: A History of an American Icon*. Baltimore: Johns Hopkins University Press, 2004.

"MODEP – Virtual Pedalboard for Raspberry Pi." Accessed May 18, 2024. https://blokas.io/modep/.

Neural Amp Modeler. "Neural Amp Modeler | Highly-Accurate Free and Open-Source Amp Modeling Plugin." Accessed May 18, 2024. https://www.neuralampmodeler.com.

Neural Amp Modeler. "The Code." Accessed May 28, 2024. https://www.neuralampmodeler.com/the-code.

"Neural Amp Modeler Online." Accessed May 18, 2024. https://www.thenam.online.

Pakarinen, Jyri, and David T. Yeh. "A Review of Digital Techniques for Modeling Vacuum-Tube Guitar Amplifiers." *Computer Music Journal* 33, no. 2 (June 2009): 85–100. https://doi.org/10.1162/comj.2009.33.2.85.

"Patchbox OS – Raspberry Pi OS for Audio Projects." Accessed May 18, 2024. https://blokas.io/patchbox-os/.

Poss, Robert M. "Distortion Is Truth." *Leonardo Music Journal* 8 (1998): 45–48. https://doi.org/10.2307/1513399.

Rabiner, Lawrence R., and Bernard Gold. *Theory and Application of Digital Signal Processing*. Englewood Cliffs, N.J: Prentice-Hall, 1975.

Roads, Curtis. *The Computer Music Tutorial*. Cambridge, Mass: MIT Press, 1996.

Stephenson, Neal. *In the Beginning...Was the Command Line*. USA: William Morrow & Co., Inc., 1999.

"ToneHunt | Sound Better!" Accessed May 18, 2024. https://tonehunt.org/.