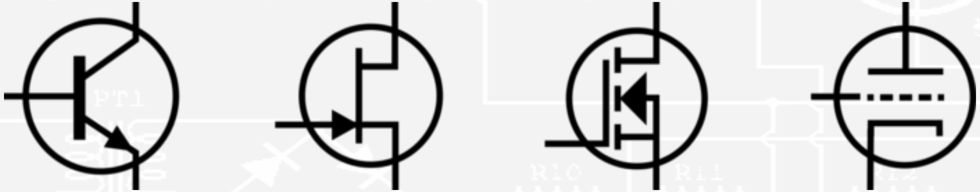
Guitar Effects Chain Plugin
“Amp Master”



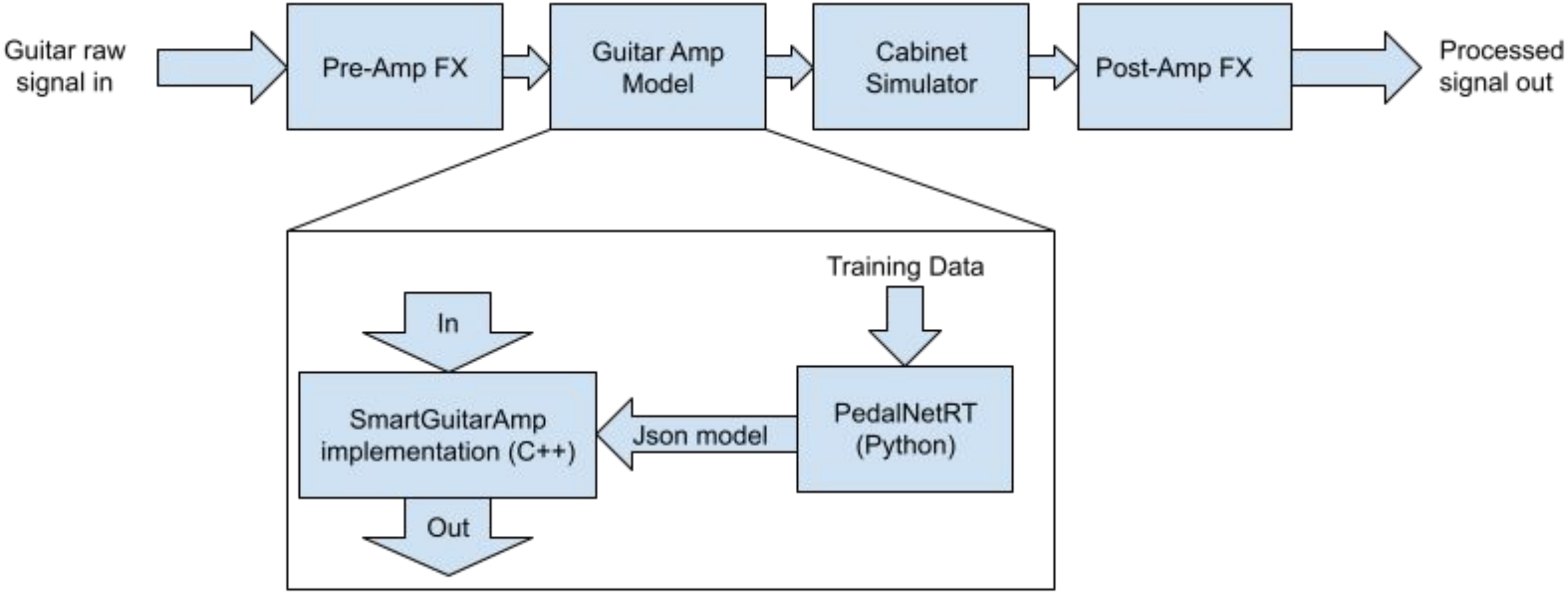
Group B - Mir Jeffres, Shan Jiang, Vedant Kalbag, Jocelyn Kavanagh, Thiago Roque

Objectives and Features

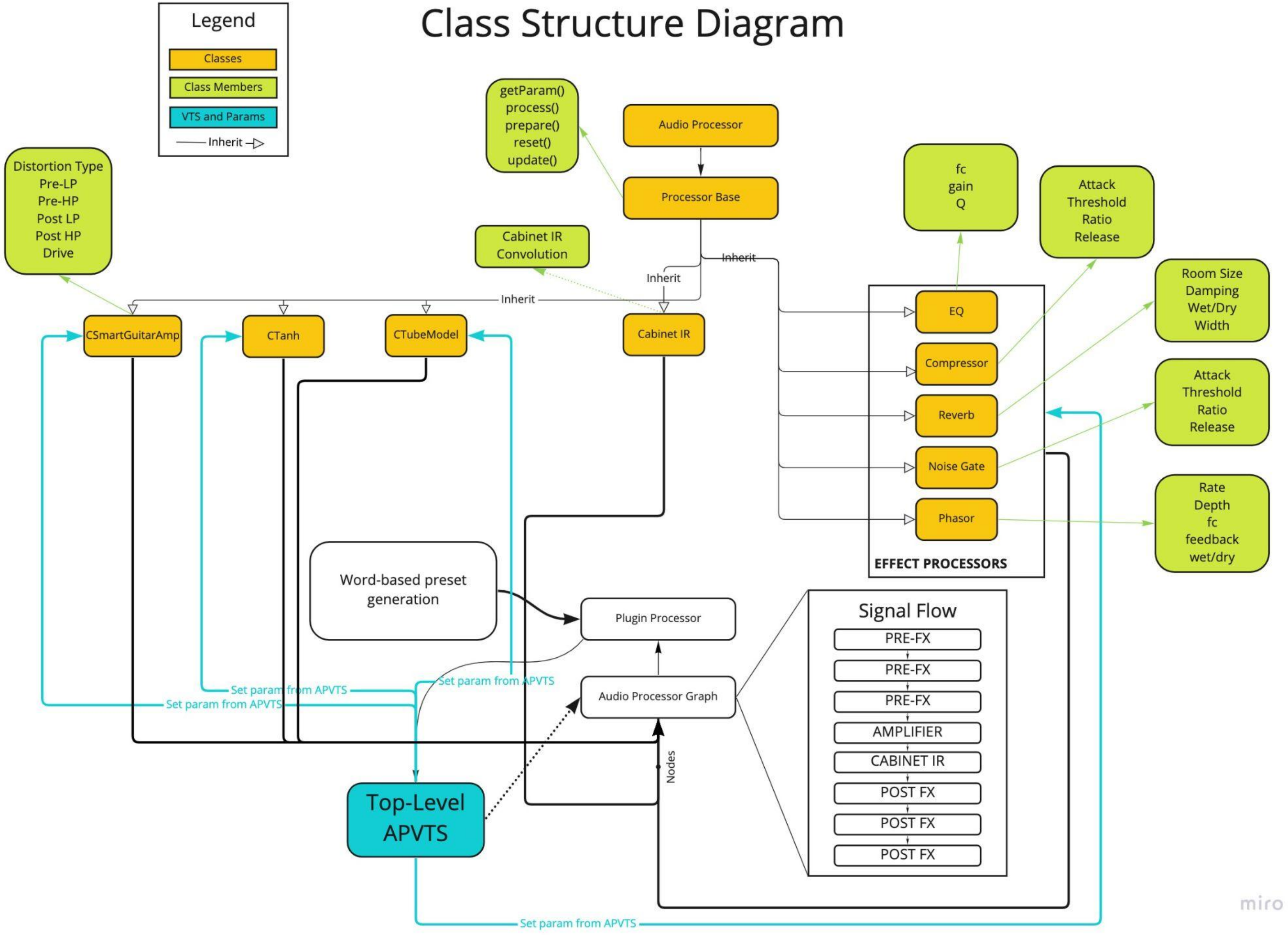
- Create a guitar amplifier simulator with multiple amp models
  - Include ML based guitar tone emulation models (SmartGuitarAmp)
  - Include a convolution-based cabinet simulator with an IR loader
- Create an FX chain for pre- and post-amplifier signal processing
- Have flexibility in multiple effects for use in the processing chain:
  - EQ
  - Compression
  - Reverb
  - Noise Gate
  - Phaser
- Include user-specified effect presets to match desired tone

Project Goals	Project Results
Create an amplifier simulator with multiple amp models	3 Amp simulators - TanhWaveshaping, Analog, and SmartGuitarAmp
Include ML based guitar tone emulation models	Integration of SmartGuitarAmp models
Create an dynamic FX chain for pre- and post-amplifier signal processing	Static FX chain with bypassing
Include automatic effect estimation to easily make new effect presets	Matching user-specified guitar tone words to effect presets (many-many map)
Include a convolution-based cabinet simulator with an IR loader	Convolution based cabinet simulator included with static IR

Guitar Amp Simulator Signal Flow



Class Structure Diagram



Distortion Algorithms

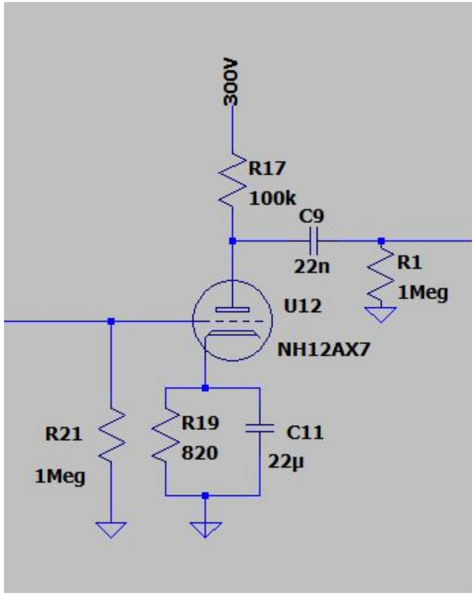
- Static waveshaper based on Tanh function
- Analog emulation of tube amplifier based on SPICE models and mathematical model of tone stack
- WaveNet-based distortion (SmartGuitarAmp)

Analog Emulation - Triode

- Vacuum tube emulation based on the modified Norman L. Koren model

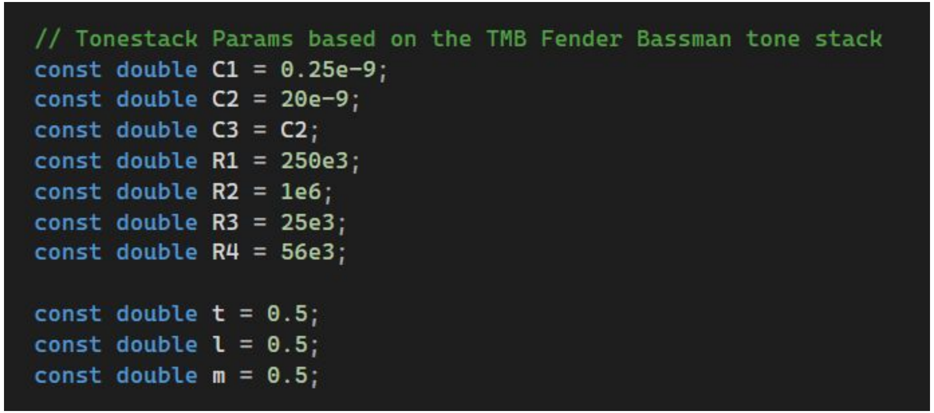
$$E_1 = ln(1 + exp(K_p(\frac{1}{\mu} + \frac{V_{gk} + V_{ct}}{\sqrt{K_{vb} + V_{pk}^2}})))$$

$$I_p = \begin{cases} 2E_1^{E_x}/K_g \text{ if } & E_1 \geqslant 0 \\ 0 & otherwise \end{cases}$$



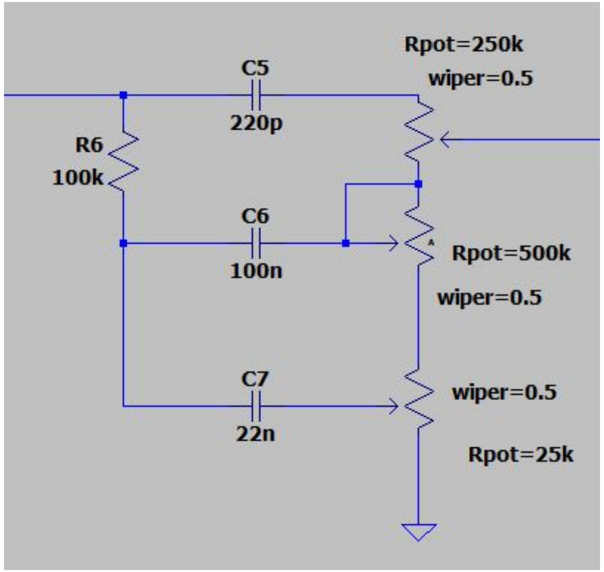
Analog Emulation - Tone Stack

- Mathematical model of the traditional TBM tone stack



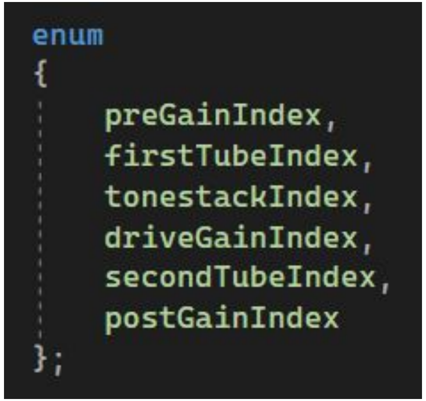
```
// Tonestack Params based on the TMB Fender Bassman tone stack
const double C1 = 0.25e-9;
const double C2 = 20e-9;
const double C3 = C2;
const double R1 = 250e3;
const double R2 = 1e6;
const double R3 = 25e3;
const double R4 = 56e3;

const double t = 0.5;
const double l = 0.5;
const double m = 0.5;
```

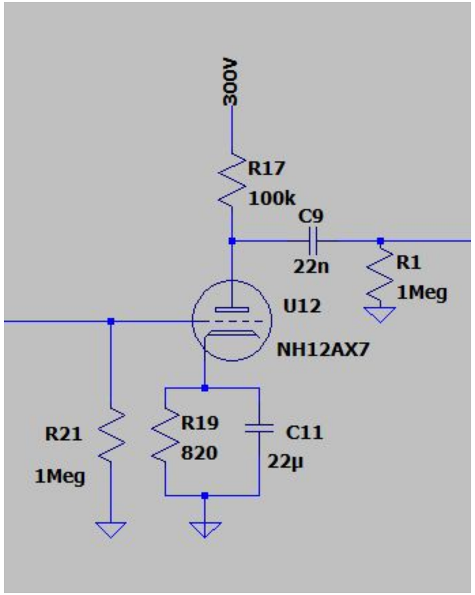


Analog Emulation - Modular Structure

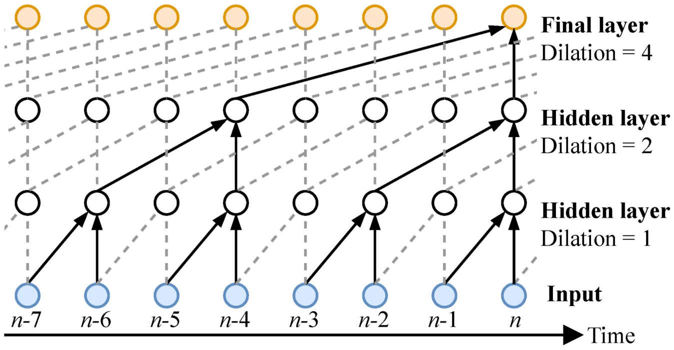
- Modular structure capable of combining triode segments with gain modules and filters, capable of constructing full emulation of real pre-amplifiers



```
enum
{
    preGainIndex,
    firstTubeIndex,
    tonestackIndex,
    driveGainIndex,
    secondTubeIndex,
    postGainIndex
};
```



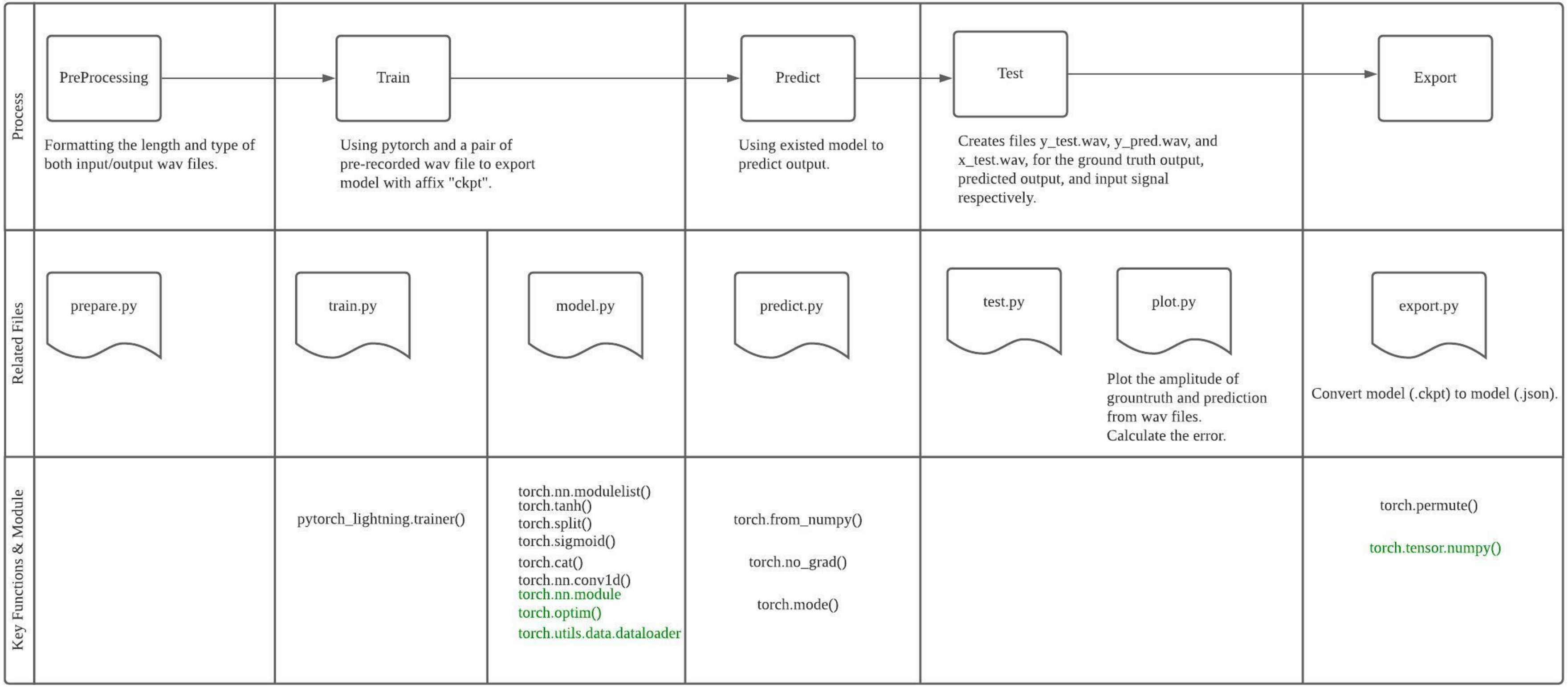
Smart Guitar Amp Integration - Concept



Training: PedalNetRT

Processing: SmartGuitarAmp

Smart Guitar Amp Training Pipeline



Process

PreProcessing → Train → Predict → Test → Export

PreProcessing: Formatting the length and type of both input/output wav files.

Train: Using pytorch and a pair of pre-recorded wav file to export model with affix "ckpt".

Predict: Using existed model to predict output.

Test: Creates files y_test.wav, y_pred.wav, and x_test.wav, for the ground truth output, predicted output, and input signal respectively.

Export

Related Files

prepare.py

train.py

model.py

predict.py

test.py

plot.py: Plot the amplitude of grountruth and prediction from wav files. Calculate the error.

export.py: Convert model (.ckpt) to model (.json).

Key Functions & Module

train.py: pytorch_lightning.trainer()

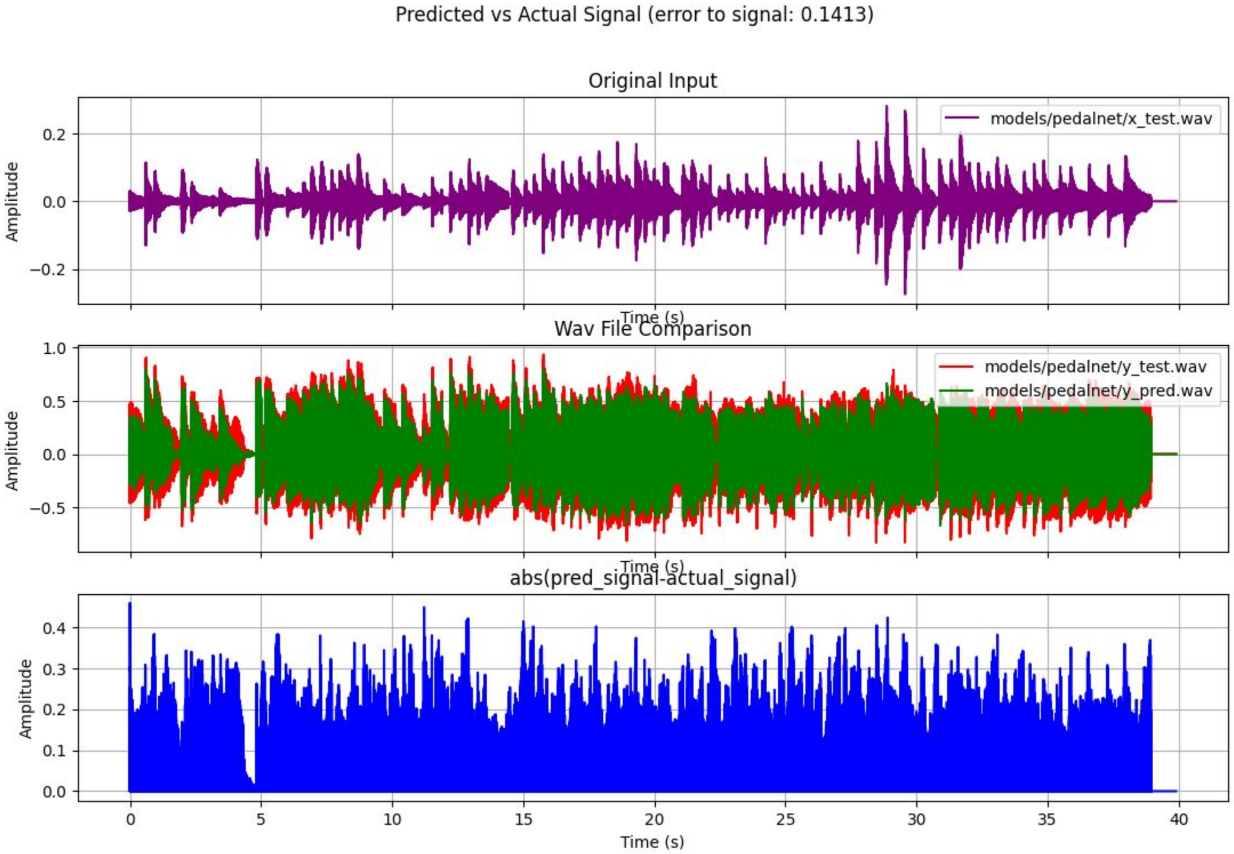
model.py: torch.nn.modulelist()
torch.tanh()
torch.split()
torch.sigmoid()
torch.cat()
torch.nn.conv1d()
torch.nn.module
torch.optim()
torch.utils.data.dataloader

predict.py: torch.from_numpy()
torch.no_grad()
torch.mode()

export.py: torch.permute()
torch.tensor.numpy()

Time (s)

Smart Guitar Amp Training Results



Proposed WaveNet-style neural network model

Smart Guitar Amp - Trained Models

Clean - 76 RC-120 (Default)

Glassy - 67 Blackface Duo

Blues - American Bass

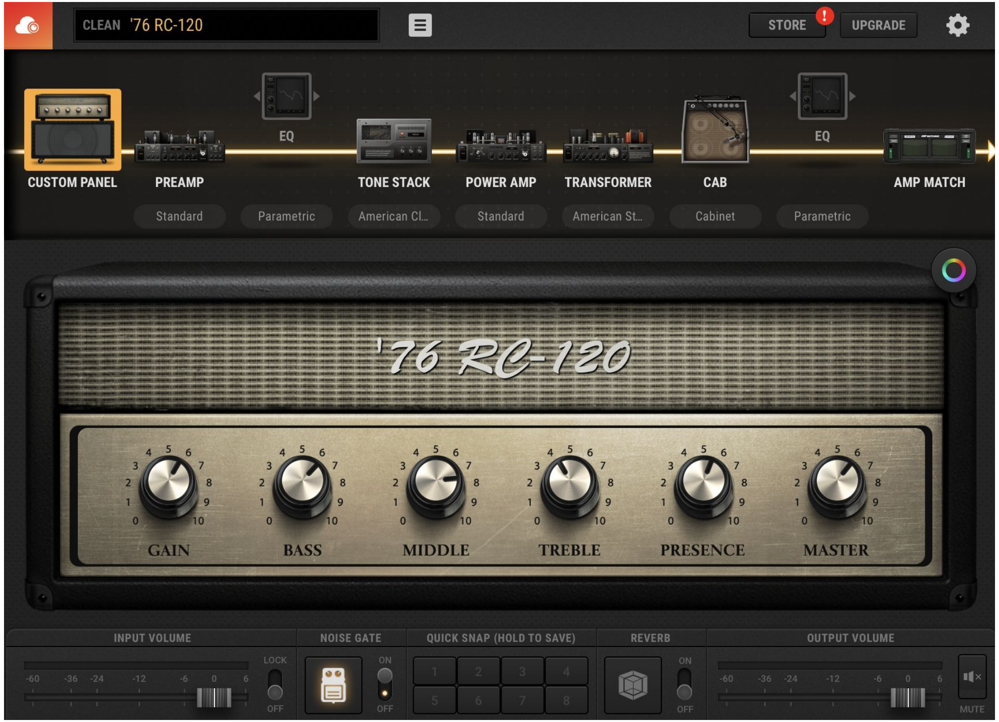
Crunch - British Plexi 50w

High Gain - British Rock 50

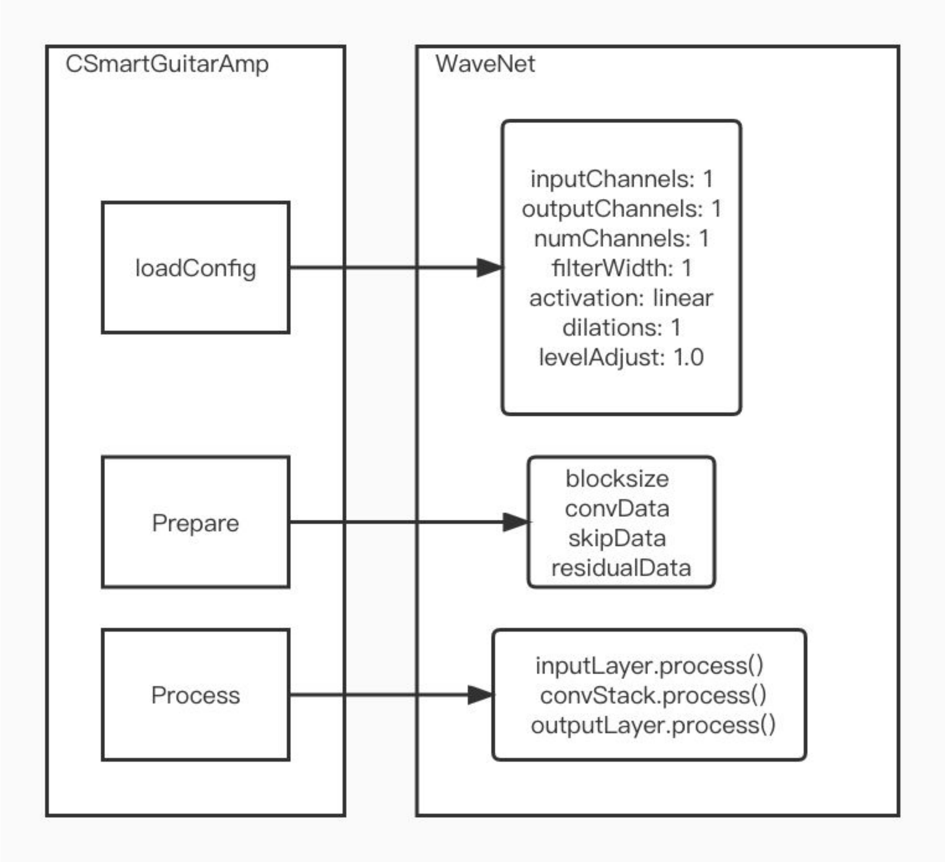
Metal - 5153 MK II

Insane - Fire

Acoustic - Acoustic Sim



Smart Guitar Amp - Processing



CSmartGuitarAmp

WaveNet

loadConfig

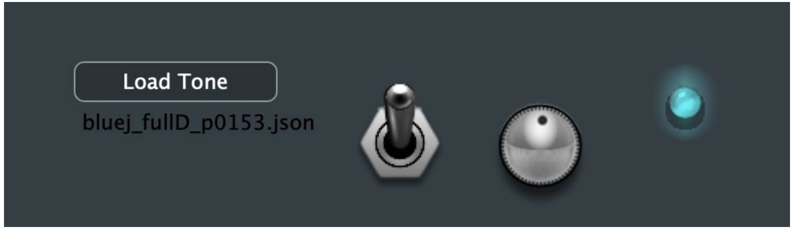
inputChannels: 1
outputChannels: 1
numChannels: 1
filterWidth: 1
activation: linear
dilations: 1
levelAdjust: 1.0

Prepare

blocksize
convData
skipData
residualData

Process

inputLayer.process()
convStack.process()
outputLayer.process()



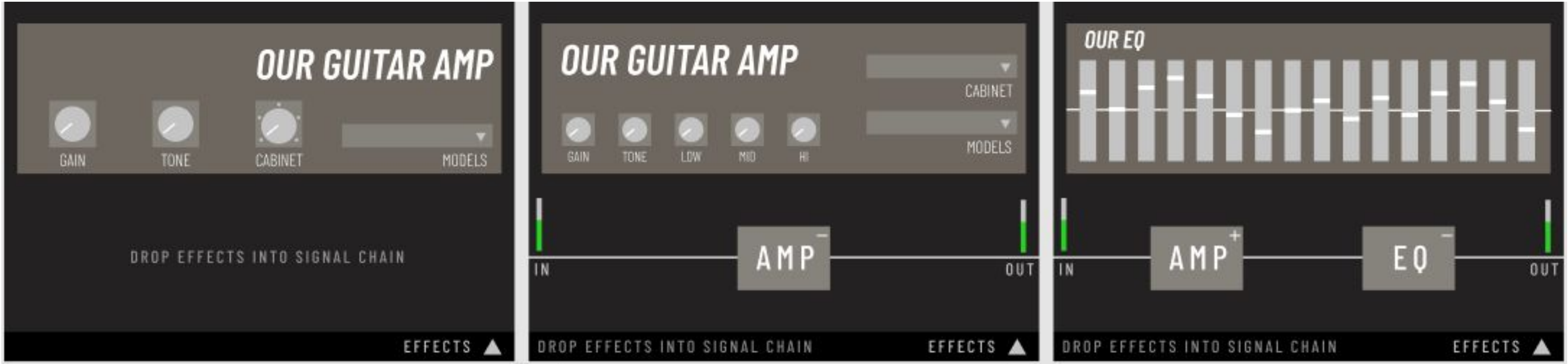
bluej_fullD_p0153.json

Cabinet Simulation

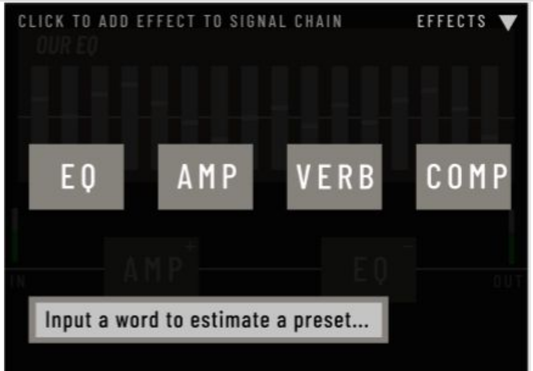
- Convolutional node capable of loading impulse responses of real guitar speaker cabinets

UI Design

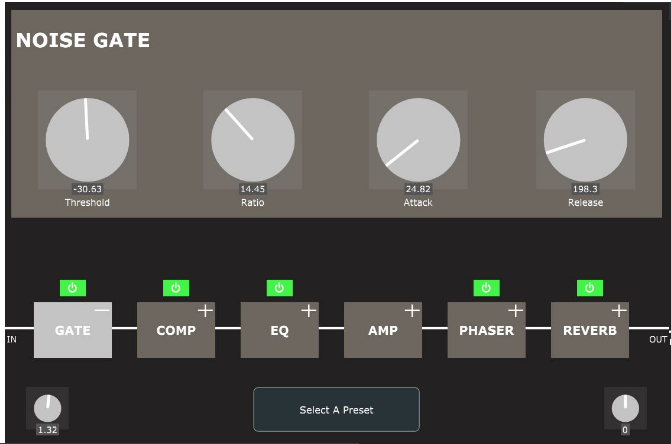
- Effects can have a variable signal chain (drag and drop functionality)



Word-based preset generation!


CLICK TO ADD EFFECT TO SIGNAL CHAIN
EFFECTS
OUR EQ
EQ
AMP
VERB
COMP
IN
AMP
EQ
OUT
Input a word to estimate a preset...

UI Implementation



NOISE GATE

-30.63 Threshold

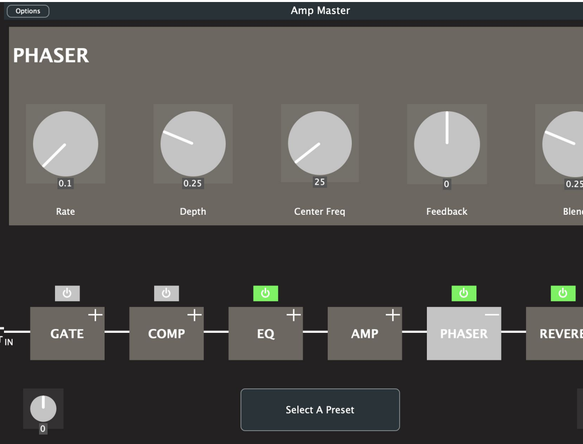
14.45 Ratio

24.82 Attack

198.3 Release

1.32

0



PHASER

0.1 Rate

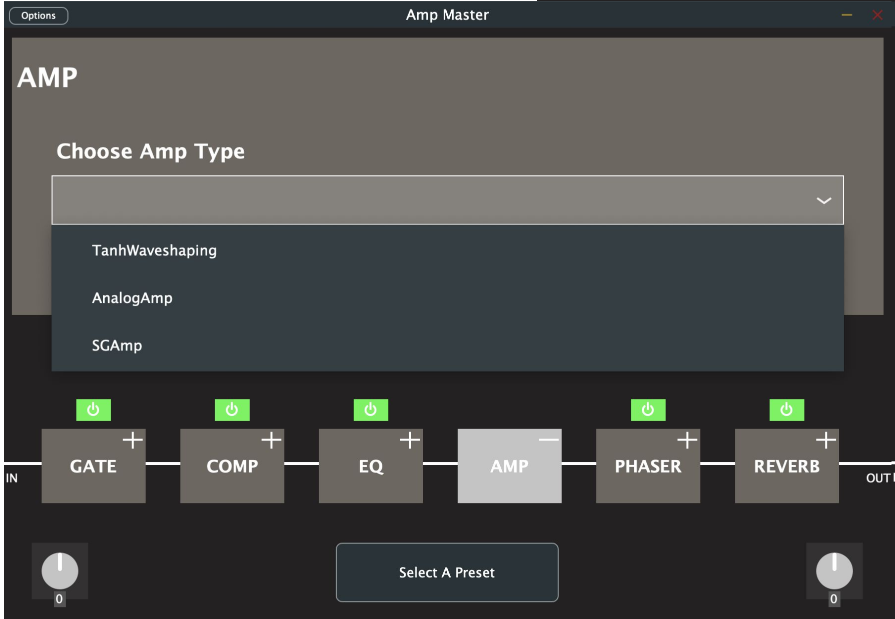
0.25 Depth

25 Center Freq

0 Feedback

0.25 Blen

0

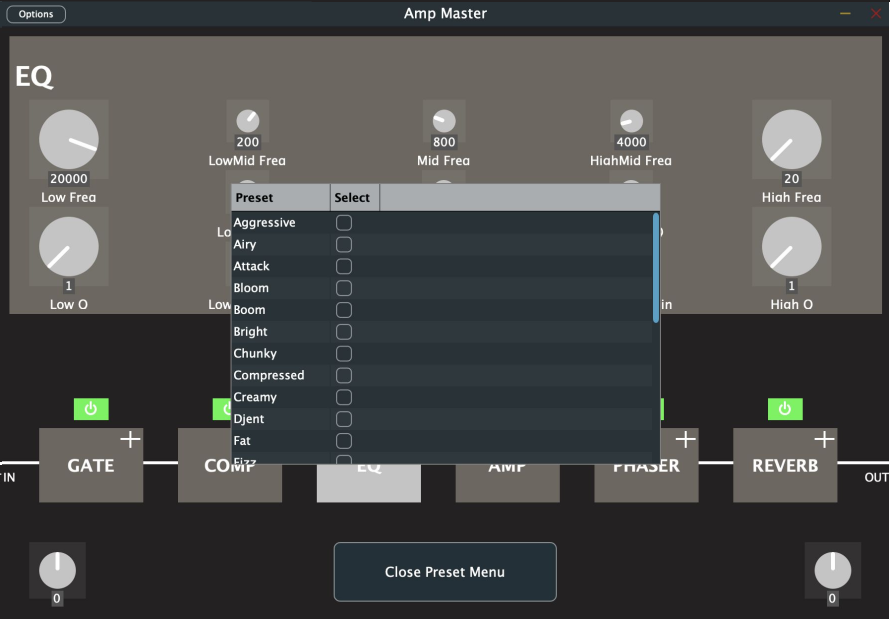


AMP

Choose Amp Type

0

0



EQ

20000 Low Freq

200 LowMid Freq

800 Mid Freq

4000 HighMid Freq

20 High Freq

1 Low Q

1 High Q

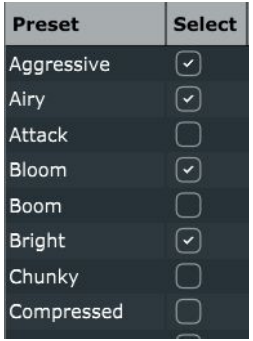
0

0

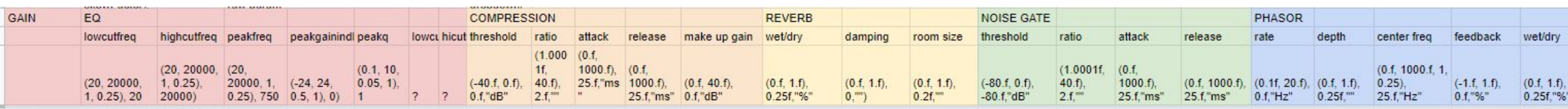
Preset	Select
Aggressive	□
Airy	□
Attack	□
Bloom	□
Boom	□
Bright	□
Chunky	□
Compressed	□
Creamy	□
Djent	□
Fat	□
Fizz	□

Tone-Matching Effect Preset

- Word-based stacking presets
- Words from Seymour Duncan's "Dictionary of Tone Terms"
- Implemented with XML files holding the parameter values and our own averaging functions to combine each preset
- Ex: "mushy" + "muddy" + "flutey" + "growl" = a new preset that should combine all aspects of each tone!
- 26 presets setting 41 different parameters in the tree



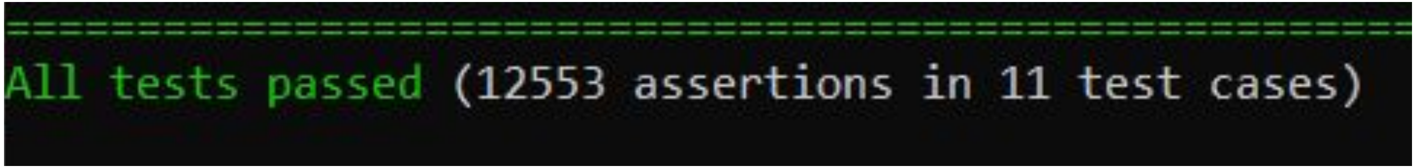
Preset	Select
Aggressive	☑
Airy	☑
Attack	□
Bloom	☑
Boom	□
Bright	☑
Chunky	□
Compressed	□



GAIN	EQ							COMPRESSION					REVERB			NOISE GATE				PHASOR				
	lowcutfreq	highcutfreq	peakfreq	peakgainindl	peakq	lowcı	hicut	threshold	ratio	attack	release	make up gain	wet/dry	damping	room size	threshold	ratio	attack	release	rate	depth	center freq	feedback	wet/dry
	(20, 20000, 1, 0.25), 20	(20, 20000, 1, 0.25), 20000)	(20, 20000, 1, 0.25), 750	(-24, 24, 0.5, 1), 0)	(0.1, 10, 0.05, 1), 1	?	?	(-40.f, 0.f), 0.f,"dB"	(1.0001f, 40.f), 2.f,""	(0.f, 1000.f), 25.f,"ms"	(0.f, 1000.f), 25.f,"ms"	(0.f, 40.f), 0.f,"dB"	(0.f, 1.f), 0.25f,"%"	(0.f, 1.f), 0,"")	(0.f, 1.f), 0.2f,""	(-80.f, 0.f), -80.f,"dB"	(1.0001f, 40.f), 2.f,""	(0.f, 1000.f), 25.f,"ms"	(0.f, 1000.f), 25.f,"ms"	(0.1f, 20.f), 0.f,"Hz"	(0.f, 1.f), 0.25f,""	(0.f, 1000.f, 1, 0.25), 25.f,"Hz"	(-1.f, 1.f), 0.f,"%"	(0.f, 1.f), 0.25f,"%

Automated Testing

- JUCE tests were confusing so we used Catch2 for test framework
- Using juce::Value for parameter setting
- Tests include:
  - Check for clipping for each parameter
  - Set and check min and max for all params
  - Effect bypassing and check overall gain for each node
  - APVTS node instantiation
  - Latency
  - Mono and stereo support
  - Etc!



```
======================================================
All tests passed (12553 assertions in 11 test cases)
```

Roadblocks

- Learning to use JUCE ProcessorGraph
- ML in C++, too complex to implement for auto-generated presets
- Integrating each part of the application together
- Lack of documentation of testing in JUCE
- Multi-select dropdown for preset selection
- Cmake and linker issues with external libraries

Timeline

- 2/13 - Plug-in framework, distortion and tone, GUI design
- 2/28 - Amp and FX components with standard GUI; ML decisions
- 3/7 - Work through class structure, create component base
- 3/14 - SmartGuitarAmp running individually
- 3/14 - Initial connection of components with standard GUI
- 3/14 - Start writing tests
- 3/21 - Creating database of effect presets
- 3/28 - Begin implementing new GUI
- 4/3 - Finish creating effect presets, start refactoring AudioGraph
- 4/11 - Finish refactoring the Graph and nodes
- 4/18 - Connecting SmartGuitarAmp and Cab Sim
- 4/25 - Finish tests and SmartGuitarAmp
- 5/3 - Finish GUI, connect presets, final bug fixes!