WAVENET-STYLE GUITAR AMPLIFIER MODEL PRUNING FOR REAL-TIME IOS DEPLOYMENT

Ryota Sato and Eli Silverstein

Dept. of Electrical Engineering
Stanford University
Stanford, USA
ryos17@stanford.edu
esilvers@stanford.edu

ABSTRACT

WaveNet-style convolutional networks emulate tube amplifiers and distortion pedals with high fidelity, but their computational cost has confined them to desktops or dedicated DSP hardware. We present a sparse-enabled WaveNet inference engine for iOS that runs heavily pruned neural guitar amplifier models in real time on iPhones. Aggressive iterative magnitude pruning removes 90% of the network weights with no perceptible loss in quality. A custom sparse C++ engine turns this sparsity directly into compute savings, sustaining low-latency real-time operation on a CPU-only iPhone implementation where the dense model cannot. On-device output matches the trained model to within int16 quantization error. At the demonstration, visitors will play a guitar through the app on iPhone hardware and A/B the on-device pruned model against the physical pedal it emulates. Source code and audio examples are available at `https://github.com/ryos17/wavenet-imp`.

1. INTRODUCTION

Virtual analog (VA) modeling of audio circuits has become an active research area, particularly for guitar amplifiers and effects [1]. White-box methods explicitly simulate the underlying circuit [2] but require detailed component knowledge, whereas black-box methods learn the nonlinear input–output relationship directly from measured data. Neural networks are especially effective here, with early approaches using Long Short-Term Memory (LSTM) recurrent networks [3, 4] and state-of-the-art methods using WaveNet-style convolutional networks with dilated one-dimensional convolutions [5].

Such models have traditionally been deployed on desktop machines or dedicated hardware [6] because of their high computational cost. To improve efficiency, prior work has turned to iterative magnitude pruning [7], which removes a large fraction of weights while preserving accuracy, as shown for LSTM-based neural amp models [8]. Such sparse subnetworks are supported by the lottery ticket hypothesis [9] and confirmed for convolutional audio models by Esling et al. [10]. However, systematic pruning of WaveNet-style neural amplifiers remains largely unexplored, and little effort has targeted sparsity-aware inference engines for neural amplifier modeling on portable consumer devices such as the iPhone.

This work demonstrates the real-time deployment of a heavily pruned WaveNet-style network for guitar amplifier emulation on a CPU-only iOS implementation. Section 2 details the training and pruning strategy, and Section 3 the real-time iOS C++ inference engine and its on-device performance. At the demonstration, attendees will play through the on-device model and compare it against the original analog hardware.

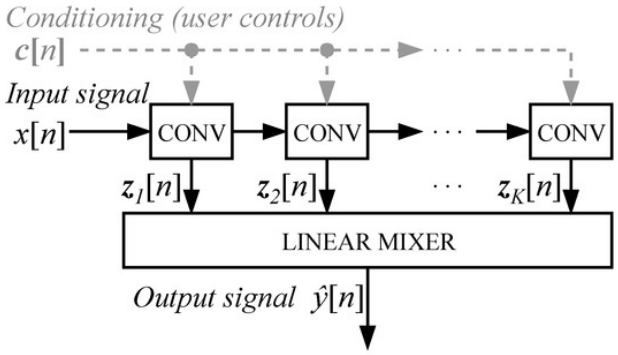

Figure 1: WaveNet-style architecture used for neural guitar amplifier modeling, reproduced from [5].

2. MODEL TRAINING AND PRUNING

2.1. Architecture

We adopted a variant of the original WaveNet architecture [11], a causal feedforward network composed of stacked dilated convolutional residual blocks (Figure 1). The network learned a direct mapping from the raw input guitar waveform $x[n]$ to the corresponding distorted output $\hat{y}[n]$. Dilation expanded the receptive field of each convolution without increasing the number of parameters per layer, allowing the network to capture long temporal dependencies. Each block applied a learned causal FIR filter followed by a nonlinearity, and skip connections aggregated intermediate representations into the final output. We followed the neural-amplifier configuration of Wright et al. [5]: a channel dimension $C = 16$, a kernel size of 3, 1500 training epochs, and an 18-layer dilation pattern $d_k = \{1, 2, 4, \ldots, 256, 1, \ldots, 256\}$. Unlike the original architecture, we did not employ gated $\tanh$ activations, instead using standard $\tanh$ nonlinearities to reduce computational overhead in the later C++ deployment. Grounding the network size and topology in this established baseline lets the reported sparsity be interpreted relative to a known reference.

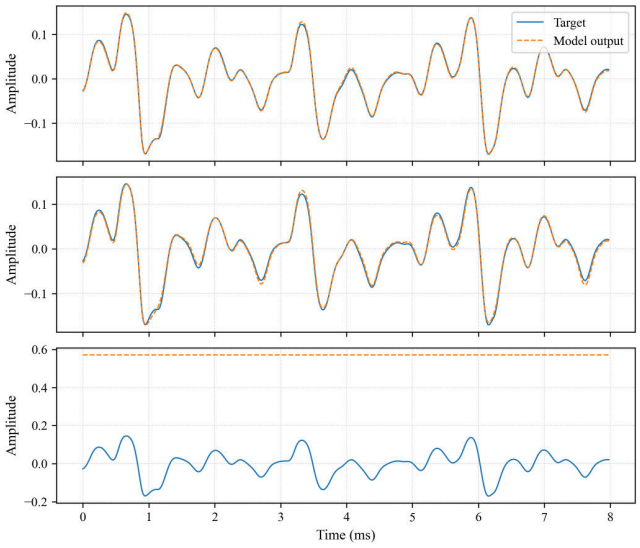

Figure 2: Output waveforms for a held-out test segment. **Top:** unpruned baseline. **Middle:** 90% iterative pruning, which tracks the target with only minor deviation. **Bottom:** 90% one-shot pruning, which fails to track the target.

2.2. Training Objective

The model was trained by minimizing the mean squared error (MSE),

$$\mathrm{MSE} = \frac{1}{N}\sum_{n=0}^{N-1}\left|y_p[n] - \hat{y}_p[n]\right|^2, \tag{1}$$

where $y_p[n]$ and $\hat{y}_p[n]$ denote the pre-emphasized target signal and model output, respectively. Pre-emphasis was applied using the first-order infinite impulse response (IIR) high-pass filter

$$H(z) = 1 - 0.95\,z^{-1}, \tag{2}$$

which is commonly used in speech processing and neural amplifier modeling [12]. Although prior work [5] adopted the error-to-signal ratio (ESR),

$$\mathcal{E}_{\mathrm{ESR}} = \frac{\sum_{n=0}^{N-1}\left|y_p[n] - \hat{y}_p[n]\right|^2}{\sum_{n=0}^{N-1}\left|y_p[n]\right|^2}, \tag{3}$$

as the training loss, our preliminary experiments showed that MSE led to faster and more stable convergence during training. Despite the effectiveness of MSE training, we used ESR as the evaluation metric, since normalizing the error by the signal energy makes it more suitable for comparing models with different output loudness levels.

2.3. Pruning Strategy

To aggressively sparsify the network, we applied iterative local magnitude pruning. Sparsity is defined as the ratio of pruned parameters to the total number of prunable parameters, with biases and other non-prunable parameters excluded. For reference, the 16-channel model contained 21,913 total parameters, of which 21,152 were prunable. At 90% sparsity, 19,074 prunable weights were removed.

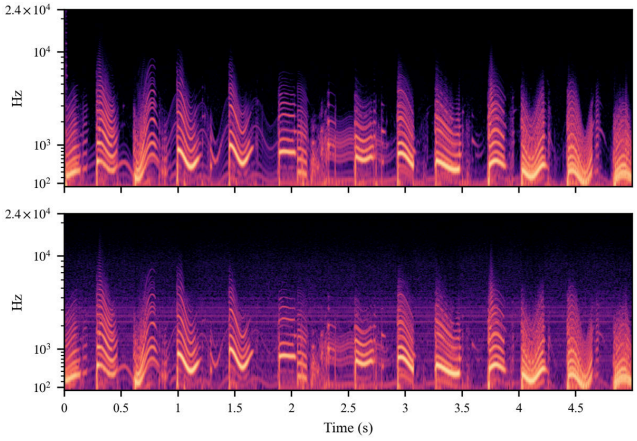

Figure 3: Fuzz Face at maximum distortion: 90% pruned model output (**top**) versus the recorded target (**bottom**). The deterministic model omits the input-uncorrelated noise and hum visible in the target.

Each prunable weight tensor W was associated with a binary mask M, and the layer used $W \odot M$ in both the forward and backward passes. Rather than pruning once after training (one-shot pruning), the mask was updated every mini-batch according to a sparsity schedule, letting the network adapt to sparsity during training. The sparsity s was ramped from 0 to the target s_t between epochs e_{start} and e_{end} using either a linear or an exponential schedule.

We evaluated all four combinations of local versus global magnitude pruning and linear versus exponential schedules. Fixing $e_{\mathrm{start}} = 10$ and sweeping $e_{\mathrm{end}} \in \{250, 500, 750, 1000, 1250\}$ on a validation split, $e_{\mathrm{end}} = 750$ achieved the lowest validation ESR. Overall, iterative local pruning with an exponential schedule performed best and was used in all subsequent experiments. Iterative pruning preserved accuracy up to high sparsity, whereas one-shot pruning collapsed well before 90%, as the waveform comparison in Figure 2 shows.

2.4. Modeling Quality

We fixed a target of 90% sparsity, a conservative real-time operating point justified in Section 3.3. At this sparsity level, the model reproduced our in-house captures of four guitar distortion sources:

- Vox AC15
- Fender Deluxe Reverb
- Fender Tweed-style amplifier
- Dunlop Fuzz Face pedal

recorded with sE Microphones using a structured three-minute excitation signal [13]. All ESR values were below 3.4×10^{-4} with no audible degradation in informal listening. Across captures, the long-reverb settings produced the largest errors, since their tails exceeded the model's receptive field, whereas the directly recorded Fuzz Face attained the lowest ESR despite its strong nonlinearity.

Figure 2 contrasts the output waveforms directly: the 90% iteratively pruned model tracks the target closely, whereas naive one-shot pruning at the same sparsity fails to track the target waveform. The main residual limitation is that the deterministic model

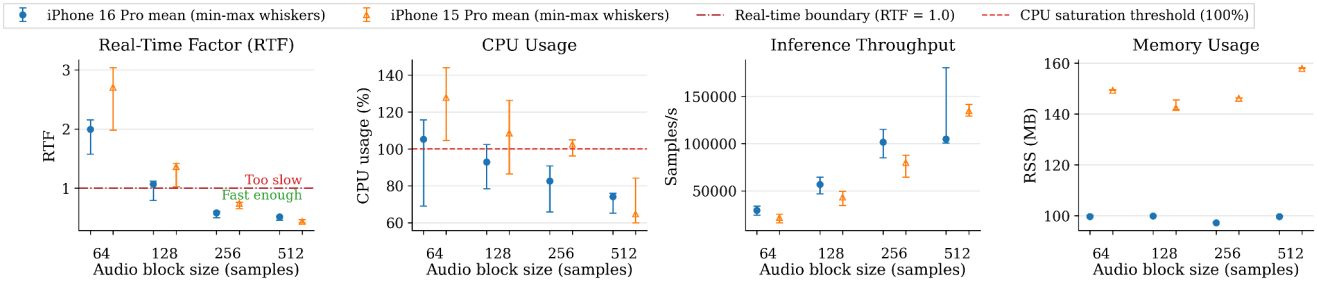

Figure 4: Block-size sweep at 90% sparsity: iPhone 16 Pro versus iPhone 15 Pro, showing RTF, CPU usage, inference throughput, and memory usage. Measurements taken from 60 seconds of audio for each block size.

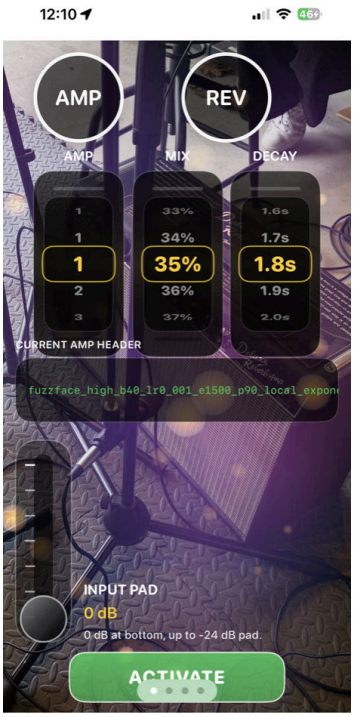

Figure 5: iOS application interface: amplifier/pedal selection, input attenuation, and reverb mix and tail-length controls.

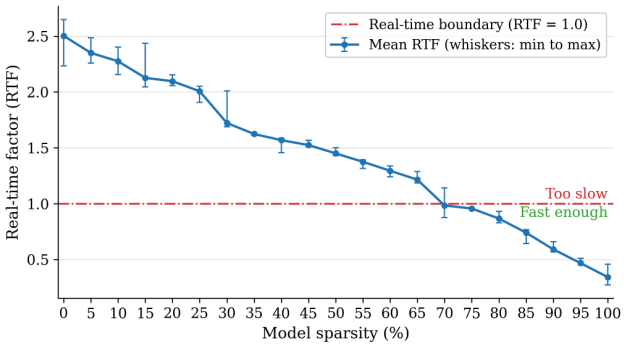

Figure 6: Real-time factor versus sparsity on an iPhone 16 Pro (block size 256 samples, 48 kHz). Points show the mean RTF over 10 s of audio; whiskers indicate the minimum and maximum. The dense model is intractable, whereas the 90% pruned model runs with comfortable margin.

cannot reproduce input-uncorrelated noise and hum, which is most pronounced for the high-gain Fuzz Face as seen in Figure 3.

3. REAL-TIME IOS DEPLOYMENT

3.1. Sparse Inference Engine

Inference runs entirely on the CPU. Although the GPU (Metal) and Neural Engine (Core ML) offer higher throughput and better power efficiency, the per-block dispatch latency of their public interfaces—together with the absence of low-level Neural Engine access—makes them impractical for the small, per-audio-block workloads of real-time inference at a 256-sample block size.

Audio is processed in fixed-size blocks at a 48 kHz sampling rate. The block size is configurable from 64 to 512 samples, with 256 samples (≈5.3 ms) adopted as the default operating point. Crucially, the engine stores and iterates over *only the nonzero weights*, kept in a compact, cache-friendly layout, rather than masking a dense kernel, so the unstructured sparsity translates directly into compute savings. For this reason we implemented the dilated-convolution stack manually rather than relying on a general-purpose inference library such as RTNeural [14], which operates on dense kernels and would not realize the same speedup.

File input can be passed through the C++ inference engine, so deterministic validation against the Python reference is possible. The inference engine inputs and outputs fixed-point audio sample values but converts to floating-point for processing. Against a Python reference using the same exported weights, the C++ engine matched to within the int16/float conversion quantization after engine startup.

3.2. Interactive Application

The pruned amplifier was embedded in an interactive iOS application (Figure 5) providing a selector for the captured amplifiers/pedals and an input-attenuation control (0 to −24 dB) for loudness matching across modeled devices. The app also provides a convolutional reverb implemented as partitioned overlap-add convolution with a measured impulse response, exposing wet/dry-mix

and tail-length controls. The reverb only accounts for $\approx 3\%$ of per-block DSP time.

3.3. On-Device Performance

We define the real-time factor (RTF) as the ratio of audio-processing time to the duration of audio produced. Thus, RTF < 1 indicates that the engine keeps up with playback. In Figure 6 measured on an iPhone 16 Pro at a 256-sample block size, RTF decreased approximately linearly with sparsity. The dense model sat well above the real-time threshold and was intractable, about 70% sparsity marked the boundary into real-time operation, and the 90% point left a comfortable margin (RTF ≈ 0.6). This is the central justification for the demonstration, where pruning is what makes an otherwise intractable WaveNet-style model run on the phone.

As seen in Figure 4, sweeping the block size across $\{64, 128, 256, 512\}$ samples on both an iPhone 16 Pro and an iPhone 15 Pro revealed that 256 samples was the smallest block that sustained real time on the 16 Pro. Older devices required a larger block or higher sparsity.

4. DEMONSTRATION

For the live demonstration attendees will play, or listen to, an electric guitar connected to an iPhone running the neural amp model and listen through headphones. They will switch freely among the captured amplifiers and pedals and adjust the input attenuation and reverb in real time. A Line 6 HX Stomp will route the signal between the on-phone pruned Fuzz Face model and the physical Fuzz Face pedal, letting attendees compare the on-device emulation directly against the analog hardware the model was trained on.

5. CONCLUSION

This work shows that heavy iterative magnitude pruning (90% sparsity), paired with a sparse-aware iOS inference engine, lets a WaveNet-style neural guitar amplifier run in real time on a CPU-only iPhone implementation where the dense model cannot, with no perceptible loss in modeling quality, accurate to within int16 quantization error. Future work includes vectorizing the dilated-convolution stack and pointwise $\tanh$ with Apple's Accelerate/vDSP framework, profiling across a wider range of devices, and a formal perceptual evaluation.

6. ACKNOWLEDGMENTS

This project grew out of the final project for *EE264W: Digital Signal Processing* in the Stanford Department of Electrical Engineering, whose real-time iOS audio framework was developed by Fernando Mujica. Many thanks to Professor Fernando Mujica and Ron Schafer for their guidance throughout the project.

7. REFERENCES

[1] V. Välimäki, S. Bilbao, J. O. Smith, J. S. Abel, J. Pakarinen, and D. Berners, *DAFx: Digital Audio Effects*. John Wiley & Sons, Ltd, 2011, ch. 12: Virtual Analog Effects, pp. 473–522. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119991298.ch12

[2] M. Karjalainen and J. Pakarinen, "Wave digital simulation of a vacuum-tube amplifier," in *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, vol. 5, 2006, pp. V–V.

[3] J. Covert and D. L. Livingston, "A vacuum-tube guitar amplifier model using a recurrent neural network," in *2013 Proceedings of IEEE Southeastcon*, 2013, pp. 1–5.

[4] Z. Zhang, E. Olbrych, J. Bruchalski, T. J. McCormick, and D. L. Livingston, "A vacuum-tube guitar amplifier model using long/short-term memory networks," in *SoutheastCon 2018*, 2018, pp. 1–5.

[5] 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. [Online]. Available: https://www.mdpi.com/2076-3417/10/3/766

[6] M. Kokaly-Bannourah and D. Castro, "Quad cortex magic sound box powered by sharc+ dsps," https://www.analog.com/en/lp/001/customer-success-stories-quad-cortex.html.

[7] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Advances in Neural Information Processing Systems*, vol. 28. Curran Associates, Inc., 2015, pp. 1135–1143.

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

[9] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Int. Conf. on Learning Representations (ICLR)*, 2019.

[10] P. Esling, N. Devis, A. Bitton, A. Caillon, A. Chemla-Romeu-Santos, and C. Douwes, "Diet deep generative audio models with structured lottery," in *Proc. 23rd Int. Conf. Digital Audio Effects (DAFx-20)*, Vienna, Austria, 2020.

[11] 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," in *9th ISCA Workshop on Speech Synthesis (SSW 9)*, 2016, p. 125.

[12] E.-P. Damskägg, L. Juvela, E. Thuillier, and V. Välimäki, "Deep learning for tube amplifier emulation," in *ICASSP 2019 – 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 471–475.

[13] S. Atkinson, "Neural amp modeler," https://www.neuralampmodeler.com/.

[14] J. Chowdhury, "RTNeural: Fast neural inferencing for real-time systems," 2021, arXiv:2106.03037.