Meta-learning for digital audio effects

Ivan Villa-Renteria, Yu Shen Lu, and Dirk Roosenburg

Abstract

Neural networks that directly emulate audio effects take a large amount of time and data to train for a single audio effect. We postulate that there exists a substantial amount of shared structure between different audio effects that can be exploited by meta-learning methods in order to learn multiple audio effects much more efficiently. This paper focuses on the implementation of MAML to exploit this shared structure and meta-learn these audio effects without access to the audio-effect parameters used to transform audio signals. We trained TCN, LSTM, and Transformer models using MAML on generated audio data in order to learn the audio effects applied.

Our dataset consists of 10 minutes of generated audio at 48 kHz sample rate. Audio samples were obtained from Freesound, utilizing a diverse number of sources such as noise-perturbed sine sweeps, speech, piano, guitar, and other sources. These samples were then fed to `DAWDreamer`, a Python-based digital audio workstation, to apply audio effects and generate target audio samples.

We experimented by using MAML with three model architectures; LSTM, TCN, and Transformer. The Transformer and the LSTM were implemented by leveraging Facebook Research's `higher` library for reliable computation of higher-order gradients, while the TCN was implemented using PyTorch's `functional` API. For each task, we sample blocks of input and output audio with the same audio effect parameters. We then show the model input and target support audio in the inner-loop and ask it to recreate the audio effect on the query input in the outer-loop. Using MAML, we update the model to obtain better starting meta-parameters. We hypothesize that since there is shared structure between different tasks, our model can improve by generalizing over different tasks.

We observe that LSTM models can achieve close to state-of-art performance in L1 loss. The TCN model struggled with learning, although its runtime and memory usage is still better than much smaller LSTM models. Moreover, our transformer achieved higher loss values than our best TCN and LSTM models. We find that the transformer model learns to output flat signals in an effort to minimize L1 Loss as opposed to learning the audio effects that are applied to the input audio. This is surprising since TCN and Transformers are both popular audio effect model candidates, but both performs sub-optimally in meta-learning settings. In addition, we see fluctuation in validation loss during all model training processes. This shows that MAML might be too unstable for training recursive model with long (>4000 time steps) input sequence.

Despite findings that show our models improve after adaptation, we are unable to consistently obtain results comparable to the literature for audio effect simulation; this gap is especially evident in the frequency domain, where all of our models struggle to remove high-frequency information from the input audio. We postulate that this is because audio is inherently both long time series and information-dense which makes meta-learning training challenging. The only model that shows promise is the recurrent model (LSTM), but the sequential nature of the model makes training extremely resource-intensive; thus, we were unable to explore and experiment more extensively due to these constraints.

Thus, we conclude that most ML models used for modeling audio effects seem to be unsuitable when directly implemented as part of MAML algorithm. Although we observe our models benefit from meta-learning, we are unable to achieve performance similar to models trained on single task.

1 Introduction

The popularization and ubiquity of the digital audio workstation (DAW) has created a new demand for high quality digital audio effects. Today, producers will often never touch a piece of analog audio gear as they create entire songs and albums entirely within the digital world. High quality emulations of that same analog audio gear used are some of the most popular audio effects used today. Previously these audio effects have been created using custom, hand-tuned algorighims. Recently, machine learning methods have been shown to be effective at creating digital models of individual analog audio effects [16, 9, 6].

Effective deep learning of audio effects has the potential to create models of analog audio equipment that cannot be reproduced using mathematical methods. Much of this equipment is hard to find, impossible to reconstruct, and prohibitively expensive to own.

Furthermore, recent interest for machine learning methods in audio has also created a demand for differentiable audio effects. Automated multitrack mixing, for example, requires some differentiable audio effect that can be computed as part of the loss of a complete multitrack mixing system. One potential solution to this is to use a neural proxy to quickly learn an audio effect and be used in place of it during training of the multitrack system [12]. This neural proxy must be accurate, quick to train, and able to emulate many audio effects.

In this experiment, we investigate modeling an audio effect as a meta-learning problem, where an individual audio effect corresponds to an individual task. Because there is significant shared structure between similar audio effects, this structure can be exploited by meta learning techniques to avoid redundant work by creating a more general model.

This work is directly related to previous research by one of the group members, recently published as part of [3]. It seeks to build on work shown in [8].

2 Background

2.1 Digital Audio Filters

Analog audio signal processing uses analog circuits to process continuous time signals. However, the ubiquity of computers beginning in the eighties created demand for digital signal processing. To convert a continuous time audio signal into a digital one, the signal is represented as some discrete-time series x_n, where n is the "sample index". Audio is sampled at a sample rate f_s creating a sample interval $T = 1/f_s$, the elapsed time between each sample. Consequently, the digital signal is an approximation, and can only correctly reconstruct the part of the analog signal with frequency below $2/f_s$, the Nyquist Rate.

Audio filters process these time-series data, often in real-time. Filters can grouped into "causal filters" and "implicit filters". A causal filter, $f_c(\cdot)$, produces some output, y_n, that is computed using only previous or current inputs $x_{0:n}$ and previous outputs $y_{0:n-1}$. It can be expressed as

$$f_c\left(y_n | x_{0:n}, y_{0:n-1}, \phi\right) \tag{1}$$

where ϕ is some set of parameters of the filter. This is a restriction used to enforce real-time realizibility; if an algorithm is running in real-time, it cannot use a future sample to create the current output because that future sample has not yet been created. Implicit filters do not have this limitation and compute the output of a sample based on any time series data.

2.2 Machine Learning Models for Audio Filters

Currently, most analog audio filters are designed and tuned by humans to create a desired effect. However, many are currently exploring the direct learning of various audio effects. This is a difficult task for several reasons. First, the use of feedback in many digital audio filters creates systems that are challenging to parallelize, leading to longer training times and high memory requirements. Furthermore, audio effects are usually expected to run in real-time, limiting the sizes of the models that can be used. Finally, the audio itself is a very information dense format that can be challenging for machine learning models to understand and reconstruct.

Much of the literature has focused on exploring different potential architectures for learning audio effects. Many papers have focused on using a feedforward variant of WaveNet [7] to learn an audio effect [2]. Custom structures, such as the recurrent structure based on state space systems presented in [8]. Temporal-convolutions networks (TCNs) have shown some success [13, 10]. Recurrent neural networks, specifically LSTMs are effective [16] but require comparatively much more computational power to train.

A secondary challenge when training audio machine learning algorithms is the evaluating the model output in comparison to the label or target. This stems from the issue that there is not agreed upon model of human perception of similarity or dissimilarity of two audio sources. Thus, there is no decided upon evaluation metric. In [17] they propose a metric using a combination of error-to-signal ratio (ESR) and DC signal offset loss applied to audio run through a A-weighted pre-emphasis filter. The authors of [11] show that, in some cases, mean-absolute error (MAE) or L_1 loss, may be another good metric, yielding strong results. They also propose an evaluation metric that uses the aggregate of MAE and the multi-resolution short-time Fourier transform loss (MRSTFT). [5] proposes learning a differentiable audio metric from human labeled audio data. Other popular metrics include mean-squared error [1] or other short-time Fourier transform (STFT) based methods. Because so many different metrics are used, direct comparison of results of between experiments can be challenging.

2.3 Model-Agnostic Meta Learning

It is reasonable to expect that different types of audio processing presumably require similar underlying structures; for example, two effects that both perform linear filtering should have somewhat similar ϕ_i. In [4], it is shown that adaptation of different speaker and accent can be achieved efficiently with meta learning, we believe that the idea of audio style transfer can be similarly adapt to our objective and simulate different audio effects.

3 Method

We plan to frame audio effect/filter as the tasks that our model will train to perform. Our neural net should learn to directly infer some time-series output signal y_n given a time-series input signal x_n.

Ideally, each audio effect corresponds to a single task, T_i; at meta-test time, the model would be able to replicate the audio effect. It should be noted that an audio effect includes variable parameter control that changes the behavior of the audio effect. Initially, we will consider each parameter possibility to be a different audio effect or task. How each parameterized audio effect is implemented is discussed in further detail in section 4.1.

Ultimately, we'd like the final model to be able to infer the total audio effect based on a task that samples audio from several parameter positions. Due to time and resource constraint, we will focus on one family of digital audio effect.

3.1 Loss Function

During training we used mean-absolute error (MAE), also called the L_1 loss. MAE is given by

$$\ell_{MAE}(y, \hat{y}) = \frac{\sum_{i=0}^{N-1} \hat{y}_i - y_i}{N}, \tag{2}$$

Where $\hat{y}_n$ is the output signal, y_n is the target signal, and N is the length of the output in samples. We used the `L1Loss` function from Pytorch.

For purposes of comparison during evaluation, we also used the short-time Fourier transform (STFT) loss as described in [11] and implemented in the auraloss package. "STFT loss, ℓ_{STFT}, is defined as the sum of of the spectral convergence, ℓ_{SC}, and spectral log-magnitude ℓ_{SM}.

$$\ell_{SC}(y, \hat{y}) = \frac{||\ |STFT(y)| - |STFT(\hat{y})|\ ||_F}{||\ |STFT(y)|\ ||_F} \tag{3}$$

$$\ell_{SM}(y, \hat{y}) = \frac{1}{N}||\ \log(|STFT(y)|) - \log(|STFT(\hat{y})|)||_1 \tag{4}$$

$$\ell_{STFT}(y, \hat{y}) = \ell_{SC}(y, \hat{y}) + \ell_{SM}(y, \hat{y}), \tag{5}$$

where $||\cdot||_F$ is the Frobenius norm, $||\cdot||_1$ is the L1 norm, and N is the number of STFT frames." [11]

4 Expriment Setup

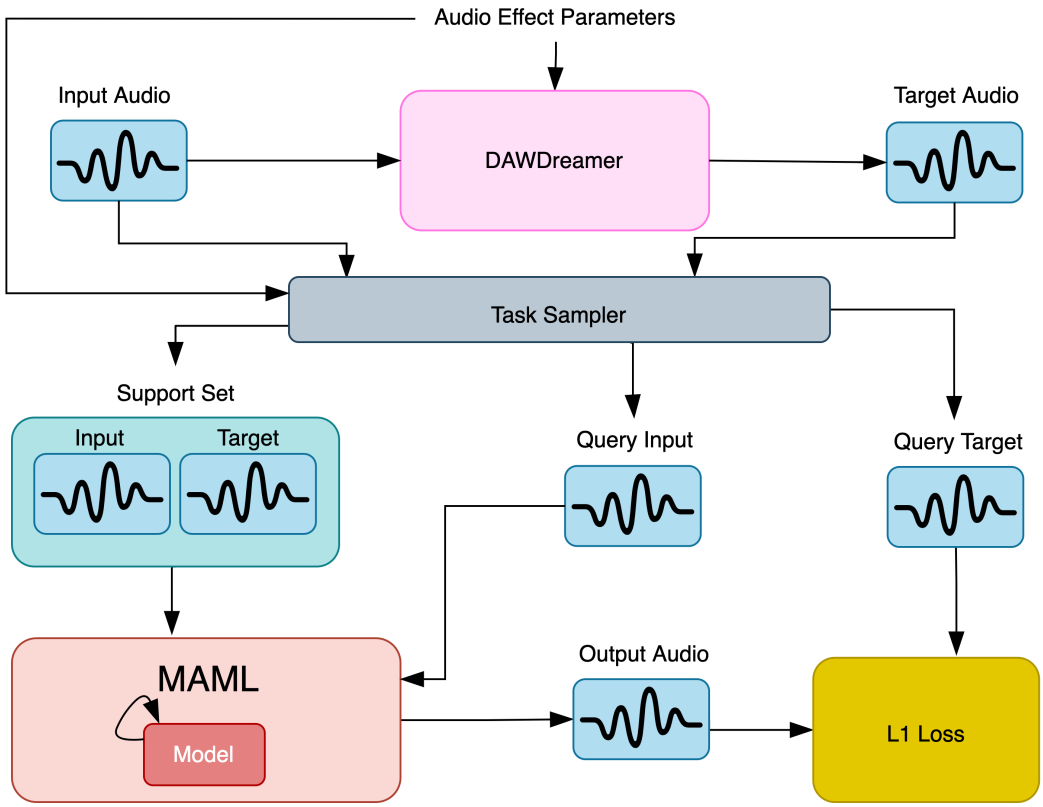


Figure 1: Experiment pipeline

Our full training pipeline is shown here in Figure 1, we generate our dataset by applying a parameterized audio effect to the input audio. For each task, we sameple input and output audio with the same audio effect parameters; we then show the input and the desired output to our audio model. Finally, the model sees the query input and makes the prediction of the query output, and we use MAML framework with to take gradient step for the meta-parameters (parameters in the audio model).

4.1 Dataset Generation

To create a dataset, we implemented an automated task generation method. First, we created a baseline dataset by collecting copyright free audio samples from Freesound and generating custom test functions in Python. Minor processing was done in Ardour to join together audio files and normalize each audio clip to the same level. The dataset consisted of human speech, music, and sine sweeps with white noise; the current dataset is around 10 minutes at a sample rate of 48 kHz. Next, we implemented an audio filter in Faust that emulates the Moog resonant low-pass filter. We parameterize the effect by specifying the cut-off frequency and resonance. Finally, we passed the audio through the filter for a variety of parameter settings, creating different tasks for the meta-learning algorithm to learn. The resulting dataset contains over 100 distinct task classes and can be expanded easily by programming additional effects in Faust.

To sample from this dataset, we first select a random task class. Then, from the task class's audio, we sample $n = K_{support} + K_{query}$ half-second slices of both the input and output audio from the class, where n is the. This constitutes a single task for the neural network.

4.2 Model Architecture

For model implementation we used two main approaches: full functionalization or integration with Facebook Research's `higher` library. We use mean absolute error (MAE/L1 Loss) between the model's predicted audio and the target audio with the a SGD optimizer to make gradient steps in inner loop. Adam optimizer is used in the outer loop.

4.2.1 TCN

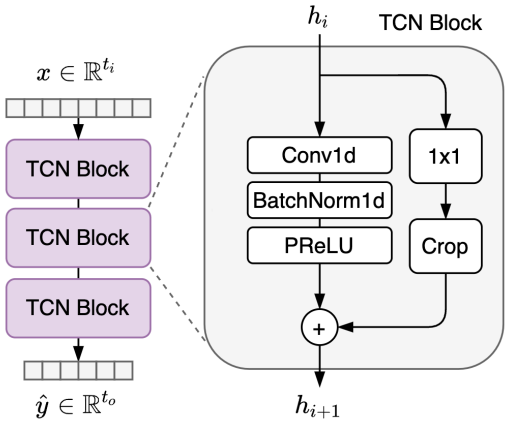


Figure 2: TCN model structure [13], t_i is the input dimension of the audio sample, and t_o is the output dimension of the audio signal. h refers to the hidden size, which varies at each TCN block.

We adapted the TCN-100-N structure from [13] using PyTorch's `functional` API. The TCN model consists of repeated TCN blocks which contains 1D convolution, batch normalization, PReLU and a skip connection. The main difference between this model and the other models we tried is that the output size of this model is smaller than the input size due to the nature of the TCN block's receptive field. As a result, we crop the label to the same size when computing the loss.

4.2.2 LSTM

We implemented two LSTM models (a large one and a small one) via MAML by leveraging the `higher` library's ease-of-use for calculating the higher-order gradients present in MAML's inner loop. Both models make use of an LSTM module, followed by a linear layer. Our large model uses an LSTM module with a hidden state size of 256, followed by a linear layer with 1 output neuron, while the small LSTM model uses a hidden state size of 64 followed by a linear layer of 1 output neuron.

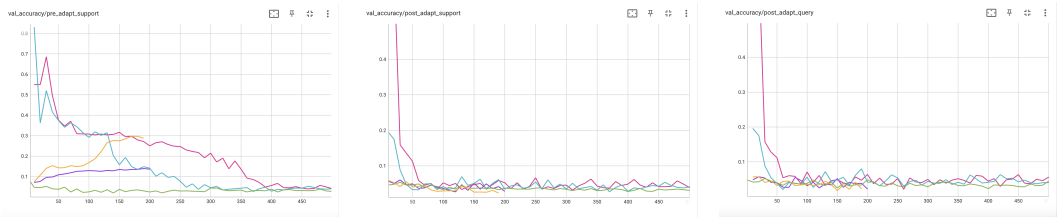
4.2.3 Transformer

We implemented two transformer architectures taking inspiration from [15], which leverage the encoder blocks from [14] in order to create an architecture capable of generating raw audio waveforms for prediction of upcoming samples for packet loss concealment in network audio. Inputs were quantized from 16-bit samples into 8-bit samples, which were then mapped into tokens in the set $\{0, ..., 255\}$ and passed into an embedding layer. Positional encoding is then added to the embeddings. After this, two transformer encoder blocks follow, which are then met by a linear layer outputting a vector of the same length as the audio waveform input sample size.

The smaller transformer architecture was trained with an embedding dimension of 68, 2 tranformer encoder blocks, 2 attention heads per block, and dropout of 0.5. The larger transformer architecture was trained with an embedding dimension of 128, 2 encoder blocks, 4 attention heads per block, and dropout of 0.5. Note that the model hyperparameters of the larget transformer model match the hyperparameters outlined by [15], except for the number of transformer encoder blocks. Our larger model uses 2 transformer encoder blocks, while [15] uses encoder 3 layers. The reduction in number of encoder blocks was made due to GPU limitations.

Base Model Structure	# parameters	inner steps	inner lr	outer lr	MAE Loss	STFT Loss
LSTM	265k	1	0.1	1e-4	0.031	2.353
LSTM-small	17k	1	0.1	1e-4	0.038	2.538
TCN	16k	1	0.5	5e-4	0.068	–
TCN	16k	3	0.05	5e-4	0.045	–
TCN	16k	10	0.05	1e-4	0.036	4.353
Transformer	1219k	1	0.02	0.02	0.052	4.942
Transformer-small	617k	1	0.02	0.02	0.055	4.954

Table 1: Experiment results from different model and hyper-parameters



(a) Pre-adaptation support L1 loss (b) Post-adaptation support L1 loss (c) Post-adaptation query L1 loss

Figure 3: Loss of the models during training: yellow: LSTM, purple: LSTM-small, pink: Transformer, cyan: Transformer-small, green: TCN with 10 inner steps. (x-axis: training steps, y-axis: MAE/L1 loss)

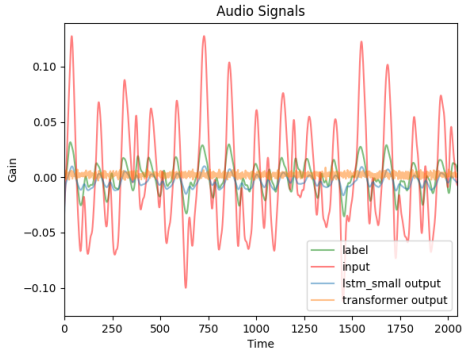
5 Data Analysis

We show the results from experiments for different model and hyper-parameters in Table 1. We report the loss on the test set that is completely unseen to our models during training and validation. We note here that we observe a considerable amounts of of fluctuation in loss during training, as shown in Figure 3. This loss volatility emerges regardless of model structure; we postulate that MAML might not be the best choice for processing audio. In addition, we stopped training for LSTM earlier compared to other models since the backpropagation process consumes vast amounts of time and memory on extremely long (>4000 time steps) audio sequences.

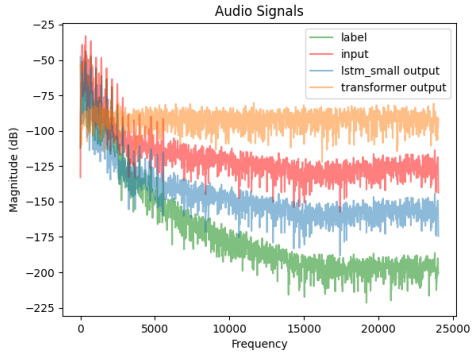
We observe that across different loss metrics, LSTM has the best performance, even when the model is compared to others with similar parameter sizes. In the literature[13], the TCN model has an MAE loss of around 0.007 to 0.02 and STFT loss of 0.6 to 1.1 (note that these losses are recorded based on a different dataset; that being said, we want to mention the expected loss for models trained on a single task in audio domain as reference.) With meta-learning, we can observe that MAE loss obtains similar range, but STFT loss is much higher than desired. This might mean that our model learns to mimic the gain level of the label but remain ineffective in filtering high frequency noise (this is further discussed in later paragraphs). We note that one big factor that seems to influence performance is the number of updates during the inner loop. For the TCN model, enabling more inner-loop steps improves the performance drastically. This might also explain why LSTM model has superior performances, as it gets multiple updates for each support example given. We also believe that since we exclusively work with data with extremely long sequence, this phenomenon is especially prominent.

Most of our models show strong improvement post-adaptation, demonstrating that the support examples are being effectively utilized during meta-training. We observe peculiar behavior in LSTM models wherein the pre-adaptation loss actually increased during training; we are unsure of the cause, but we hypothesize that good meta-parameters for LSTM might simply be sub-optimal to use directly before adaptation.

In the next page, we illustrate our results in both the time and frequency domains (Figure 4). Since we are working with a Moog resonant low-pass filter, we see that the label's high frequency information is reduced and the lower frequency gets amplified. The LSTM seems to be the only model that is able to learn this behavior. We still see some high-frequency information in LSTM output; this might be due to the sequential nature of LSTM, as it is unable to take account for the future input. Both

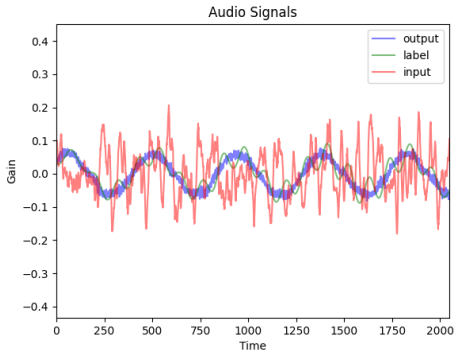


(a) time domain output (x-axis: time step, y-axis: model Output/Audio Gain)

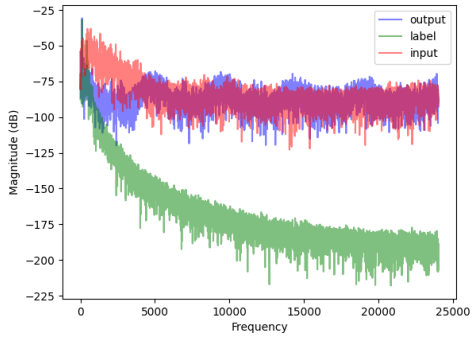


(b) frequency domain output (x-axis: frequency, y-axis: magnitude in log scale)

Figure 4: Sample Output from Transformer and LSTM model



(a) time domain output (x-axis: time step, y-axis: model Output/Audio Gain)



(b) frequency domain output (x-axis: frequency, y-axis: magnitude in log scale)

Figure 5: Sample Output from TCN model

transformer and TCN models are unable to remove high frequency information from the data. In Figure 5, we see that TCN model is able to somewhat track the label in gain in the left graph, but the model is unable to remove any of the high frequency noises left, as shown in the frequency domain. This illustrates why TCN still has high STFT loss even when it is doing relatively well in MAE Loss.

6 Future Work

In the future, we hope to explore additional meta-learning methods. While model-agnostic meta-learning does seem to show some potential promise, the results are still far from what is needed. Thus, exploring other meta-learning solutions may be better. In particular, we want to explore black box meta learning methods applied small recurrent structures. This may be more effective at predicting a model parameters directly.

We found GPU memory to be a major bottleneck when implementing MAML algorithims. This is in large part to the high information density of audio but also due to the large number of gradients in the computation graph that must remain held in memory when computing the outer loop of MAML. Future work with this system will focus on optimizing the algorithm further to be less memory intensive, allowing larger block-sizes to be used during training.

Finally, we hope to investigate custom small recurrent models for audio, for example the one shown in [8]. In the authors' opinion, these small models with few parameters hold the most promise for MAML methods in the future. They have the potential to be highly expressive with comparatively few

parameters to predict. We also would like explore bidirectional sequential model such as bidirectional LSTM that is able to reason both on the future and past data sequence.

7 Conclusion

In this experiment, we implemented a model-agnostic meta learning algorithm that attempts to learn audio filters. We implemented various popular machine learning structures in audio effect modeling inside of the MAML algorithm. We obseved that attention-based and convolution-based models performs sub-optimally with MAML, whereas the less popular recurrent models achieve close to state-of-the-art performance in the L1 loss metric. However, all models still lack the capability to remove high-frequency information from audio, and we struggle to achieve comparable results to models that are trained on one single task.

8 Team Contribution

Dirk assembled the unprocessed set of audio samples, wrote scripts to process the audio using `DAWDreamer`, and implemented the functional version of TCN model. Ivan integrated the `higher` library for more rapid model prototyping and experimentation with MAML and implemented and trained both transformer models. Lucy worked on training the TCN model without meta-learning for earlier prototyping, implemented and trained all LSTM models, and performed data analysis and visualization. All team members contributed to literature review and writing of the required documents for the project equally. Many tasks has changed from the initial proposal phase, including the extra need to functionalize model, adding additional LSTM and transformer model. There were also many debugging and data analysis taks done by all the members during the project. The new tasks are split relatively equally across the team.

References

[1] Jatin Chowdhury and Christopher Johann Clarke. "Emulating diode circuits with differentiable wave digital filters". In: *Proc. Sound and Music Computing Conf.(SMC)*. 2022.

[2] Eero-Pekka Damskägg et al. "Deep Learning for Tube Amplifier Emulation". In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, pp. 471–475. DOI: 10.1109/ICASSP.2019.8682805.

[3] Champ C Darabundit and O Roosenburg Dirk; Smith III. "Neural-Net Tube Models for Wave Digital Filters". In: (2022).

[4] Sung-Feng Huang et al. "Meta-tts: Meta-learning for few-shot speaker adaptive text-to-speech". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 30 (2022), pp. 1558–1571.

[5] Pranay Manocha et al. "A differentiable perceptual audio metric learned from just noticeable differences". In: *arXiv preprint arXiv:2001.04460* (2020).

[6] Marco A. Martínez Ramírez and Joshua D. Reiss. "Modeling Nonlinear Audio Effects with End-to-end Deep Neural Networks". In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. ISSN: 2379-190X. May 2019, pp. 171–175. DOI: 10.1109/ICASSP.2019.8683529.

[7] Aaron van den Oord et al. *WaveNet: A Generative Model for Raw Audio*. 2016. DOI: 10.48550/ARXIV.1609.03499. URL: https://arxiv.org/abs/1609.03499.

[8] Julian D Parker, Fabián Esqueda, and André Bergner. "Modelling of nonlinear state-space systems using a deep neural network". In: *Proceedings of the International Conference on Digital Audio Effects (DAFx), Birmingham, UK*. 2019, pp. 2–6.

[9] Julian D Parker et al. "Physical Modeling using Recurrent Neural Networks with Fast Convolutional Layers". In: *arXiv preprint arXiv:2204.10125* (2022).

[10] Davide Plozza, Marco Giordano, and Michele Magno. "Real-Time Low Power Audio Distortion Circuit Modeling: a TinyML Deep Learning Approach". In: *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. 2022, pp. 415–418. DOI: 10.1109/AICAS54282.2022.9870024.

[11] Christian J Steinmetz and Joshua D. Reiss. *Auraloss: Audio-focused loss functions in Pytorch*. URL: https://static1.squarespace.com/static/5554d97de4b0ee3b50a3ad52/t/5fb1e9031c7089551a30c2e4/1605495044128/dmrn15_auraloss_audio_focused_loss_functions_in_pytorch.pdf.

[12] Christian J Steinmetz et al. "Automatic multitrack mixing with a differentiable mixing console of neural audio effects". In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2021, pp. 71–75.

[13] Christian J. Steinmetz and Joshua D. Reiss. *Efficient neural networks for real-time modeling of analog dynamic range compression*. 2021. DOI: 10.48550/ARXIV.2102.06200. URL: https://arxiv.org/abs/2102.06200.

[14] Ashish Vaswani et al. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

[15] Prateek Verma and Chris Chafe. *A Generative Model for Raw Audio Using Transformer Architectures*. 2021. DOI: 10.48550/ARXIV.2106.16036. URL: https://arxiv.org/abs/2106.16036.

[16] Alec Wright, Eero-Pekka Damskägg, Vesa Välimäki, et al. "Real-time black-box modelling with recurrent neural networks". In: *22nd international conference on digital audio effects (DAFx-19)*. 2019.

[17] Alec Wright and Vesa Välimäki. "Perceptual loss function for neural modeling of audio systems". In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 251–255.