Audio Engineering Society

Conference Paper 34

Presented at the AES International Conference on Machine
Learning and Artificial Intelligence for Audio
2025 September 8–10, London, UK

This paper was peer-reviewed as a complete manuscript for presentation at this conference. This paper is available in the AES E-Library (http://www.aes.org/e-lib), all rights reserved. Reproduction of this paper, or any portion thereof, is not permitted without direct permission from the Journal of the Audio Engineering Society.

NablAFx: A Framework for Differentiable Black-box and Gray-box Modeling of Audio Effects

Marco Comunità[1], Christian J. Steinmetz[1], and Joshua D. Reiss[1]

[1]*Centre for Digital Music, Queen Mary University of London, UK*

Correspondence should be addressed to Marco Comunità (`m.comunita@qmul.ac.uk`)

ABSTRACT

We present NablAFx, an open-source framework developed to support research in differentiable black-box and gray-box modeling of audio effects. Built in PyTorch, NablAFx offers a versatile ecosystem to configure, train, evaluate, and compare various architectural approaches. It includes classes to manage model architectures, datasets, and training, along with features to compute and log losses, metrics and media, and plotting functions to facilitate detailed analysis. It incorporates implementations of established black-box architectures and conditioning methods as well as differentiable DSP blocks and controllers, enabling the creation of both parametric and non-parametric gray-box signal chains. Beside established conditioning methods like concatenation, feature-wise linear modulation (FiLM) and temporal feature-wise linear modulation (TFiLM), we propose three further methods: time-varying concatenation (TVCond), tiny TFiLM (TTFiLM) and time-varying FiLM (TVFiLM), as efficient implementations of time-varying conditioning similar to TFiLM. We also propose the Static Rational Linearity as a flexible and efficient differentiable processor to learn nonlinear functions. The code is accessible at `https://github.com/mcomunita/nablafx`.

1 Introduction

Audio effects are central for engineers and musicians to shape timbre, dynamics, and spatialisation of sound [1]. Therefore, research related to audio effects, especially with the success of deep learning and differentiable digital signal processing (DDSP) [2], is a very active field [3]. This includes applications such as classification and identification [4], parameters estimation [5, 6], modeling [7, 8], style transfer [9, 10], automatic mixing [11, 12]. Audio effects modeling is one of the most active applications of differentiable approaches, with the majority of methods falling into black-box (i.e., neural networks) and gray-box (i.e., DDSP) paradigms. While black-box models achieve state-of-the-art accuracy [7, 13–15] there is interest in gray-box ones [16–20] due to interpretability and potential for efficiency.

Comparing modeling paradigms remains challenging due to significant variations in training and evaluation methods. In addition, the lack of standardized implementations for models and DDSP blocks further impedes reproducibility and performance assessment. There are a growing number of audio effect implementations available to researchers, however existing options remain limited in a number of ways (see Table 1).

While the `Spotify Pedalboard`[1] library offers Python implementations of common audio effects and allows to define signal chains, these are not differentiable. DDSP, introduced in [2], provides some differentiable blocks[2], though they are neither common nor

[1] `github.com/spotify/pedalboard`

[2] `github.com/magenta/ddsp`

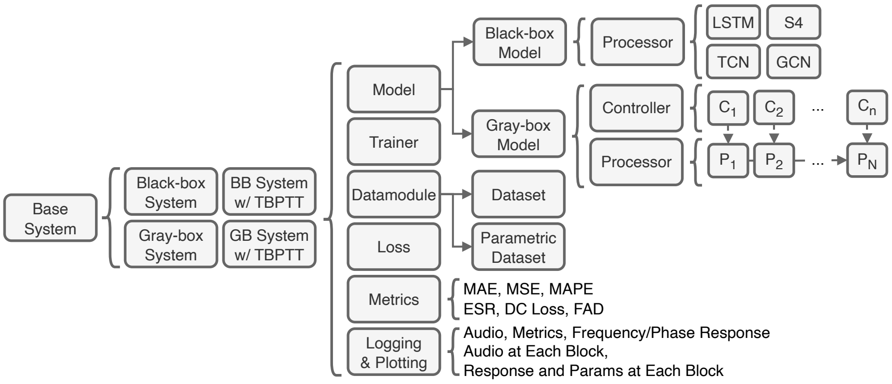


Fig. 1: Overview of the NablAFx framework for audio effects modeling

Table 1: Python libraries for processing/modeling applications. We show if: they include differentiable (Diff.) implementations, neural networks (NN), DSP processors (Proc.) and controllers (Contr.), they allow to define signal chains and include analysis tools.

Library	Diff.	NN	Proc.	Contr.	Chains	Analysis
`Pedalboard`	✗	✗	✓	✗	✓	✗
`DDSP`	✓	✗	✓	✗	✗	✗
`dasp`	✓	✗	✓	✗	✗	✗
`diffmoog`	✓	✗	✓	✗	✓	✗
`GRAFX`	✓	✗	✓	✗	✓	✗
`pyneuralfx`	✓	✓	✗	✗	✗	✓
`NablAFx`	✓	✓	✓	✓	✓	✓

easily reusable, since they are focused on specific applications within audio synthesis. `dasp`[3][9] includes differentiable implementations of common processing and mixing blocks, and while useful when imported into larger projects, the library in not meant to define signal chains. Interconnections of processors can be defined in `diffmoog`[4][21], although mainly focused on FM synthesis and not suitable for effects modeling. Also `GRAFX`[5] [22] enables complex interconnections, but lacks external control, limiting parametric, time-varying, and modulated signal chains for effects modeling.

`pyneuralfx`[6][23] is the only framework designed for modeling and, while it includes state-of-the-art neural networks, it focuses only on black-box approaches and does not include time-varying models [7]. Even though it provides functions for inference-time analysis, it lacks logging and plotting features during training and testing. Also, experiment configurations are hard to modularize and adapt to different datasets, models, or training procedures, limiting repeatability and comparison.

To address these limitations and advance differentiable audio effects modeling, we propose **NablAFx**, which provides:

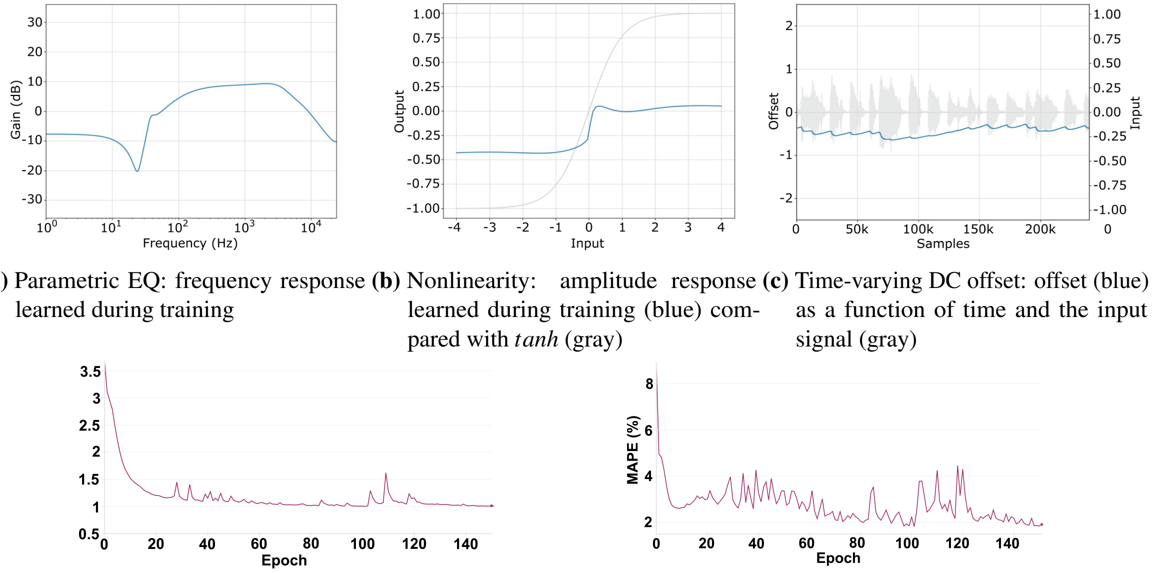
- Black-box architectures and condition methods: concatenation, FiLM, TFiLM, TVCond, TTFiLM, TVFiLM
- Gray-box architectures composed of differentiable processors, and differentiable controllers for parametric/non-parametric models.
- Modules to manage datasets, training, and loss functions.
- Tools to log metrics and media during training and testing.
- Plotting functions for analysis throughout training.

2 Framework

NablAFx is a framework for audio effects modeling that allows researchers to easily define, train, evaluate and compare differentiable black-box and gray-box models. As shown in Fig. 1, it integrates models, datasets, trainers, loss functions, metrics, and logging/plotting tools. Built with PyTorch Lightning[7], it leverages Weights&Biases[8] to log results and media.

System — In NablAFx all necessary functionalities are contained in an audio effects modeling system class. The *BaseSystem* class handles the initialization of loss functions, optimizers, learning rate scheduler, metrics, and includes shared methods to compute and log loss, metrics, audio and frequency/phase response. The *BaseSystem* is divided into *BlackBoxSystem* and *GrayBoxSystem*, which initialize black-box and gray-box

[3] `github.com/csteinmetz1/dasp-pytorch`
[4] `github.com/aisynth/diffmoog`
[5] `github.com/sh-lee97/grafx`
[6] `github.com/ytsrt66589/pyneuralfx`
[7] `lightning.ai/pytorch-lightning`
[8] `wandb.ai/site`



(a) Parametric EQ: frequency response learned during training

(b) Nonlinearity: amplitude response learned during training (blue) compared with *tanh* (gray)

(c) Time-varying DC offset: offset (blue) as a function of time and the input signal (gray)

(d) Loss: L1 + MR-STFT

(e) Metrics: MAPE

Fig. 2: Examples of plotting features included in NablAFx

models, respectively, and implement train, validation, and test steps. The *GrayBoxSystem* adds methods to log audio output, plot/log frequency and time responses, and parameters values for each stage of the signal chain. Both systems are extended with *WithTBPTT* classes, which implement truncated backpropagation through time to enable faster training of recurrent networks[13].

Model — In our framework, black-box models can be any neural network - with outputs defined as a function of input and controls $y = f(x,c)$ - represented by the *Processor* class in *BlackBoxModel*. Gray-box models comprise interconnected differentiable blocks, forming a function composition: $y = (f_1 \circ f_2 \circ \ldots \circ f_N)(x,c)$, and the *Processor* class defines a chain of processors. A *Controller* class defines a chain of controllers, each associated with a processor, allowing the definition of parametric and time-varying models that are a function of both input audio and controls.

Data — The *DataModule* class takes care of initializing the dataset and dataloaders for train, validation and testing. *AudioEffectDataset* and *ParametricAudioEffectDataset* classes are used to manage data for non-parametric and parametric models.

Metrics — Metrics are computed with: *torchmetrics*[9] for mean absolute error (MAE), mean squared error (MSE) and mean absolute percentage error (MAPE); *auraloss*[10][24] for error-to-signal ratio (ESR) and DC loss; and the *frechet-audio-distance*[11] package for Frechét Audio Distance (FAD) [25].

Plotting — In addition to logging losses, metrics and audio examples, we provide methods to plot and log frequency/phase response for the whole system, as well as frequency/time response and parameters values for each DDSP block in a gray-box system. We offer two methods to compute the frequency and phase response: one using an exponential sine sweep[12] [26], suitable for linear and mildly nonlinear systems, and a custom method designed for nonlinear systems. The latter measures the system's response in steps, using sinusoidal inputs at exponentially spaced frequencies. To ensure reliable measurements, each sinusoid lasts several seconds for the system to reach steady state, with magnitude/phase response computed only from the final segment as a function of the minimum frequency.

$$x = x[-T \cdot \lfloor f_s/f_1 \rfloor :]$$
$$y = y[-T \cdot \lfloor f_s/f_1 \rfloor :]$$

where T is the signal duration (e.g. 5 s), f_s is the sample rate and f_1 is the minimum frequency of the stepped sweep (e.g., 10 Hz). Fig. 2a shows the frequency response of a Parametric EQ block, while Fig. 2b and 2c display examples of learned nonlinearity (vs. *tanh*, light gray) and time-varying DC offset (vs. input signal, light gray). In Fig. 2d and 2e we also show examples of plots for loss (L1 + MR-STFT) and metrics (MAPE) during training, while in Fig. 3 we show and example of frequency and phase responses measured after training using our proposed method for highly nonlinear effects.

[9] `lightning.ai/docs/torchmetrics/stable/`

[10] `github.com/csteinmetz1/auraloss`

[11] `github.com/gudgud96/frechet-audio-distance`

[12] `ant-novak.com/pages/sss/`

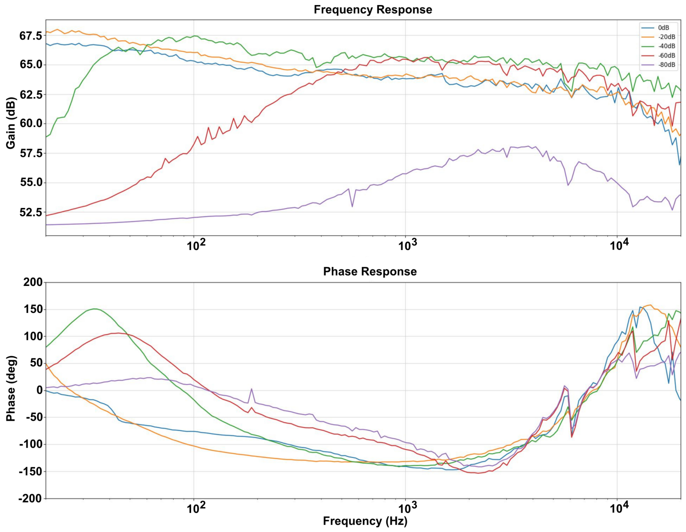


Fig. 3: Example of frequency and phase response measurement after training

2.1 Differentiable Black-box Models

This section provides an overview of state-of-the-art neural network architectures and conditioning methods included in NablAFx.

LSTM — The recurrent neural network architecture we implement is widely adopted for nonlinear effects (e.g., overdrive, distortion, guitar amps) [13, 27], nonlinear time-varying effects (e.g., fuzz, compressor) [7, 14], and modulation effects (e.g., phaser, flanger) [6, 28]. As shown in Fig. 4a, it consists of a single LSTM layer, a linear layer, and a *tanh* activation. For parametric models, a conditioning block processes control values and optionally the input sequence.

TCN — Temporal Convolution Networks (TCNs), introduced in [29] and shown to outperform recurrent architectures [30] on a variety of tasks, were proposed for audio effects modeling [14, 31, 32] and applied to linear (EQ, reverb) and nonlinear time-varying (compressor) effects. The architecture (Fig. 4b) consists of a series of residual blocks (Fig. 4c) made of 1-dimensional convolutions with increasing dilation factors, optionally followed by batch normalization and conditioning block, and an activation function (here *tanh*). A linear layer matches the output channels to the input size.

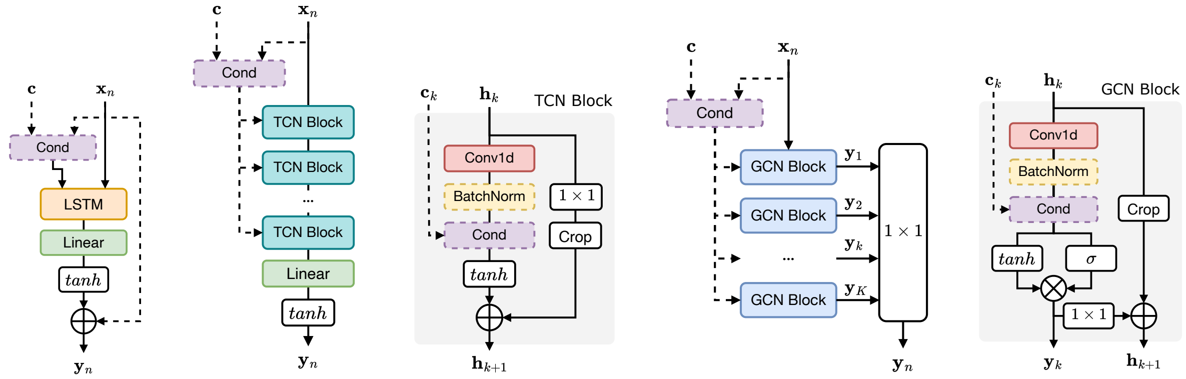
GCN — Gated Convolution Networks (GCNs), introduced in [33] as a feed-forward WaveNet, are a special case of TCNs with gated convolutions. GCNs have been used in [27, 34] for nonlinear audio effects (guitar amp, overdrive, distortion) and in [7] for nonlinear time-varying effects (compressor, fuzz). Beside the activation function at each block (Fig.4e), a GCN (Fig.4d) differs from a TCN in that its output is a linear combination of the activation features at each block.

S4 — Structured state space sequence models (S4) were introduced in [35] as a general sequence modeling architecture and shown to outperform recurrent, convolutional and Transformer architectures on a variety of tasks. An S4 layer is a differentiable implementation of an infinite impulse response (IIR) system in state-space form, with a theoretically infinite receptive field, similar to recurrent networks. Based on these observations state-space models were adopted for non-linear time-varying (compressor) effects modeling [36, 37].

The architecture in our framework, based on [36] (Fig. 5a), consists of S4 blocks. Unlike standard convolutional ones, S4 layers are not combined or mixed across data channels, this explains the use of a linear layer and activation function (tanh) at the input of each S4 block (Fig. 5b) for affine transformations along the channel dimension. These are followed by an S4D layer [38], which uses diagonal matrices for a parameter-efficient implementation, optional batch normalization and conditioning block, followed by an activation function (*tanh* in this case). Linear layers are used at the start and end to adjust the channel count to match the input data.

2.1.1 Conditioning for Black-box Models

Conditioning mechanisms for black-box models have been explored for different purposes: to include parametric control [8, 14, 36], to capture long-range dependencies [7] or for modulation in LFO-driven effects [6, 28]. While concatenation and feature-wise linear modulation (FiLM) [6, 8, 14, 36] remain the most common methods, temporal FiLM (TFiLM) has been adopted to capture time-varying behavior [7]. Beside these established methods, in this work we also propose three further conditioning methods: time-varying concatenation (TVCond), tiny TFiLM (TTFiLM) and time-varying FiLM (TVFiLM), as efficient implementations of time-varying conditioning similar to TFiLM. Concatenating control values ($\mathbf{c}$) to the input sequence ($\mathbf{x_n}$)



(a) LSTM **(b)** TCN **(c)** TCN block **(d)** GCN **(e)** GCN block

Fig. 4: Black-box architectures included in NablAFx

along the channel dimension is a simple, parameter-efficient conditioning method. It serves as a baseline for recurrent networks and has been used for parametric control in, e.g., compression [14] and overdrive [15].

Equally common is FiLM conditioning, mainly when using TCN [14, 15], GCN [15] or S4 [36, 37] backbones, with works adopting it for compressors [8, 14, 37] and overdrive [8, 15] modeling. Introduced in [39] as a general-purpose conditioning method, FiLM modulates a neural network's intermediate features using a conditioning vector $\mathbf{c}$. It learns functions f and g to generate scaling ($\gamma_{k,c} = f(\mathbf{c})$) and bias ($\beta_{k,c} = g(\mathbf{c})$) parameters for each layer k and channel c, which are used to modulate the activations at each layer $\mathbf{h}_{k,c}$, via a feature-wise affine transformation:

$$\mathrm{FiLM}(\mathbf{h}_{k,c}, \gamma_{k,c}, \beta_{k,c}) = \gamma_{k,c} \cdot \mathbf{h}_{k,c} + \beta_{k,c}. \tag{1}$$

In practice, f and g are neural networks (Fig. 5c) that learn a latent representation $\mathbf{z}$ of the conditioning vector $\mathbf{c}$; then, a linear layer uses the latent representation to generate scaling and bias parameters for each block of the main network (Fig. 5d).

TFiLM [40] enhances network expressivity by using recurrent networks to modulate intermediate features over time as a function of layer activations $\mathbf{h}_k$ and optionally a conditioning vector $\mathbf{c}$ (Fig. 6a). Given a sequence of activations $\mathbf{h}_k$ from the k-th block of a network, the sequence is split into T blocks of B samples $\mathbf{h}_{k,b_1:b_T}$ along the sequence dimension. For each block $\mathbf{h}_{k,b_t}$, 1-dimensional max pooling downsamples the signal by a factor of B. To include the conditioning vector $\mathbf{c}$, it is repeated T times and concatenated with the downsampled activations. Then, an LSTM generates scaling $\gamma_{k,b_1:b_T,c}$ and bias $\beta_{k,b_1:b_T,c}$ parameters for each channel c, which are used to modulate the activations in each block via an affine transformation:

$$\mathrm{TFiLM}(\mathbf{h}_{k,b_1:b_T,c}, \gamma_{k,b_1:b_T,c}, \beta_{k,b_1:b_T,c}) = \gamma_{k,b_1:b_T,c} \cdot \mathbf{h}_{k,b_1:b_T,c} + \beta_{k,b_1:b_T,c}.$$

In its standard formulation, TFiLM conditioning adds a recurrent network for each block in the main neural network, which can lead to a significant increase in parameters due to the number of blocks (typically 5-10) and channels (typically 16-32).

To retain TFiLM's expressivity while reducing parameters and computational cost, we propose two methods: TTFiLM and TVFiLM. TTFiLM (Fig.6b) is structurally similar to TFiLM, and reduces the computational complexity by using fewer channels in the recurrent network, achieved through a linear layer before it. The output is then scaled up to the required number of scaling $\gamma_{k,b_1:b_T,c}$ and bias $\beta_{k,b_1:b_T,c}$ channels using a small MLP. TVFiLM is a time-varying extension of FiLM conditioning. It replaces the MLP in the FiLM controller (Fig. 5c) with a recurrent network (Fig. 6c), creating a time-dependent latent representation $\mathbf{z}_{n,b1:b_T}$ shared across the main network's blocks. Modulation sequences are then generated at each block via a linear layer (Fig.6d), similarly to standard FiLM (Fig.5d).

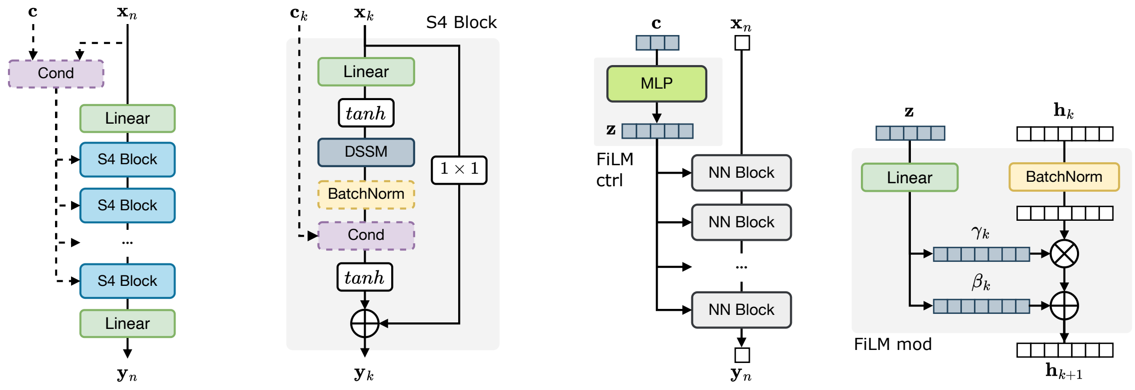
We also implement time-varying concatenation (TVCond) for recurrent models by using a TVFiLM controller to generate a time-dependent conditioning sequence, which is concatenated to the input for greater expressivity compared to standard concatenation.

2.2 Differentiable Gray-box Models

As described in Sec. 2 we define a gray-box model as a sequence of differentiable processors, each with an associated controller which generates the control parameters that dictate the behavior of the processor.

2.2.1 Differentiable Audio Processors

For our application, we define three types of audio processors: basic (e.g., phase inversion, gain), filters (e.g., EQ, shelving), and nonlinearities (e.g., tanh, MLP).



(a) S4 **(b)** S4 block **(c)** FiLM controller **(d)** FiLM modulator

Fig. 5: Black-box architectures and conditioning methods included in NablAFx

Most processors can be controlled by any of the controllers in Sec. 2.2.2, enabling parametric and time-varying configurations. All filters are implemented as differentiable biquads [41], with the transfer function:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}}.$$

Biquad coefficients are calculated based on center/cutoff frequency (Hz), gain (dB), and Q factor, following Robert Bristow-Johnson's method[13]. To implement N^{th} order filters (e.g., Parametric EQ) we follow the common practice of using K cascaded second order sections:

$$H(z) = \prod_{k=0}^{K} H_k(z) \tag{2}$$

The frequency response is computed evaluating the transfer function along the unit circle in the complex plane and taking the magnitude:

$$\left|H(e^{j\omega})\right| = \left|\prod_{k=0}^{K} H_k(e^{j\omega})\right|.$$

For efficiency, during training we adopt the frequency sampling method, which approximates a cascade of second order IIR filters by computing the frequency response as in Eq. 2, applying the convolution in the frequency domain via multiplication and using the inverse FFT to transform the signal back to the time domain:

$$y[n] = F^{-1}[Y(e^{j\omega})] = F^{-1}[X(e^{j\omega})H(e^{j\omega})].$$

In the following paragraph we describe each processor. Generally, all processors' parameters can depend on the input **x** and/or controls **c**.

Phase Inversion — Invert the phase of the input.

Gain — Multiply input by a gain value in dB.

DC Offset — Add a constant value to the input.

Lowpass/Highpass — Second order lowpass/highpass implemented as a single biquad section. Each filter is defined by 2 parameters: cutoff frequency and Q factor.

Low/High Shelf — Second order low/high shelving filter implemented as a single biquad section. Each filter is defined by 3 parameters: gain, cutoff frequency, Q factor.

Peak/Notch — Second order peak/notch filter implemented as a single biquad section. Each filter is defined by 3 parameters: gain, center frequency, Q factor.

Parametric EQ — We define a Parametric EQ as a chain of 5 filters: low shelf, three peak/notch filters and a high shelf. Each Parametric EQ has 15 parameters.

Shelving EQ — We define a Shelving EQ as a chain of 4 filters: highpass, low shelf, high shelf, lowpass. Each Shelving EQ is defined by a total of 10 parameters.

Static FIR Filter — We define a Static FIR Filter using a SIREN layer [42], which stores the tap values of an N^{th}-order impulse response. The network can be initialized with a pre-trained response (e.g., loudspeaker) and its hyperparameters (hidden dimension and layers count) to be customized.

Tanh Nonlinearity — Standard hyperbolic tangent.

Static MLP Nonlinearity — MLP Nonlinearity implemented with SIREN layer, initialized by default with a pre-trained model approximating a *tanh*.

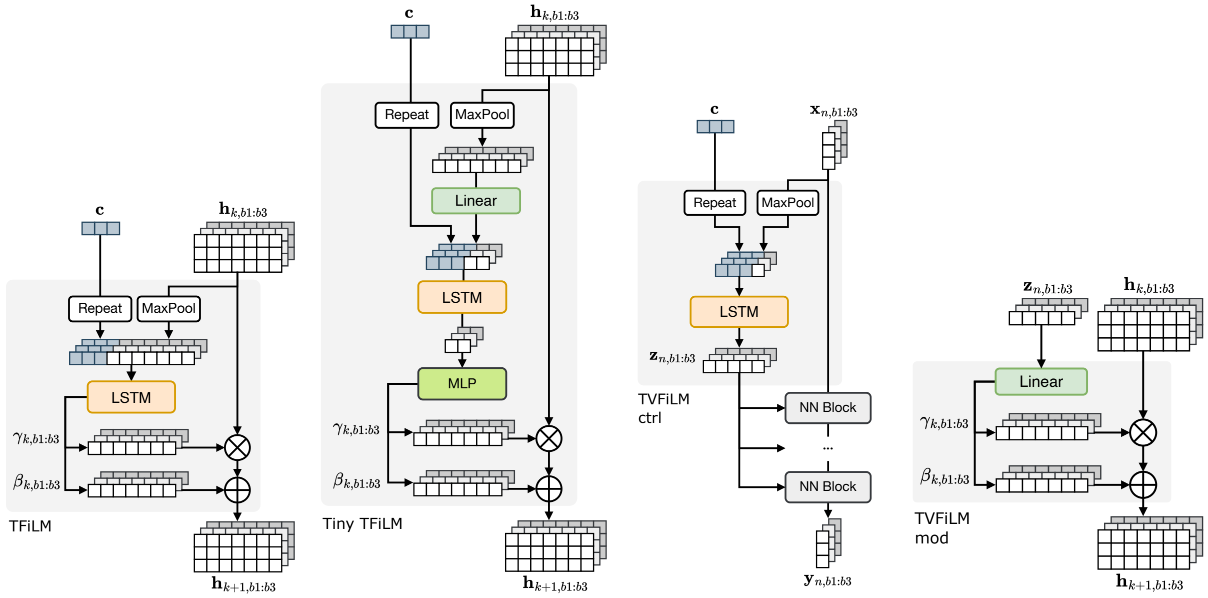
Static Rational Nonlinearity — A Padé approximant [43] is a rational function of order $[m/n]$ that best approximates a function $f(x)$ near a specific point, with $m \geq 0$ and $n \geq 1$:

$$R(x) = \frac{a_0 + a_1 x + a_2 x^2 + \cdots + a_m x^m}{1 + b_1 x + b_2 x^2 + \cdots + b_n x^n}$$

which agrees with $f(x)$ to the highest possible order. Learnable Padé approximants[14] [44] enable flexible ra-

[13] www.musicdsp.org/en/latest/Filters/197-rbj-audio-eq-cookbook.html

[14] github.com/ml-research/rational_activations



(a) TFiLM **(b)** TTFiLM **(c)** TVFiLM controller **(d)** TVFiLM modulator

Fig. 6: Black-box architectures and conditioning methods included in NablAFx

tional activation functions with few weights (numerator and denominator coefficients). We define a learnable Static Rational Nonlinearity using a single rational activation layer, initialized by default to a *tanh* approximation of order $[6,5]$.

2.2.2 Differentiable Controllers

We define five types of differentiable controllers (Fig. 7) to generate control parameters for an audio processor.
Dummy — A Dummy controller is a placeholder for processors that don't require control parameters (e.g., Phase Inversion, Static FIR Filter, Static Nonlinearity).
Static — A Static controller is a tensor of trainable controls $\mathbf{b}$ - one for each control parameter in the respective processor - followed by a sigmoid function to limit the output to the [0,1] range: $g = \sigma(\mathbf{b})$.
Static Conditional — A Static Conditional controller uses an MLP with a sigmoid activation to adjust control parameters based on audio effects controls (or some other fixed values): $g = f(\mathbf{c})$. Hyperparameters include number of input controls and output control parameters, number of layers, and hidden dimensions.
Dynamic — A Dynamic controller is used to adjust control parameters over time based on another signal, oftentimes the input audio: $g[n] = f(x[n])$. The control signal is downsampled (default downsampling factor is 128), processed through an LSTM, a sigmoid activation, and upsampled to output a control parameters sequence $\mathbf{g}_n$ at the original rate. Hyperparameters include block size (i.e., downsampling factor) and number of recurrent layers, while the hidden size is set by the number of control parameters for each processor.

Dynamic Conditional — A Dynamic Conditional controller adjusts control parameters based on both fixed values (typ., audio effect controls) and a time-varying control (typ., input signal): $g[n] = f(x[n], c)$. The signal is downsampled while the controls are upsampled and concatenated, the sequence processed by an LSTM, and after sigmoid activation and upsampling, the control parameters sequence $\mathbf{g}_n$ is returned at the original rate.

Although the control parameters sequences are at sampling rate, the block size (i.e., downsampling rate) is used internally in each processors to downsample the sequence so that the coefficients are recomputed once per block. This is not a limitation, as setting the block size to 1 provides sequences at audio rate. No interpolation methods have been implemented for smooth control sequences at the time of writing.

3 Audio Effects Modeling

To showcase our audio effects modeling framework and evaluate the proposed conditioning methods we train parametric black- and gray-box models of the Multidrive Pedal Pro F-Fuzz—a digital emulation of the Dallas Arbiter Fuzz Face—while for a complete study on differentiable black-box and gray-box modelling of audio effects that uses NablAFx, we point the reader to [45]. Table 2 shows all models configurations. We select TCN and S4 models and evaluate all conditioning methods available (i.e., FiLM, TFiLM, TTFiLM, TVFiLM). The table shows how TTFiLM and TVFiLM enable implementation of time-varying conditioning with a small overhead w.r.t. FiLM.

Table 3: Test loss for parametric models trained on Multidrive Pedal Pro F-Fuzz. Best model for each architecture shown in **bold**.

Model	Tot.	$L1$	MR-STFT
TCN-F-45-S-16	0.7095	0.0217	0.6878
TCN-TF-45-S-16	**0.4886**	0.0077	0.4809
TCN-TTF-45-S-16	0.5324	0.0102	0.5223
TCN-TVF-45-S-16	0.5356	0.0115	0.5241
S4-F-S-16	0.7687	0.0243	0.7444
S4-TF-S-16	0.4034	0.0075	0.3959
S4-TTF-S-16	0.3816	0.0066	0.3749
S4-TVF-S-16	**0.3354**	0.0058	0.3296
GB-C-DIST-MLP	1.2104	0.0611	1.1492
GB-C-DIST-RNL	1.2531	0.0672	1.1858
GB-C-FUZZ-MLP	**0.9303**	0.0345	0.8958
GB-C-FUZZ-RNL	0.9395	0.0355	0.9040

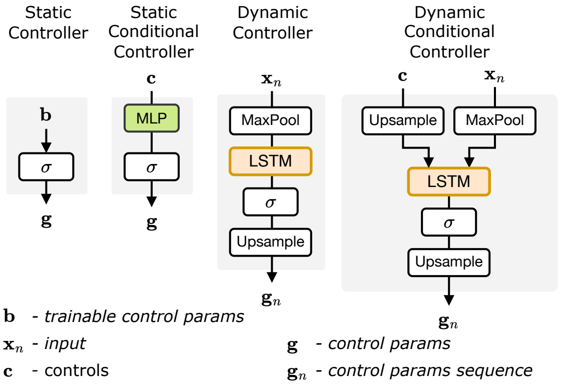


Fig. 7: Controllers included in NablAFx

We propose two gray-box architectures (GB-DIST and GB-FUZZ) that are extensions of the typical Weiner-Hammerstein model [16] adopted for distortion modeling, which includes a memoryless nonlinearity in between pre-emphasis and de-emphasis linear time-invariant filters. We test two nonlinearities: Static MLP (MLP) and Static Rational Nonlinearity (RNL). While GB-DIST models only use Static Conditional controllers, in GB-FUZZ we opt for a Dynamic Conditional controller for the Offset block, to capture the characteristic dynamic bias shift of fuzz effects.

Models are trained for a maximum of 15k steps using a weighted sum of $L1$ and MR-STFT [24] losses and the results shown in Table 3. For TCN models, TTF and TVF conditioning perform on par with TF; while for S4 models TTF and TVF outperform TF. For GB models, regardless of the nonlinearity type, GB-FUZZ achieves better results than GB-DIST, proving the Dynamic controller useful. Also, RNL in shown to be an effective and efficient alternative to the MLP nonlinearity.

4 Conclusion

In this work we presented NablAFx, an open-source framework developed to support research in differentiable black-box and gray-box audio effects modeling. Its modular design enables easy configuration of experiments with different architectures, datasets, training settings, and loss functions. With logging, plotting, and performance metrics, it simplifies experiment analysis and comparison. We consider gray-box models as a series connection of DDSP blocks, but this could be generalized using a graph representation. While black-box models are currently single networks, they could be extended to interconnected networks. Hybrid models could be introduced to combine black- and gray-box processors, allowing DDSP blocks with known functions alongside neural networks for learning functions. Moreover, community contributions could help expand our framework in various ways, including new architectures, loss functions, metrics, and more.

5 Acknowledgments

Funded by UKRI and EPSRC - grant EP/S022694/1.

References

[1] Wilmering, T., Moffat, D., Milo, A., and Sandler, M. B., "A history of audio effects," *Applied Sciences*, 10(3), 2020.

[2] Engel, J., Hantrakul, L., Gu, C., and Roberts, A., "DDSP: Differentiable digital signal processing," *arXiv preprint arXiv:2001.04643*, 2020.

[3] Comunita, M. and Reiss, J. D., "AFxResearch: a repository and website of audio effects research," in *DMRN+ 19: Digital Music Research Network One-day Workshop 2024*, 2024.

[4] Comunità, M., Stowell, D., and Reiss, J. D., "Guitar Effects Recognition and Parameter Estimation With Convolutional Neural Networks," *Journal of the Audio Engineering Society*, 69(7/8), 2021.

[5] Colonel, J. T., Steinmetz, C. J., Michelen, M., and Reiss, J. D., "Direct design of biquad filter cascades with deep learning by sampling random polynomials," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022.

[6] Mitcheltree, C., Steinmetz, C. J., Comunità, M., and Reiss, J. D., "Modulation extraction for LFO-driven audio effects," *arXiv preprint arXiv:2305.13262*, 2023.

[7] Comunità, M., Steinmetz, C. J., Phan, H., and Reiss, J. D., "Modelling black-box audio effects with time-varying feature modulation," in *2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.

Table 2: Parametric models included in the experiments. Cond. = conditioning method, R.F. = receptive field in samples. PEQ = Parametric EQ, G = Gain, O = Offset, MLP = Multilayer Perceptron, RNL = Rational Non Linearity. Controllers: .s = static, .d = dynamic, .sc = static conditional, .dc = dynamic conditional

Model	Cond.	R.F.	Blocks	Kernel	Dilation	Channels	# Params	FLOP/s	MAC/s
TCN-F-45-S-16	FiLM	2047	5	7	4	16	15.0k	736.5M	364.3M
TCN-TF-45-S-16	TFiLM	2047	5	7	4	16	42.0k	762.8M	364.2M
TCN-TTF-45-S-16	TTFiLM	2047	5	7	4	16	17.3k	744.0M	367.4M
TCN-TVF-45-S-16	TVFiLM	2047	5	7	4	16	17.7k	740.4M	366.2M

Model	Cond.	R.F.	Blocks	State Dimension	Channels	# Params	FLOP/s	MAC/s
S4-F-S-16	FiLM	-	4	4	16	8.9k	135.2M	53.8M
S4-TF-S-16	TFiLM	-	4	4	16	30.0k	155.6M	53.8M
S4-TTF-S-16	TTFiLM	-	4	4	16	10.2k	141.0M	56.3M
S4-TVF-S-16	TVFiLM	-	4	4	16	11.6k	138.9M	55.3M

Model	Signal Chain	# Params	FLOP/s	MAC/s
GB-C-DIST-MLP	PEQ.sc → G.sc → O.sc → MLP → G.sc → PEQ.sc	4.5k	202.8M	101.4M
GB-C-DIST-RNL	PEQ.sc → G.sc → O.sc → RNL → G.sc → PEQ.sc	2.3k	920.5k	4.3k
GB-C-FUZZ-MLP	PEQ.sc → G.sc → O.dc → MLP → G.sc → PEQ.sc	4.2k	202.8M	101.4M
GB-C-FUZZ-RNL	PEQ.sc → G.sc → O.dc → RNL → G.sc → PEQ.sc	2.0k	988.9k	3.6k

[8] Simionato, R. and Fasciani, S., "Comparative Study of Recurrent Neural Networks for Virtual Analog Audio Effects Modeling," *arXiv preprint arXiv:2405.04124*, 2024.

[9] Steinmetz, C. J., Bryan, N. J., and Reiss, J. D., "Style Transfer of Audio Effects with Differentiable Signal Processing," *Journal of the Audio Engineering Society*, 70(9), 2022.

[10] Steinmetz, C. J., Singh, S., Comunità, M., Ibnyahya, I., Yuan, S., Benetos, E., and Reiss, J. D., "St-ito: Controlling audio effects for style transfer with inference-time optimization," *arXiv preprint arXiv:2410.21233*, 2024.

[11] Steinmetz, C. J., Pons, J., Pascual, S., and Serra, J., "Automatic multitrack mixing with a differentiable mixing console of neural audio effects," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021.

[12] Sai Vanka, S., Safi, M., Rolland, J.-B., and Fazekas, G., "Adoption of AI Technology in Music Mixing Workflow: An Investigation," in *Audio Engineering Society Convention 154*, 2023.

[13] Wright, A., Damskägg, E.-P., and Välimäki, V., "Real-time black-box modelling with recurrent neural networks," in *Proc. of the Int. Conf. on Digital Audio Effects (DAFx-19)*, 2019.

[14] Steinmetz, C. J. and Reiss, J. D., "Efficient neural networks for real-time modeling of analog dynamic range compression," in *Audio Engineering Society Convention 152*, 2022.

[15] Yeh, Y.-T., Hsiao, W.-Y., and Yang, Y.-H., "Hyper recurrent neural network: Condition mechanisms for black-box audio effect modeling," *arXiv preprint arXiv:2408.04829*, 2024.

[16] Colonel, J. T., Comunità, M., and Reiss, J., "Reverse engineering memoryless distortion effects with differentiable waveshapers," in *Audio Engineering Society Convention 153*, 2022.

[17] Wright, A. and Valimaki, V., "Grey-box modelling of dynamic range compression," in *Proc. of the Int. Conf. on Digital Audio Effects (DAFx-20in22)*, 2022.

[18] Carson, A., King, S., Botinhao, C. V., and Bilbao, S., "Differentiable grey-box modelling of phaser effects using frame-based spectral processing," in *Proceedings of the 26th International Conference on Digital Audio Effects*, 2023.

[19] Miklanek, S., Wright, A., Välimäki, V., and Schimmel, J., "Neural grey-box guitar amplifier modelling with limited data," in *Proc. of the Int. Conf. on Digital Audio Effects (DAFx-23)*, 2023.

[20] Yeh, Y.-T., Chen, Y.-H., Cheng, Y.-C., Wu, J.-T., Fu, J.-J., Yeh, Y.-F., and Yang, Y.-H., "DDSP Guitar Amp: Interpretable Guitar Amplifier Modeling," *arXiv preprint arXiv:2408.11405*, 2024.

[21] Uzrad, N., Barkan, O., Elharar, A., Shvartzman, S., Laufer, M., Wolf, L., and Koenigstein, N., "DiffMoog: a Differentiable Modular Synthesizer for Sound Matching," *arXiv preprint arXiv:2401.12570*, 2024.

[22] Lee, S., Martínez-Ramírez, M., Liao, W.-H., Uhlich, S., Fabbro, G., Lee, K., and Mitsufuji,

Y., "GRAFX: an open-source library for audio processing graphs in PyTorch," *arXiv preprint arXiv:2408.03204*, 2024.

[23] Yeh, Y.-T., Hsiao, W.-Y., and Yang, Y.-H., "PyNeuralFx: A Python Package for Neural Audio Effect Modeling," *arXiv preprint arXiv:2408.06053*, 2024.

[24] Steinmetz, C. J. and Reiss, J. D., "auraloss: Audio focused loss functions in PyTorch," in *Digital music research network one-day workshop (DMRN+ 15)*, 2020.

[25] Kilgour, K., Zuluaga, M., Roblek, D., and Sharifi, M., "Fr\'echet audio distance: A metric for evaluating music enhancement algorithms," *arXiv preprint arXiv:1812.08466*, 2018.

[26] Novak, A., Simon, L., Kadlec, F., and Lotton, P., "Nonlinear system identification using exponential swept-sine signal," *IEEE Transactions on Instrumentation and Measurement*, 59(8), 2009.

[27] Wright, A., Damskägg, E.-P., Juvela, L., and Välimäki, V., "Real-time guitar amplifier emulation with deep learning," *Applied Sciences*, 10(3), 2020.

[28] Wright, A. and Valimaki, V., "Neural modeling of phaser and flanging effects," *Journal of the Audio Engineering Society*, 69(7/8), 2021.

[29] Lea, C., Vidal, R., Reiter, A., and Hager, G. D., "Temporal convolutional networks: A unified approach to action segmentation," in *Computer Vision–ECCV Workshops*, Springer, 2016.

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

[31] Steinmetz, C. J., "Learning to mix with neural audio effects in the waveform domain," *MS thesis*, 2020.

[32] Stein, M., Abeßer, J., Dittmar, C., and Schuller, G., "Automatic detection of audio effects in guitar and bass recordings," in *Audio Engineering Society Convention 128*, 2010.

[33] Rethage, D., Pons, J., and Serra, X., "A wavenet for speech denoising," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

[34] Damskägg, E.-P., Juvela, L., and Välimäki, V., "Real-time modeling of audio distortion circuits with deep learning," in *Sound and music computing conference*, 2019.

[35] Gu, A., Goel, K., and Ré, C., "Efficiently modeling long sequences with structured state spaces," *arXiv preprint arXiv:2111.00396*, 2021.

[36] Yin, H., Cheng, G., Steinmetz, C. J., Yuan, R., Stern, R. M., and Dannenberg, R. B., "Modeling Analog Dynamic Range Compressors using Deep Learning and State-space Models," *arXiv preprint arXiv:2403.16331*, 2024.

[37] Simionato, R. and Fasciani, S., "Modeling Time-Variant Responses of Optical Compressors with Selective State Space Models," 2024.

[38] Gupta, A., Gu, A., and Berant, J., "Diagonal state spaces are as effective as structured state spaces," *Advances in Neural Information Processing Systems*, 35, 2022.

[39] Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A., "Film: Visual reasoning with a general conditioning layer," in *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[40] Birnbaum, S., Kuleshov, V., Enam, Z., Koh, P. W. W., and Ermon, S., "Temporal FiLM: Capturing long-range sequence dependencies with feature-wise modulations." *Advances in Neural Information Processing Systems*, 32, 2019.

[41] Nercessian, S., "Neural parametric equalizer matching using differentiable biquads," 2020.

[42] Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G., "Implicit neural representations with periodic activation functions," *Advances in neural information processing systems*, 33, 2020.

[43] Baker Jr, G. A. and Gammel, J. L., "The Padé approximant," *Journal of Mathematical Analysis and Applications*, 2(1), 1961.

[44] Molina, A., Schramowski, P., and Kersting, K., "Padé activation units: End-to-end learning of flexible activation functions in deep networks," *arXiv preprint arXiv:1907.06732*, 2019.

[45] Comunità, M., Steinmetz, C. J., and Reiss, J. D., "Differentiable black-box and gray-box modeling of nonlinear audio effects," *arXiv preprint arXiv:2502.14405*, 2025.