AUDIO EFFECT CHAIN ESTIMATION AND DRY SIGNAL RECOVERY FROM MULTI-EFFECT-PROCESSED MUSICAL SIGNALS

Osamu Take[1], *Kento Watanabe*[2], *Takayuki Nakatsuka*[2], *Tian Cheng*[2], *Tomoyasu Nakano*[2], *Masataka Goto*[2], *Shinnosuke Takamichi*[3,1], *and Hiroshi Saruwatari*[1]

[1]Graduate School of Information Science and Technology, University of Tokyo, Tokyo, Japan
[2]National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba, Japan
[3]Faculty of Science and Technology, Keio University, Yokohama, Japan
flymoons0325@g.ecc.u-tokyo.ac.jp

ABSTRACT

In this paper we propose a method that can address a novel task, audio effect (AFX) chain estimation and dry signal recovery. AFXs are indispensable in modern sound design workflows. Sound engineers often cascade different AFXs (as an AFX chain) to achieve their desired soundscapes. Given a multi-AFX-applied solo instrument performance (wet signal), our method can automatically estimate the applied AFX chain and recover its unprocessed dry signal, while previous research only addresses one of them. The estimated chain is useful for novice engineers in learning practical usages of AFXs, and the recovered signal can be reused with a different AFX chain. To solve this task, we first develop a deep neural network model that estimates the last-applied AFX and undoes its AFX at a time. We then iteratively apply the same model to estimate the AFX chain and eventually recover the dry signal from the wet signal. Our experiments on guitar phrase recordings with various AFX chains demonstrate the validity of our method for both the AFX-chain estimation and dry signal recovery. We also confirm that the input wet signal can be reproduced by applying the estimated AFX chain to the recovered dry signal.

1. INTRODUCTION

Audio effects (*AFXs*) have played an essential role in modern music composition, live performance, and recording scenes [1]. Each type of AFXs (e.g., reverb, distortion) possesses unique properties [2], and sound engineers utilize these properties to create desired soundscapes by adjusting their control parameters. In practical sound design, it is common to sequentially apply multiple AFXs to certain musical audio signals [3]. This sequence of AFXs is referred to as the *audio effect chain* (*AFX chain*), defined by the selection of AFXs, their control parameter values, and the order in which AFXs are applied. Figure 1 illustrates the process of applying an AFX chain to an unprocessed musical signal (*dry* signal). Sound engineers focus on designing AFX chains to ensure that musical signals (*wet* signals) blend well with captivating songs or performances. They often draw inspiration from sonic characteristics of musical signals in existing tracks for their own sound design. Research about musical-synthesizer sound matching [4, 5] and AFX-chain recommendation [6] aim at supporting these styles of sound design.

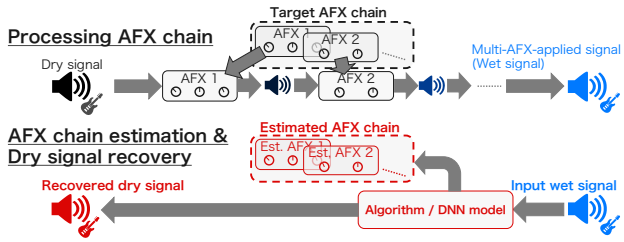


Figure 1: *In the common sound-design process, sound engineers design a (target) AFX chain and apply each AFX to the source (dry) signal with the designed order, from left to right as depicted in the upper figure. The lower figure illustrates our task of AFX-chain estimation and dry signal recovery, taking the multi-AFX-applied (wet) signal as the input to estimate the applied AFX chain and recover the dry signal.*

In general, it is easy to apply an AFX chain to a dry signal as engineers usually do, but it is not easy to recover the dry signal from an existing AFX-applied wet signal by undoing its AFX chain because it is a kind of "reverse-engineering" process. If we could achieve this inverse process by automatically estimating the applied AFX chain, it could be useful in streamlining sound design processes and providing support for novice sound engineers in understanding the practical use of various AFXs.

We therefore introduce a novel task "AFX-chain estimation and dry signal recovery" and propose a method to solve it. This task first takes a multi-AFX-applied musical audio signal (wet signal) as input. It then estimates the applied AFX-chain configuration (types, control parameters, and the order) from the wet signal and recovers the dry signal, as illustrated in Figure 1. To the best of our knowledge, previous research has focused solely on either AFX-chain estimation or dry signal recovery, without addressing their combined task. The key idea of our proposed method is to iteratively apply the "inverse process" of the single "universal" AFX application. Each iteration undoes the last-applied AFX and estimates its type and parameters. If three AFXs are applied, for example, we could repeat its inverse process three times to eventually estimate the whole AFX chain and recover the dry signal. In its repetition, we also challenge to estimate the number of the applied AFXs, given the wet signal only.

We conducted experiments on our proposed method for AFX-chain estimation and dry signal recovery, targeting guitar phrase recordings where sound design with multiple AFXs is frequently employed. We also conducted an ablation study to confirm that key elements of our proposed method are effective. We finally investigated whether the input wet signal could be reproduced by

applying the estimated AFX chain to the recovered dry signals, demonstrating the potential of our method as a useful sound-design tool. Sound examples are available[1].

2. RELATED WORK

2.1. Deep-learning-based Audio Effect Manipulation

Deep neural networks (DNNs) have been widely utilized for manipulating AFXs because they are highly expressive and flexible compared to rule-based methods. For example, DNNs can emulate existing AFXs with black-box data-driven approaches [7, 8, 9]. In automatic mixing [10] and music-mixing style transfer [11], AFXs are incorporated into DNNs as differentiable modules to improve the performance and make the inference interpretable for humans. We also take the advantage of DNNs to model the single AFX inversion process in a black-box data-driven way, but the overall architecture of repeating its process in a cascading manner is a white box, as we dare not take an untransparent end-to-end approach.

2.2. Audio Effect Chain Estimation

The estimation of AFXs (types and control parameters) using DNNs has been investigated for single [12] and multiple [13, 14, 15, 16] AFX settings. For multi-AFX-applied wet signals, the estimation of AFX-chain configuration with the fixed types of AFXs and the fixed order was investigated [13]. For scenarios involving varying numbers of AFXs in the target chain, some research addresses the estimation of AFX categories used in the AFX chain [14, 15]. Moreover, graph neural networks are employed to estimate the configuration of complex AFX-processing graphs [16]. These graphs encompass not only chain cascades but also parallel processing and audio busses. However, these methods do not recover the dry signal by undoing the applied multiple AFXs. They thus have the limitation of not being able to apply the estimated AFX chain to the dry signal to reproduce the wet signal. This limitation does not allow sound engineers to redesign the given wet signal (i.e., obtain a different sound) by changing some parameters of the estimated AFXs. Our method has the advantage of not having this limitation and enables such redesign.

2.3. Dry Signal Recovery

Conventional methods of dry signal recovery were based on signal-processing manipulations, aiming to undo only the one specific AFX such as reverb [17] and dynamic range compression [18]. Recent DNN-based methods have also tackled the inversion process of a single specific AFX such as distortion [19], limiter [20], and reverb [21]. To recover the dry signal from a multi-AFX-applied signal, a compositional method that detects all AFXs applied and evokes each AFX-wise removal network was proposed [22]. However, since these methods do not estimate the AFX control parameters, they cannot reuse the recovered dry signal to reproduce the wet signal and redesign it as exemplified in the last two sentences of Section 2.2. Furthermore, the above compositional method [22] does not estimate the order of AFXs, which can deteriorate the performance in recovery due to non-linear, time-varying AFXs. Our method does not have such drawbacks.

[1] `https://sarulab-speech.github.io/afxchest-dryrec`

3. AUDIO EFFECT CHAIN ESTIMATION AND DRY SIGNAL RECOVERY

3.1. Task Formulation

First we formulate the process of applying a single AFX to the input dry signal to obtain the wet signal as follows:

$$\boldsymbol{y} = f_{c,\boldsymbol{p}_c}(\boldsymbol{x}), \tag{1}$$

where the dry signal $\boldsymbol{x} \in \mathbb{R}^{2\times T}$ and the wet signal $\boldsymbol{y} \in \mathbb{R}^{2\times T}$ are both stereo signals (2 channels) with a fixed sample length T. The function f represents the application of the AFX, and its form is determined by the AFX type denoted by a discrete label c and its corresponding control parameter values $\boldsymbol{p}_c$. The tuple of these two variables $(c, \boldsymbol{p}_c)$ represents the configuration of the AFX. The AFX type c is chosen from a predefined set $\mathcal{C}$ (e.g., $\mathcal{C} = \{\mathrm{compressor}, \mathrm{EQ}, \cdots\}$), meaning that the set of realizations for c is predetermined and finite. The number of control parameters, i.e., the dimension of $\boldsymbol{p}_c$, varies for each AFX type c, and all control parameters are assumed to have continuous values.

An AFX chain can be represented as a composite function F_Λ that applies multiple AFXs sequentially:

$$\boldsymbol{y} = F_\Lambda(\boldsymbol{x}) = f_{c_n,\boldsymbol{p}_{c_n}^{(n)}} \circ \cdots f_{c_2,\boldsymbol{p}_{c_2}^{(2)}} \circ f_{c_1,\boldsymbol{p}_{c_1}^{(1)}}(\boldsymbol{x}), \tag{2}$$

$$\Lambda = ((c_i, \boldsymbol{p}_{c_i}^{(i)}))_{i=1}^{n}, \tag{3}$$

where n denotes the length of the AFX chain, indicating the total number of AFXs applied to the dry signal. The sequence Λ represents the whole configuration of an AFX chain, determined by the number of AFXs n, their types c_i, control parameters $\boldsymbol{p}_{c_i}^{(i)}$, and their order. The superscript i in $\boldsymbol{p}_{c_i}^{(i)}$ represents that it is the control parameter values of i-th applied AFX in Λ, and the subscript c_i represents the AFX type only, not including the information about the order of AFXs.

Our task takes a wet signal $\boldsymbol{y}$ as input, and then estimates the applied AFX-chain configuration Λ and recovers the AFX-chain-unprocessed dry signal $\boldsymbol{x}$. We denote the estimates of these two outputs as $\hat{\Lambda}$ and $\hat{\boldsymbol{x}}$. We can regard this task as the AFX chain inverse modeling, which aims to obtain functions H_{chain} and H_{sig} that can approximate F_Λ^{-1}:

$$\boldsymbol{x} = F_\Lambda^{-1}(\boldsymbol{y}) = f^{-1}_{c_1,\boldsymbol{p}_{c_1}^{(1)}} \circ \cdots f^{-1}_{c_2,\boldsymbol{p}_{c_2}^{(2)}} \circ f^{-1}_{c_n,\boldsymbol{p}_{c_n}^{(n)}}(\boldsymbol{y}), \tag{4}$$

$$\hat{\Lambda}(= ((\hat{c}_i, \hat{\boldsymbol{p}}_{\hat{c}_i}^{(i)}))_{i=1}^{\hat{n}}) = H_{\mathrm{chain}}(\boldsymbol{y}), \quad \hat{\boldsymbol{x}} = H_{\mathrm{sig}}(\boldsymbol{y}). \tag{5}$$

Here, $\hat{c}_i, \hat{\boldsymbol{p}}_{\hat{c}_i}^{(i)}$, and $\hat{n}$ denote the estimates of the AFX-chain configuration, corresponding to variables in Eq. (3). This task thus requires that the length of AFX chain, $\hat{n}$, be estimated as well.

Note that the inverse of AFX f^{-1} does not always exist in Eq. (4). For example, if an AFX f is a kind of non-injective mappings such as hard-clipping distortion or noise gate [2], its inverse function does not exist. From the viewpoint of information loss, this means that it is impossible to recover their dry signals from the wet signals only. We therefore utilize DNNs for this task, expecting their data-driven approach to capture a plausible mapping from the output to the input for such non-injective AFXs.

3.2. Proposed Method

Figure 2 illustrates the overall architecture of our proposed method for AFX-chain estimation and dry signal recovery. This method

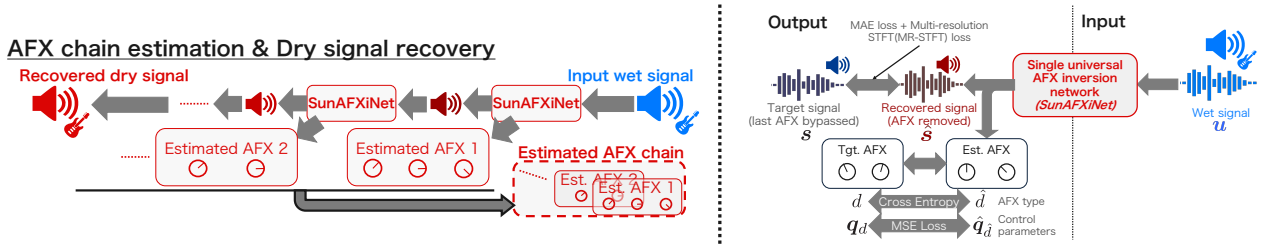


Figure 2: *Overview of our proposed method for AFX-chain estimation and dry signal recovery task. The method illustrated in the left figure applies the "inverse process" of AFX-chain processing to the existing musical (wet) signal. It repeatedly applies the single DNN model (SunAFXiNet, detailed in the right figure) to construct the estimated AFX chain from right to left.*

does not determine H_{chain} and H_{sig} in an end-to-end manner. Instead, starting from the wet signal $\boldsymbol{y}$ as input, this method applies an identical DNN model iteratively to obtain the entire AFX-chain estimate $\hat{\Lambda}$ and the recovered dry signal $\hat{\boldsymbol{x}}$. We also propose this identical DNN model named as *SunAFXiNet*[2]. With input as the signal $\boldsymbol{u} \in \mathbb{R}^{2\times T}$, SunAFXiNet estimates the configuration $(d, \boldsymbol{q}_d)$ of the last-applied AFX to $\boldsymbol{u}$ and outputs the signal $\hat{\boldsymbol{s}} \in \mathbb{R}^{2\times T}$ approximating the signal bypassing (undoing) the estimated AFX from $\boldsymbol{u}$. SunAFXiNet consists of an AFX estimation block h_{afx} and a dry signal recovery block h_{sig}, named respectively as *AFX config estimator* and *bypassed signal estimator*. Each block works as follows:

$$(\hat{d}, \hat{\boldsymbol{q}}_{\hat{d}}) = h_{\text{afx}}(\boldsymbol{u}), \tag{6}$$

$$\hat{\boldsymbol{s}} = h_{\text{sig}}(\boldsymbol{u}, (\hat{d}, \hat{\boldsymbol{q}}_{\hat{d}})). \tag{7}$$

Here, we do not prepare an individual bypassed signal estimator for each AFX type. Instead, we train our bypassed signal estimator h_{sig} as the universal inverse model that can cover all types of single AFX f, supporting an arbitrary configuration (type $\hat{d}$ and control parameters $\hat{\boldsymbol{q}}_{\hat{d}}$) estimated by the AFX config estimator h_{afx}.

The procedure of the iterative SunAFXiNet application is described as pseudo-code in Algorithm 1. We iteratively apply h_{afx} and h_{sig} to the wet signal $\boldsymbol{y}$ (in the initial iteration) or the output signal $\hat{\boldsymbol{s}}$ (in subsequent iterations) to reconstruct the entire AFX chain $\hat{\Lambda}$ backwards. Since this output signal $\hat{\boldsymbol{s}}$ after each iteration is the bypassed signal, we can also undo the entire AFX chain and obtain the final dry signal $\hat{\boldsymbol{x}}$ after all the iterations.

The remaining problem is to estimate the AFX-chain length n that corresponds to the number of iterations. Algorithm 1 also depicts the procedure for stopping the iterative process. Here, we simply stop if the difference $\mathcal{M}(\hat{\boldsymbol{s}}, \boldsymbol{u})$ between the input $\boldsymbol{u}$ and output $\hat{\boldsymbol{s}}$ of SunAFXiNet is smaller than a certain threshold. If no AFX was applied to the input, SunAFXiNet cannot undo anything, and the output will be almost the same, resulting in a small $\mathcal{M}(\hat{\boldsymbol{s}}, \boldsymbol{u})$. Therefore, the number of iterations until just before this stop will be the length $\hat{n}$ for the estimated configuration $\hat{\Lambda}$.

We take this iterative approach since we expect it to have better training efficiency and performance compared to an approach that estimates the AFX-chain configuration all at once. This expectation is in line with the literature [22] that deals with the task focusing solely on the dry signal recovery and reported that higher performance was achieved by removing one estimated AFX at a time rather than removing multiple AFXs simultaneously. Furthermore, in terms of practical applications, it is noteworthy that

[2]Abbreviation of "**S**ingle **uni**versal **AFX** **i**nversion **Net**work."

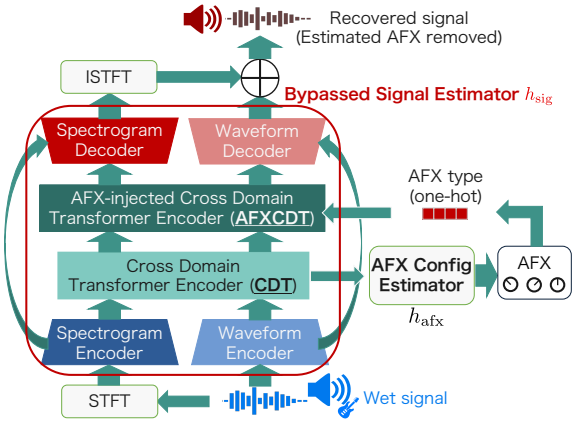
Algorithm 1 The procedure of our proposed method for AFX-chain estimation and dry signal recovery.

$\hat{\Lambda} \leftarrow [\,], \hat{X} \leftarrow [\boldsymbol{y}], \hat{n} \leftarrow 0$
$\boldsymbol{u} \leftarrow \boldsymbol{y}$: the input wet signal
flag $\leftarrow true$
while flag **do**
  $(\hat{d}, \hat{\boldsymbol{q}}_{\hat{d}}) = h_{\text{afx}}(\boldsymbol{u})$, $\hat{\boldsymbol{s}} = h_{\text{sig}}(\boldsymbol{u}, (\hat{d}, \hat{\boldsymbol{q}}_{\hat{d}}))$
  if the stop criterion $\mathcal{M}(\hat{\boldsymbol{s}}, \boldsymbol{u})$ is smaller than the threshold **then**
    flag $\leftarrow false$
  else
    $\hat{\Lambda}.append((\hat{d}, \hat{\boldsymbol{q}}_{\hat{d}}))$, $\hat{X}.append(\hat{\boldsymbol{s}})$
    $\boldsymbol{u} \leftarrow \hat{\boldsymbol{s}}$
    $\hat{n} \leftarrow \hat{n} + 1$
  end if
end while
Reverse the order of $\hat{\Lambda}$
Recovered dry signal $\hat{\boldsymbol{x}} \leftarrow \hat{X}[-1]$

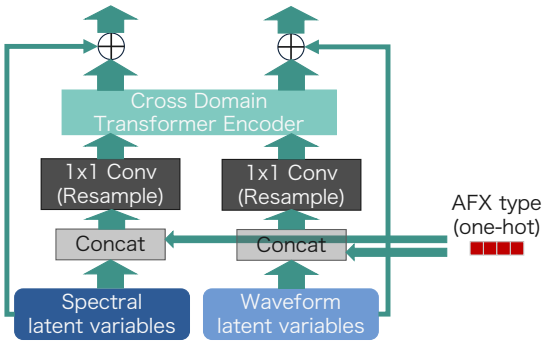
the proposed method can provide not only the final dry signal $\hat{\boldsymbol{x}}$, but also the intermediate bypassed signals $\hat{X}$ obtained after each iteration. This paves the way for a new approach to interactive and innovative sound design, making it easier to reuse and reconstruct AFX chains from existing AFX-applied audio signals. For example, sound engineers can try different configurations of subsequent AFXs or different lengths n of the AFX chain on any intermediate bypassed signal in a trial and error manner, or apply the estimated AFX chain to any other sound.

3.3. Single Universal Audio Effect Inversion Network (SunAFXiNet)

Figure 3 depicts the overview of SunAFXiNet used in our proposed method. The bypassed signal estimator h_{sig} utilizes and extends Hybrid Transformer Demucs (HTDemucs) [23]. HTDemucs is a model that extends the wave-to-wave encoder-decoder model [24, 25], which has demonstrated high performance in music source separation, by incorporating a Transformer encoder in the bottleneck. We employ this model in dry signal recovery, as presented in related work [9, 19, 22]. Our extension adds an AFX-injected cross domain Transformer encoder (AFXCDT), depicted in Figure 3b, after the cross domain Transformer encoder (CDT) existing in the bottleneck of HTDemucs. AFXCDT is designed to condition the original CDT by leveraging the configuration of the estimated AFX.



(a) *Overall architecture*



(b) *Detailed structure of AFX-injected cross domain Transformer encoder (AFXCDT)*

Figure 3: *Architecture of single universal AFX inversion network (SunAFXiNet) based on Hybrid Transformer Demucs [23].*

3.3.1. AFX Config Estimator

The AFX config estimator h_{afx} in Figure 3a takes the latent representation output by CDT as input to estimate the configuration $(\hat{d}, \hat{\boldsymbol{q}}_{\hat{d}})$ of the AFX last-applied to the wet signal $\boldsymbol{u}$. The AFX config estimator not only estimates the control parameter values $\hat{\boldsymbol{q}}_{\hat{d}}$ corresponding to the estimated type of AFX $\hat{d}$, but also outputs the control parameter values $\boldsymbol{q}_c$ for all possible types of AFX $c \in \mathcal{C}$, assuming each type is applied last. This enables the AFX config estimator to compare the target values $\boldsymbol{q}_d$ with the corresponding estimated values $\hat{\boldsymbol{q}}_d$, even if the type $\hat{d}$ estimated by the AFX config estimator is incorrect (i.e., differs from the target type d).

3.3.2. AFXCDT and One-hot Vector of AFX Type

In AFXCDT, a one-hot vector representing the estimated AFX type $\hat{d}$ is then concatenated with the latent representation obtained from the CDT. Since $\hat{d}$ is represented as the probability distribution $P : \mathcal{C} \to [0, 1]$, it is transformed into the one-hot vector in which the AFX type with the highest probability is 1 and the others are 0. After the concatenation, AFXCDT passes the concatenated values to another internal CDT as shown in Figure 3b.

Here, we dared not concatenate the AFX control parameter values because we observed a sharp decline in the performance of dry signal recovery when conditioning on the AFX control parameter values in a preliminary experiment.

3.3.3. Training process of SunAFXiNet

We divide the training of SunAFXiNet into two separate stages. In the first stage, we train the bypassed signal estimator h_{sig} solely, without estimating the AFX configuration. During this stage, we condition h_{sig} using the ground-truth target type d of the last-applied AFX. Then, the second stage freezes the model parameters of h_{sig} and trains the AFX config estimator exclusively. This training scheme enables the SunAFXiNet to avoid unstable training, which was observed in a preliminary experiment in which the entire model was trained at once.

The details of the SunAFXiNet model architecture and training scheme used in experiments are described in Section 4.1.

4. EXPERIMENTAL EVALUATION

We evaluated our method on guitar phrase recordings since they are typical musical audio signals in which AFXs play an important role. We also conducted an ablation study of disabling CDT and AFXCDT in the SunAFXiNet model to understand their effectiveness.

4.1. Experimental Setup

4.1.1. Dataset

We collected GuitarSet [26], IDMT-SMT-Guitar [27], and guitar tracks in Slakh2100 [28] as raw guitar recordings to construct the dataset. We resampled all the raw recordings (stereo audio signals) at a sampling frequency of 48 kHz and trimmed silences longer than one second on the basis of the predetermined volume threshold. Then, from each recording, a random 10-second segment was taken as the source signal, resulting in 5975 source signals.

We selected four commonly-used AFXs in guitar sound design as the possible set $\mathcal{C}$ of AFXs for the target AFX chain:

$$\mathcal{C} = \{\texttt{distortion}, \texttt{delay}, \texttt{chorus}, \texttt{reverb}\}. \quad (8)$$

All of these AFXs sourced from the built-in plugins in `pedalboard` [29], a Python audio-processing library. For each AFX, we manually determined one to five representative control parameters and their ranges of values, considering practical use. All control parameters were randomly set within these ranges. We finally generated $\sum_{k=1}^{4} {}_4P_k = 64$ possible AFX chains in which each of the four AFX types appears at most once, considering its order.

We applied each of the 64 generated AFX chains to each of the 5975 source signals to generate 382400 wet signals. For each source signal, we used different random control parameters. We then prepared 382400 dataset entries for training and testing SunAFXiNet. Since we are interested in the last-applied AFX only, each dataset entry consists of the wet signal $\boldsymbol{u}$ as input, the configuration $(d, \boldsymbol{q}_d)$ of the last-applied AFX, and the signal $\boldsymbol{s}$ before its AFX as output. Some data entries from the same source signal were prepared by sharing some AFXs as depicted in Figure 4. We normalized all control parameters within the range of $[0, 1]$. We did not normalize musical audio signals because their amplitudes matter for modeling AFXs. To avoid information loss due to clipping, all musical signals were stored in 32-bit float PCM format.

We split the 382400 dataset entries into train/valid/test sets (287424/58368/36608 entries corresponding to 4491/572/912 source signals) with no overlap in instruments of source signals.

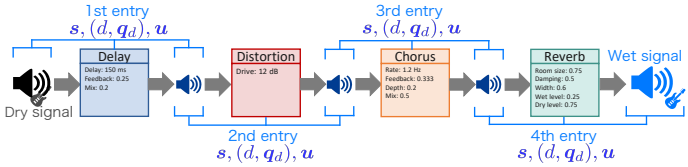


Figure 4: *Dataset entry examples for SunAFXiNet training. Each blue line illustrates the input-output data for each entry. As a result, a source signal yielded 64 entries from 64 different AFX chains. This figure illustrates four entries, the first entry from the chain of delay only, the second entry from the chain of delay and distortion, the third entry from the chain of delay, distortion, and chorus, and the fourth entry from the chain of all the four AFXs.*

4.1.2. Model Configurations

In this experiment, we adopt the training and hyperparameter settings from previous research [23] to train the HTDemucs-based SunAFXiNet model. We used a window length of 8192 samples for the STFT applied to the input of the spectrogram encoder in the bypassed signal estimator. The output channel of the first layer of the waveform/spectrogram encoder was set to 32. All encoders and decoders were configured with 6 layers each. CDT and AFXCDT were constructed with 3 and 2 cross-domain Transformer layers, respectively.

The AFX config estimator was constructed with 3 layers of convolutional blocks and 3 layers of fully connected blocks. The convolutional block consisted of a one-dimensional convolutional layer along the time axis (kernel size 4, stride 2), followed by one-dimensional batch normalization and ReLU. For the output of the final layer, the mean and maximum along the time axis were computed, and their sum was passed as input to the subsequent fully connected block. The fully connected block was separately configured for estimating the AFX type d and control parameter values $\boldsymbol{q}_d$. The fully connected block consisted of fully connected layers followed by batch normalization and ReLU. All fully connected layers in the AFX config estimator was set to have 512 hidden dimensions. Additionally, a dropout with probabilities of 0.2 and 0.05 was applied during training to the fully connected blocks responsible for estimating the AFX type and control parameter values, respectively. We called this SunAFXiNet model as "Proposed" below.

Models for ablation study. We prepared two other models, "noAFXCDT" and "noCDT", for the ablation study to investigate the functionality of CDT and AFXCDT. The noAFXCDT model did not include AFXCDT, and instead, CDT was composed of 5 layers of Transformer layers. With this model, we can confirm whether the AFXCDT is effective. In noCDT, CDT and AFXCDT were composed of 0 and 5 layers of Transformer layers, respectively. This setup allows us to investigate performance changes due to the presence of Transformer layers in CDT. In Proposed and noAFXCDT, the input to the AFX config estimator is the output of CDT, while in noCDT, the input is the concatenation of the encoder outputs.

Training configurations. The first and second stages of training SunAFXiNet used the objectives $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively:

$$\mathcal{L}_1 = \mathcal{L}_{\mathrm{MAE}}(\hat{\boldsymbol{s}}(\boldsymbol{u}, d), \boldsymbol{s}) + 0.05\mathcal{L}_{\mathrm{mrstft}}(\hat{\boldsymbol{s}}(\boldsymbol{u}, d), \boldsymbol{s}), \quad (9)$$

$$\mathcal{L}_2 = \mathcal{L}_{\mathrm{ce}}(\hat{d}(\boldsymbol{u}), d) + \mathcal{L}_{\mathrm{MSE}}(\hat{\boldsymbol{q}}_d(\boldsymbol{u}), \boldsymbol{q}_d), \quad (10)$$

where $\mathcal{L}_{\mathrm{MAE}}, \mathcal{L}_{\mathrm{mrstft}}, \mathcal{L}_{\mathrm{ce}}$, and $\mathcal{L}_{\mathrm{MSE}}$ denote the mean average error (MAE), multi-resolution STFT loss [30], cross-entropy loss, and the mean squared error (MSE), respectively. We used the `auraloss` [31] implementation to compute $\mathcal{L}_{\mathrm{mrstft}}$. The coefficient value 0.05 before $\mathcal{L}_{\mathrm{mrstft}}$ was determined by our preliminary experiment. During the training of SunAFXiNet, we set the batch size to 32 and the learning rate to 5×10^{-5}. The number of epochs for the first and second stages were 400 and 150, respectively.

Table 1: *Results of AFX-bypassed signal $\boldsymbol{s}$ recovery.*

Model	**SI-SDR** (↑)	**MR-STFT** (↓)
Wet $\boldsymbol{u}$	10.57 dB	2.32
Proposed	**13.63 dB**	**0.68**
noCDT	13.30 dB	0.69
noAFXCDT	9.74 dB	1.06

Table 2: *Results of AFX configuration estimation.*

Model	**AFX Acc.**(↑)	**Param. MSE**(↓)
Proposed	0.695	0.034
noCDT	0.724	**0.030**
noAFXCDT	**0.728**	0.038

4.2. Evaluation Scheme and Metrics

The evaluation includes three parts. We first evaluated the performance of SunAFXiNet itself (Section 4.3) to confirm that the SunAFXiNet successfully estimates the last-applied AFX and undoes it to obtain the bypassed signal. The bypassed signal recovery was assessed by the scale-invariant SDR (SI-SDR) [32] and multi-resolution STFT loss (MR-STFT) between the target $\boldsymbol{s}$ and its estimate $\hat{\boldsymbol{s}}$. The evaluation metrics for the AFX estimation were the accuracy of AFX type estimation (AFX Acc.) as well as the MSE of the normalized control parameter values between the target $\boldsymbol{q}_d$ and its estimate $\hat{\boldsymbol{q}}_d$ for the target AFX d (Param. MSE).

We then investigated the performance of the dry signal recovery (Section 4.4). Given $\boldsymbol{y}$, we used the proposed method with the trained SunAFXiNet to obtain the dry signal estimate $\hat{\boldsymbol{x}}$ in Eq. (5), and assessed whether $\hat{\boldsymbol{x}}$ successfully approximates the target dry signal $\boldsymbol{x}$ in Eq. (4). The evaluation metrics of this recovery were SI-SDR and MR-STFT between the target $\boldsymbol{x}$ and its estimate $\hat{\boldsymbol{x}}$.

Finally we evaluated the reproducibility of the wet signal $\boldsymbol{y}$ from the estimated AFX-chain configuration $\hat{\Lambda}$ and dry signal $\hat{\boldsymbol{x}}$ (Section 4.5). We re-applied the estimated AFX chain $F_{\hat{\Lambda}}$ to the recovered dry signal $\hat{\boldsymbol{x}}$, and evaluated if the reproduced wet signal $F_{\hat{\Lambda}}(\hat{\boldsymbol{x}})$ successfully approximates the original input wet signal $\boldsymbol{y}$. The evaluation metrics of this wet signal reproduction were SI-SDR and MR-STFT, same as the dry signal recovery evaluation.

4.3. Evaluation of SunAFXiNet

Table 1 shows the results of the bypassed signal recovery. For the Proposed and noCDT models, the bypassed signal was recovered with the AFXCDT conditioned on the ground-truth target AFX type. For comparison, we added the "Wet" condition without any SunAFXiNet applied, which evaluates the input wet signal $\boldsymbol{u}$ as it is, as if it were the recovered signal (i.e., the estimate $\hat{\boldsymbol{s}} = \boldsymbol{u}$). The results showed that for the Proposed and noCDT models, the evaluation metrics improved compared to the Wet. Since the noAFXCDT model was the worst for SI-SDR, the use of AFXCDT conditioned on the AFX configuration was effective in this recovery.

Table 3: *Results of dry signal $\boldsymbol{x}$ recovery from AFX-chain-processed wet signal $\boldsymbol{y}$.*

Model	SI-SDR(↑)	MR-STFT(↓)
Wet $\boldsymbol{y}$	1.26 dB	5.66
Proposed	**5.81 dB**	**2.48**
noCDT	5.30 dB	2.53
noAFXCDT	2.09 dB	3.78

Table 2 shows the results of the AFX configuration estimation. All three models achieved comparable estimation performance, with a slight decline in the Proposed model performance.

4.4. Dry Signal Recovery from Wet Signal

We evaluated the proposed method with the trained SunAFXiNet for the performance of recovering the dry signal $\boldsymbol{x}$ from the wet signal $\boldsymbol{y}$. For each data entry in the test set, we input the wet signal $\boldsymbol{y}(=\boldsymbol{u})$ to the proposed method to estimate the AFX-chain configuration $\hat{\Lambda}$ and recover the dry signal $\hat{\boldsymbol{x}}$, which was then compared to the ground-truth dry signal $\boldsymbol{x}$ instead of $\boldsymbol{s}$ in the data entry. In this estimation, we ensured that each AFX type appeared at most once by masking, in the output distribution P of the AFX config estimator, the values of AFXs that had already appeared in $\hat{\Lambda}$ obtained in previous iterations.

For this evaluation, we need to determine the stop criteria (the threshold for $\mathcal{M}(\hat{\boldsymbol{s}}, \boldsymbol{u})$). We therefore analyzed distributions of the MAE difference Δ_{MAE} between the input and output waveforms of SunAFXiNet during its iterative application in the proposed method. The Δ_{MAE} is defined as follows:

$$\Delta_{\mathrm{MAE}} = \mathcal{L}_{\mathrm{MAE}}(\hat{\boldsymbol{s}}, \boldsymbol{u})/||\boldsymbol{u}||_1. \tag{11}$$

For this analysis, we iteratively applied SunAFXiNet to the wet signal $\boldsymbol{y}(=\boldsymbol{u})$ using 58368 data entries in the valid set. We did not employ any stop criterion but the above-mentioned AFX-type masking, ensuring that each of the four AFXs in $\mathcal{C}$ appeared exactly once. We thus collected 233472 $(= 58368 \times 4)$ values of Δ_{MAE}. Each estimated AFX type thus has 58368 difference values. To determine the threshold, it is necessary to visualize them so that we can compare the values when the estimated AFX is actually included in the ground-truth target chain (IN target chain) with the values when it is not (NOT IN target chain).

Figure 5 displays the violin plots of the distributions so that we can analyze the values for the above purpose. We show them for each of the three models, Proposed, noCDT, and noAFXCDT. For the Proposed and noCDT models, this figure clearly shows that, for each AFX type, the MAE difference between the input $\boldsymbol{u}$ and the output $\hat{\boldsymbol{s}}$ is larger when the AFX is correctly estimated ("IN target chain" plots shown in the blue color) than when it is not ("NOT IN target chain" plots shown in the orange color). This allows us to determine the threshold for $\mathcal{M}(\hat{\boldsymbol{s}}, \boldsymbol{u})(=\Delta_{\mathrm{MAE}})$ as the top 20 percentile of the "NOT IN target chain" distribution, as shown in the magenta line in Figure 5. Each estimated AFX type $\hat{d}$ in each model thus has a different threshold.

Table 3 shows the results of the dry signal recovery. The results confirmed that the proposed method successfully recovered the dry signal $\boldsymbol{x}$, compared to the Wet $\boldsymbol{y}$ condition. Among the three models, the Proposed model performed best in both SI-SDR and MR-STFT.

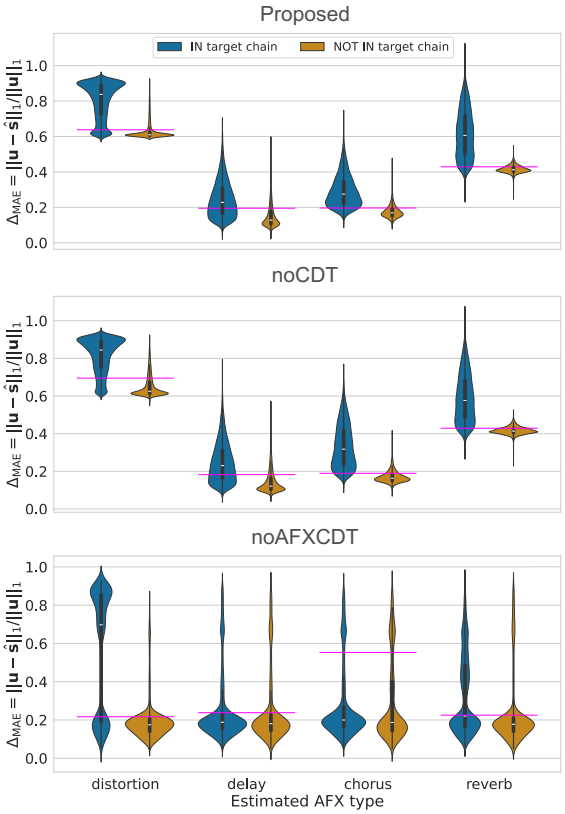


Figure 5: *The distributions of the MAE difference Δ_{MAE} between the input $\boldsymbol{u}$ and the output $\hat{\boldsymbol{s}}$ of SunAFXiNet. Each magenta line represents the top 20 percentile of the "NOT IN target chain" distribution for each AFX.*

4.5. Wet Signal Reproduction

We evaluated whether the estimated AFX chain $F_{\hat{\Lambda}}$ could reproduce the original input wet signal $\boldsymbol{y}$. We used the following evaluation metrics to assess the reproduction performance:

- SI-SDRi $(\mathrm{SI\text{-}SDR}(\boldsymbol{y}, F_{\hat{\Lambda}}(\cdot)) - \mathrm{SI\text{-}SDR}(\boldsymbol{y}, \cdot))$,
- MR-STFTi $(\mathcal{L}_{\mathrm{mrstft}}(\boldsymbol{y}, \cdot) - \mathcal{L}_{\mathrm{mrstft}}(\boldsymbol{y}, F_{\hat{\Lambda}}(\cdot)))$.

We evaluated the wet signal reproduced from the recovered dry signal $\hat{\boldsymbol{x}}$ as well as the wet signal reproduced from the ground-truth target dry signal $\boldsymbol{x}$. Note that if there was no AFX estimated to apply to the input wet signal $\boldsymbol{y}$ in the proposed method ($F_{\hat{\Lambda}} = \mathrm{id}$, the identity mapping, $\hat{n} = 0$), the SI-SDRi and MR-STFTi were both set to 0.

Table 4 shows the results of the wet signal reproduction. Given $\hat{\boldsymbol{x}}$, the Proposed model achieved a positive SI-SDRi value, suggesting successful reproduction of the input wet signal $\boldsymbol{y}$. Conversely, when using the noCDT and noAFXCDT models, the SI-SDRi was negative, indicating failure to reproduce $\boldsymbol{y}$. When $\boldsymbol{x}$ was used, all the three models achieved positive SI-SDRi values that were higher than the values for $\hat{\boldsymbol{x}}$.

However, in our further investigation, we found that the average SI-SDR value after application of the estimated AFX-chain, $\mathrm{SI\text{-}SDR}(\boldsymbol{y}, F_{\hat{\Lambda}}(\boldsymbol{x}))$ (3.49 dB for the Proposed model), was smaller than the SI-SDR value of the recovered dry signal, $\mathrm{SI\text{-}SDR}(\boldsymbol{y}, \hat{\boldsymbol{x}})$

Table 4: *Results of wet signal reproduction using estimated AFX chain* $F_{\hat{\Lambda}}$.

Model	SI-SDRi(↑)	MR-STFTi(↑)
Reproduced from recovered signal $\hat{\boldsymbol{x}}$		
Proposed	**0.67 dB**	**0.58**
noCDT	−0.59 dB	0.52
noAFXCDT	−0.81 dB	0.27
Reproduced from ground-truth signal $\boldsymbol{x}$		
Proposed	**2.24 dB**	**0.53**
noCDT	1.89 dB	0.52
noAFXCDT	1.41 dB	0.30

(5.25 dB for the Proposed model). For the purpose of re-designing musical signals using $F_{\hat{\Lambda}}$, considering $\hat{\boldsymbol{x}}$ as the dry signal would provide a meaningful starting point closer to the desired wet signal $\boldsymbol{y}$.

4.6. Discussion

The experimental results above confirm that the proposed method with the Proposed model enables the estimation of AFX chains, recovery of dry signals, and reproduction of wet signals. This contrasts with the outcomes obtained when using the noCDT and noAFXCDT models. Our ablation study indicates two findings. First, the proposed structure AFXCDT conditioned by the AFX type in SunAFXiNet is necessary to realize the valid dry signal recovery. Second, it is crucial to process the latent representations using the Transformer encoder's complexity both before and after the AFX config estimator.

Limitations. While previous research has not tackled the AFX-chain estimation and dry signal recovery jointly, it is worth comparing the performance of each component (bypassed signal estimator and AFX config estimator) to previously proposed methods mentioned in Section 2. Furthermore, researchers can integrate promising ideas presented in the related work, such as the compositional approach [22], graph representation [16], and the autoencoder approach [13], into our proposed approach that features the iterative application of SunAFXiNet as a key concept. These directions can be explored in future work.

5. APPLICATION TO SOUND DESIGN

With the successful recovery of dry signals and reproduction of wet signals, sound engineers can utilized the outputs of our proposed method for the new-style sound design referring to existing musical signals. Here we present two scenarios of sound design using our proposed method:

- An engineer has a wet guitar recording but does not have its dry signal, and applies our method to its wet signal. The method extracts the AFX chain (chorus, distortion) used in it and the corresponding dry guitar sound. Then the engineer boosts the *drive* parameter of the distortion to obtain a harsher distorted-guitar phrase, as illustrated in Figure 6.
- An engineer finds the AFX chain used in a power-chord guitar recording interesting, and wants to reuse it. The engineer then leverages our method to identify the chain (distortion, chorus, reverb) with the appropriate parameters, and applies it to another guitar lead recording in a melodic style, obtaining a distorted guitar-lead with similar sonic characteristics.

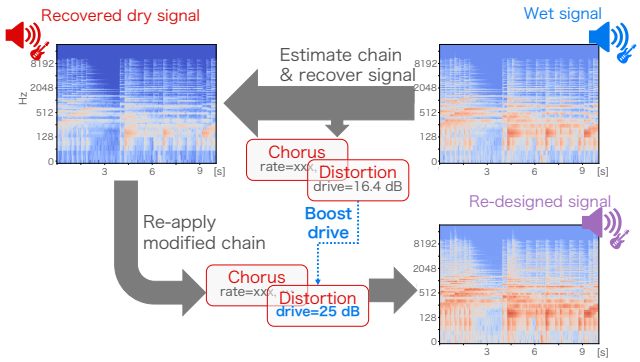


Figure 6: *An example of the new-style sound design enabled by our proposed method. Engineers can extract the AFX chain and the unprocessed dry signal from an existing wet signal, manipulate the estimated AFX chain, and then re-apply the modified AFX chain to the recovered dry signal. This process results in a musical signal with altered sonic characteristics.*

6. CONCLUSIONS

We proposed a method that analyzes the wet signal without any additional information and can estimate the AFX chain used in it and recover the dry signal. The contributions of this paper can be summarized as follows. First, we introduced a novel task "audio effect (AFX) chain estimation and dry signal recovery", which was not tackled in the literature. Second, we proposed a unique approach of repeatedly applying an identical DNN model to estimate the last-applied AFX and recover the signal before applying its AFX until the stop criterion is satisfied. It can thus reconstruct the entire AFX chain backwards and obtain the intermediate bypassed signal for each AFX in the chain. Third, we confirmed the effectiveness of our proposed method by using guitar phrase recordings with four types of AFXs. We further conducted the ablation study to reveal that the use of the proposed AFXCDT and several Transformer encoders are important.

Future work will include the extension of our method to handle more AFX types and different instruments. We also plan to build and validate a sound design support tool based on our method.

7. ACKNOWLEDGMENTS

This work is supported by JSPS KAKENHI 21H04900, 22H03639, and 23H03418, JST FOREST JPMJFR226V, and Moonshot R&D Grant Number JPMJPS2011.

8. REFERENCES

[1] Thomas Wilmering, David Moffat, Alessia Milo, and Mark B. Sandler, "A history of audio effects," *Applied Sciences*, vol. 10, no. 3, 791, 2020.

[2] Udo Zölzer, *DAFX-Digital Audio Effects (Second Edition)*, John Wiley & Sons, 2011.

[3] Brecht De Man, "Audio effects in sound design," in *Foundations in Sound Design for Linear Media*, pp. 113–128. Routledge, 2019.

[4] Naotake Masuda and Daisuke Saito, "Improving semi-supervised differentiable synthesizer sound matching for practical applications," *IEEE/ACM Trans. Audio Speech Lang. Process.*, vol. 31, pp. 863–875, 2023.

[5] Yang Yuting, Jin Zeyu, Barnes Connelly, and Finkelstein Adam, "White box search over audio synthesizer parameters," in *Proc. 24th Intl. Soc. Music Information Retrieval Conf. (ISMIR)*, 2023, pp. 190–196.

[6] Spyridon Stasis, Nicholas Jillings, Sean Enderby, and Stables Ryan, "Audio processing chain recommendation," in *Proc. 20th Intl. Conf. Digital Audio Effects (DAFx-17)*, 2017, pp. 103–109.

[7] Alec Wright, Eero-Pekka Damskägg, Lauri Juvela, and Vesa Välimäki, "Real-time guitar amplifier emulation with deep learning," *Applied Sciences*, vol. 10, no. 3, 766, 2020.

[8] Marco A. Martìnez Ramírez, Emmanouil Benetos, and Joshua D. Reiss, "Deep learning for black-box modeling of audio effects," *Applied Sciences*, vol. 10, no. 2, 638, 2020.

[9] Reemt Hinrichs, Kevin Gerkens, Alexander Lange, and Jörn Ostermann, "Blind extraction of guitar effects through blind system inversion and neural guitar effect modeling," *EURASIP J. Audio Speech Music Process.*, vol. 2024, no. 1, 9, 2024.

[10] Christian J. Steinmetz, Jordi Pons, Santiago Pascual, and Joan Serrà, "Automatic multitrack mixing with a differentiable mixing console of neural audio effects," in *Proc. IEEE Intl. Conf. Acoust. Speech Signal Process. (ICASSP)*, 2021, pp. 71–75.

[11] Christian J. Steinmetz, Nicholas J. Bryan, and Joshua D. Reiss, "Style transfer of audio effects with differentiable signal processing," *J. Audio Eng. Soc.*, vol. 70, no. 9, pp. 708–721, 2022.

[12] Marco Comunità, Dan Stowell, and Joshua D. Reiss, "Guitar effects recognition and parameter estimation with convolutional neural networks," *J. Audio Eng. Soc.*, vol. 69, no. 7/8, pp. 594–604, 2021.

[13] Côme Peladeau and Geoffroy Peeters, "Blind estimation of audio effects using an auto-encoder approach and differentiable digital signal processing," in *Proc. IEEE Intl. Conf. Acoust. Speech Signal Process. (ICASSP)*, 2024, pp. 1–5.

[14] Michael Stein, "Automatic detection of multiple, cascaded audio effects in guitar recordings," in *Proc. 13th Intl. Conf. Digital Audio Effects (DAFx-10)*, 2010, pp. 4–7.

[15] Jinyue Guo and Brian McFee, "Automatic recognition of cascaded guitar effects," in *Proc. 26th Intl. Conf. Digital Audio Effects (DAFx23)*, 2023, pp. 189–195.

[16] Sungho Lee, Jaehyun Park, Seungryeol Paik, and Kyogu Lee, "Blind estimation of audio processing graph," in *Proc. IEEE Intl. Conf. Acoust. Speech Signal Process. (ICASSP)*, 2023, pp. 1–5.

[17] Akira Maezawa, Katsutoshi Itoyama, Kazuyoshi Yoshii, and Hiroshi G. Okuno, "Nonparametric bayesian dereverberation of power spectrograms based on infinite-order autoregressive processes," *IEEE/ACM Trans. Audio Speech Lang. Process.*, vol. 22, no. 12, pp. 1918–1930, 2014.

[18] Stanislaw Gorlow and Joshua D. Reiss, "Model-based inversion of dynamic range compression," *IEEE Trans. Audio Speech Lang. Process.*, vol. 21, no. 7, pp. 1434–1444, 2013.

[19] Johannes Imort, Giorgio Fabbro, Marco A. Martínez Ramírez, Stefan Uhlich, Yuichiro Koyama, and Yuki Mitsufuji, "Distortion audio effects: Learning how to recover the clean signal," in *Proc. 23rd Intl. Soc. Music Information Retrieval Conf. (ISMIR)*, 2022, pp. 218–225.

[20] Chang-Bin Jeon and Kyogu Lee, "Music de-limiter networks via sample-wise gain inversion," in *Proc. IEEE Workshop Applications of Signal Process. to Audio Acoust. (WASPAA)*, 2023, pp. 1–5.

[21] Koichi Saito, Naoki Murata, Toshimitsu Uesaka, Chieh-Hsin Lai, Yuhta Takida, Takao Fukui, and Yuki Mitsufuji, "Unsupervised vocal dereverberation with diffusion-based generative models," in *Proc. IEEE Intl. Conf. Acoust. Speech Signal Process. (ICASSP)*, 2023, pp. 1–5.

[22] Matthew Rice, Christian J. Steinmetz, George Fazekas, and Joshua D. Reiss, "General purpose audio effect removal," in *Proc. IEEE Workshop Applications of Signal Process. to Audio Acoust. (WASPAA)*, 2023, pp. 1–5.

[23] Simon Rouard, Francisco Massa, and Alexandre Défossez, "Hybrid transformers for music source separation," in *Proc. IEEE Intl. Conf. Acoust. Speech Signal Process. (ICASSP)*, 2023, pp. 1–5.

[24] Alexandre Défossez, Nicolas Usunier, Léon Bottou, and Francis R. Bach, "Music source separation in the waveform domain," *arXiv preprint arXiv:1911.13254*, 2019.

[25] Alexandre Défossez, "Hybrid spectrogram and waveform source separation," in *Proc. Music Demixing Workshop 2021 (MDX 2021)*, 2021, pp. 1–11.

[26] Qingyang Xi, Rachel M. Bittner, Johan Pauwels, Xuzhou Ye, and Juan P. Bello, "GuitarSet: A dataset for guitar transcription," in *Proc. 19th Intl. Soc. Music Information Retrieval Conf. (ISMIR)*, 2018, pp. 453–460.

[27] Christian Kehling, Jakob Abeßer, Christian Dittmar, and Gerald Schuller, "Automatic tablature transcription of electric guitar recordings by estimation of score- and instrument-related parameters," in *Proc. 17th Intl. Conf. Digital Audio Effects (DAFx-14)*, 2014, pp. 219–226.

[28] Ethan Manilow, Gordon Wichern, Prem Seetharaman, and Jonathan Le Roux, "Cutting music source separation some Slakh: A dataset to study the impact of training data quality and quantity," in *Proc. IEEE Workshop Applications of Signal Process. to Audio Acoust. (WASPAA)*, 2019, pp. 1–5.

[29] Peter Sobot, "Pedalboard," Zenodo, https://doi.org/10.5281/zenodo.7817838, 2021.

[30] Xin Wang, Shinji Takaki, and Junichi Yamagishi, "Neural source-filter waveform models for statistical parametric speech synthesis," *IEEE/ACM Trans. Audio Speech Lang. Process.*, vol. 28, pp. 402–415, 2020.

[31] Christian J. Steinmetz and Joshua D. Reiss, "auraloss: Audio focused loss functions in PyTorch," in *Proc. Digital Music Research Network One-day Workshop (DMRN+15)*, 2020.

[32] Jonathan Le Roux, Scott Wisdom, Hakan Erdogan, and John R. Hershey, "SDR – half-baked or well done?," in *Proc. IEEE Intl. Conf. Acoust. Speech Signal Process. (ICASSP)*, 2019, pp. 626–630.