GENERAL PURPOSE AUDIO EFFECT REMOVAL

Author 1,[1] *Author 2,*[2] *Author 3,*[3]

[1] Author names and affiliations omitted for double-blind review.
[2] Please do not make changes to this section.
[3] Author information may be added after paper acceptance.

ABSTRACT

Although the design and application of audio effects is well understood, the inverse problem of removing these effects is significantly more challenging and far less studied. Recently, deep learning has been applied to audio effect removal, however existing approaches have focused on narrow formulations considering only one effect or source type at a time. In realistic scenarios, multiple effects are applied with varying source content. This motivates a more general task, which we refer to as *general purpose audio effect removal*. We developed a dataset for this task using five audio effects across four different sources and used it to train and evaluate a set of existing architectures. We found that no single model performed optimally on all effect types and sources. To address this, we introduced RemFX, an approach designed to mirror the compositionality of applied effects. We first trained a set of the best-performing effect-specific removal models and then leveraged an audio effect classification model to dynamically construct a graph of our models at inference. We found our approach outperforms single model baselines, however, examples with many effects present remain challenging.

Index Terms— audio effects, deep learning, audio engineering

1. INTRODUCTION

Audio effects are signal processing devices used to shape sonic characteristics and they play a central role in audio production with applications in music, film, broadcast, and video games [1]. While there is a mature body of work for the design and implementation of audio effects [2], the inverse problem of audio effect removal is more challenging. With the rise of music source separation, interest in remixing, manipulating, and re-purposing recorded audio content has continued to grow [3, 4]. Audio effect removal unlocks further control over remixing content and also facilitates more powerful audio effect style transfer applications [5, 6]. In addition, audio effect removal also has applications for data generation, which could improve source separation and automatic mixing systems [7, 8], and could also be useful in educational contexts, enabling students to better understand the techniques of professional audio engineers.

Previous systems for audio effect removal rely on traditional signal processing methods that target specific effects such as distortion [9, 10], compression [11], and reverberation [12]. However, these approaches require specialized techniques for each effect and make strong assumptions about the effect implementation, limiting their generality. More recently, deep learning has been applied to this task, enabling a more general and powerful data-driven approach. Nonetheless, existing systems are still narrow in their scope, addressing only a small number of effects such as distortion [13, 14] or reverberation [15, 16]. Although some work on speech enhancement has considered the removal of audio effects, which can be seen as corruptions of speech [17, 18], these approaches are limited in that they consider only speech and operate at relatively low sample rates ($\leq$ 16 kHz). This limits their applicability in post-production where high-fidelity and support for a wide range of content is required. In addition, previous work has focused on removing only one effect at a time, whereas real-world audio often has multiple effects present simultaneously [19]. It is common to chain together multiple audio effects to achieve a specific result, which significantly complicates the task of removing these effects.

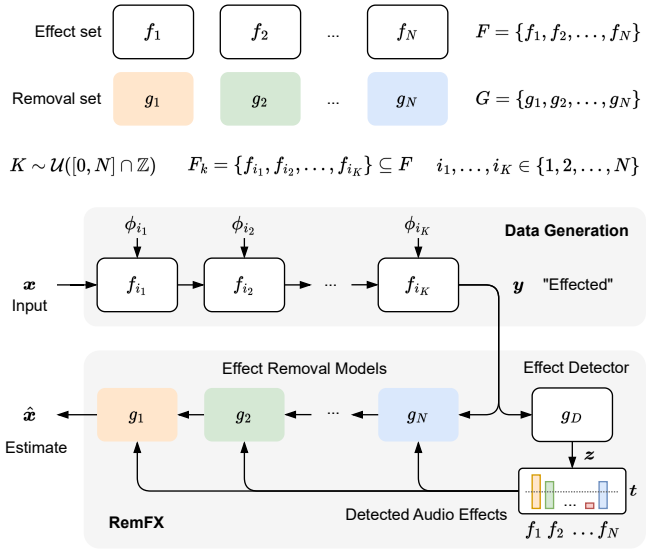

Figure 1: We introduce the task of *general purpose audio effect removal*, which considers removing multiple audio effects from the same recording and we propose RemFX, a compositional approach that dynamically combines effect-specific removal models.

We have three main contributions. Firstly, we address the shortcomings of previous research by introducing a more comprehensive task we name *general purpose audio effect removal*. Secondly, we conduct a series of experiments with our benchmark datasets on single and multiple effect removal. We discover that some architectures are more effective at removing certain effects and that certain effects are more challenging than others. We also find that when using single models for multiple effect scenarios, performance is degraded. Thirdly, to overcome this, we introduce RemFX, which we demonstrate surpasses baselines by dynamically composing pretrained effect-specific models at inference. Despite improved performance, our results suggest more work is necessary in cases with many effects applied at the same time. We provide listening examples, datasets, code, and pretrained models to aid further research.[1]

[1] https://anon2881.github.io/RemFX

2. AUDIO EFFECT REMOVAL

Audio effects are signal processing devices used to manipulate attributes of a sound recording and can be represented by a function

$$\boldsymbol{y} = f(\boldsymbol{x}; \phi), \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^T$ denotes the monophonic audio input with T samples, while $\phi \in \mathbb{R}^P$ represents the device operation with P parameters. The function f takes $\boldsymbol{x}$ as input and produces an output signal $\boldsymbol{y} \in \mathbb{R}^T$, which we refer to as the "effected" signal.

In the basic audio effect removal task, the objective is to construct a function g that estimates the signal $\hat{\boldsymbol{x}}$ given the effected $\boldsymbol{y}$

$$\hat{\boldsymbol{x}} = g(\boldsymbol{y}). \tag{2}$$

While knowledge of the device parameters ϕ would be helpful, they are generally unknown and therefore not considered. It should be noted that achieving a perfect reconstruction of the original signal is generally not possible due to the uncertainty surrounding the ground truth given only the effected signal without access to the original device or its parameters. Therefore, our aim is to reduce the perceptual difference between the signals such that when a listener hears the output $\hat{\boldsymbol{x}}$, they perceive it as close to the original signal $\boldsymbol{x}$.

In audio production there is a range of effects including time-based effects such as reverberation and delay, dynamic processing such as distortion and compression, spectral effects such as equalization, and modulation effects such as chorus, tremolo, flanger, and phaser [2]. To address this, we represent a set of N audio effects as a set of functions $F = \{f_1, f_2, ..., f_N\}$ and aim to construct a removal function g

$$\hat{\boldsymbol{x}} = g(f_i(\boldsymbol{x}; \phi_i)), \tag{3}$$

that can recover an estimate of the original signal $\hat{\boldsymbol{x}}$ after the application any effect f_i for $i \in 1, 2, ..., N$. However, this task is further complicated by the fact that multiple effects can be applied to the same recording in order to achieve a desired sound [19]. This can be represented by a composition of multiple effect functions, each with its own control parameters. It is important to note that the order of effects can vary and that each effect may or may not be present in any given example, which further complicates the task.

Motivated by this, we formulate the task of *general purpose audio effect removal*. We begin by defining a set of N functions $F = \{f_1, f_2, ..., f_N\}$ that represent a group of common audio effects. We also define a dataset $\mathcal{D} = \{\boldsymbol{x}^{(j)}\}_{j=1}^{J}$ containing J clean audio recordings where no effects have been applied. To generate effected recordings we randomly sample the number of effects to apply as $K \sim \mathcal{U}([0, N] \cap \mathbb{Z})$. Then, we sample K functions without replacement from F to produce a subset $F_K = \{f_{i_1}, f_{i_2}, \ldots, f_{i_K}\} \subseteq F$, where $i_1, i_2, ..., i_K$ represent the indices of elements in F. We then compose the effects in F_K following the order in which they were drawn and sample continuous parameters $\phi_{i_k} \sim \mathcal{U}(a_{i_k}, b_{i_k})$ for the k-th effect over a predefined range $[a_{i_k}, b_{i_k}]$. We represent the composition of these K functions with random parameters as

$$\boldsymbol{y}^{(j)} = f_K(f_{K-1}(\ldots f_2(f_1(\boldsymbol{x}^{(j)}; \phi_{i_1}); \phi_{i_2}) \ldots; \phi_{i_{K-1}}); \phi_{i_K}), \tag{4}$$

where $\boldsymbol{y}^{(j)}$ is the resulting output of processing $\boldsymbol{x}^{(j)}$, the j-th example from the dataset. Our goal is then to construct a general purpose function g such that, given a signal processed by a randomly sampled composition of effect functions, it will produce an estimate of the recording $\hat{\boldsymbol{x}}^{(j)}$ without the presence of effects. While in some cases effects may be applied in parallel or using more complex routing, our formulation that considers sequential effects captures much of the complexity in real-world audio effect removal.

3. APPROACH

As introduced in Sec. 2, the *general purpose audio effect removal* task involves removing any number of audio effects applied to a recording from a set of possible effects. One straightforward approach to address this task could involve using a single neural network model to remove all effects at once with the model trained to regress the original signal. We refer to these approaches as monolithic networks, since they use a singular model to remove a range of effects. However, due to the combinatorial and compositional nature of the audio effect removal task, we hypothesize that using a monolithic network will not produce adequate results. Given N different effects and assuming that each effect is applied at most once, the total number of possible effect configurations is given by $\sum_{k=0}^{N} P(N, k) = \sum_{k=0}^{N} \frac{N!}{(N-k)!}$, the sum of all permutations across each number of chosen effects $k \in 1, 2, ..., N$. Beyond the combinatorial nature of the problem, it is also likely that there will be a significant variance in the difficulty of training examples, since some examples will contain N effects while others may contain none. This may lead the network to focus more on difficult examples that contribute to higher training error. This may disrupt training, potentially harming performance on simpler cases.

3.1. Compositional audio effect removal

To address the limitations of monolithic networks in this task, we propose a compositional approach, which we name RemFX. As shown in Fig. 1, our approach is designed to mirror the process of applying a series of audio effects. We achieve this by first constructing a set of N audio effect-specific removal models $G = \{g_1, g_2, \ldots, g_N\}$. We choose the best-performing model architecture for each effect removal model based on our initial experiments. We train each of these networks with a separate dataset to remove a different effect from the effect set $F = \{f_1, f_2, \ldots, f_N\}$.

After constructing our set of removal models G, we then introduce an audio effect detection network $\boldsymbol{z} = g_D(\boldsymbol{y})$ where $\boldsymbol{z} \in \mathbb{R}^N$. This network is trained in a separate task to detect the presence of any effect from F in the effected recording $\boldsymbol{y}$, which we frame as a multi-label classification task. At inference, we apply a threshold t to the logits $\boldsymbol{z} = (z_1, z_2, ..., z_N)$ from g_D, selecting all effects where $\boldsymbol{z} \geq t$. This enables us to construct a series connection of our effect-specific removal models from G and then apply this composite function to remove any effects, dynamically adapting computation at inference. We do not estimate the order since we found that random ordering performs similarly to the true ordering (Sec. 5.5).

During inference, our effect-specific models will encounter effects in the input signal that they are not trained to remove. To improve robustness in these scenarios, we propose an approach called FXAug. When training an effect-specific removal model g_n to remove an effect f_n from the effect set F, we apply additional distractor effects from the set $F \setminus \{f_n\}$. In general, we sample $K_d \sim \mathcal{U}([0, N-1] \cap \mathbb{Z})$ distractor effects and apply them with randomly sampled parameters before applying the effect to be removed f_n. We then use the intermediate signal containing the distractor effects as the target signal during training, instead of the clean signal.

Compared to monolithic approaches, our approach offers several benefits: it allows for adaptive computation during inference, running only the removal networks of effects that are present, enables expansion to more effects without requiring complete retraining of existing removal models, and facilitates using different architectures specialized for the removal of each effect type.

Approach	Params	DST		DRC		RVB		CHS		DLY		AVG	
		SI-SDR	STFT	SI-SDR	STFT	SI-SDR	STFT	SI-SDR	STFT	SI-SDR	STFT	SI-SDR	STFT
Input	-	16.37	0.654	15.57	0.779	9.30	0.866	8.31	0.539	11.28	0.742	12.17	0.716
DPTNet	2.9 M	22.38	0.798	16.95	0.810	9.817	1.128	8.50	0.870	11.768	0.957	13.89	0.913
UMX	6.3 M	17.38	0.505	15.39	0.534	11.39	0.706	8.88	0.534	12.87	0.688	13.18	0.593
DCUNet	7.7 M	16.27	0.528	13.80	0.591	12.13	**0.645**	**11.08**	**0.504**	13.48	**0.616**	13.35	0.577
TCN	10.0 M	18.47	0.632	14.49	0.733	13.25	0.804	8.452	0.669	11.23	0.882	13.18	0.744
HDemucs	83.6 M	**24.36**	**0.402**	**20.08**	**0.422**	**13.59**	0.735	9.828	0.580	**13.54**	0.671	**16.30**	**0.562**

Table 1: Audio effect specific models trained to remove one effect across five audio processing network architectures.

Approach	Train	DST	DRC	RVB	CHS	DLY	AVG
Input	-	0.586	0.775	0.813	0.511	0.713	0.680
HDemucs	Single	**0.354**	**0.398**	0.689	0.572	0.644	**0.531**
	Multiple	0.487	0.558	0.782	0.632	0.761	0.644
DCUNet	Single	0.490	0.573	**0.601**	**0.484**	**0.586**	0.547
	Multiple	0.562	0.666	0.748	0.578	0.687	0.648

Table 2: Multi-resolution STFT error for single effect models compared to the same architecture trained to remove all five effects.

4. EXPERIMENTAL SETUP

Dataset — We source audio from four datasets: VocalSet [20] for singing voice, GuitarSet [21] for acoustic guitar, DSD100 [22] for bass guitar, and IDMT-SMT-Drums [23] for drums. For each experimental configuration, we split each set into train, validation, and test, ensuring there is no overlap between song, performer, or instruments, where applicable. We resample to $f_s = 48$ kHz and split audio into ∼5.5 sec chunks (262144 samples). We fix the number of train, validation, and test examples to 8 k, 1 k, and 1 k. We generate effected audio by applying randomly sampled effects and parameters using `Pedalboard` [24]. The ranges of these parameters are selected heuristically to model real-world use cases. After each effect, we loudness normalize the audio with a target of -20 dB LUFS [25]. We consider five effects: Distortion (DST), Dynamic range compression (DRC), Reverberation (RVB), Chorus (CHS), and Feedback delay (DLY). This results in 12.1 h for training, 1.5 h for validation, and 1.5 h for testing per effect experiment.

Removal models — We consider five audio processing model architectures in our experiments. These include, Hybrid Demucs [26], DCUNet [27], DPTNet [28], TCN [29, 30], and UMX [31].

Detection models — Similar to past work in effect classification [32, 33, 34], we consider convolutional architectures operating on Mel spectrograms. As baselines, we train simple linear layers on top of a set of frozen pretrained audio representations including PANNs [35], wav2vec2.0 [36], and Wav2CLIP [37]. For comparison, we also train PANNs from scratch at $f_s = 48$ kHz.

Training details — All models are trained with Adam. Removal models are trained for 50 k steps with an initial learning rate of 10^{-4} and weight decay of 10^{-3} using a batch size optimal for a single A100 GPU. Audio effect classifiers are trained with a learning rate of $3 \cdot 10^{-4}$ for 300 epochs using a batch size of 64. We use learning rate scheduling, decreasing by a factor of 10 at 80% and 95% through training, and gradient clipping with a value of 10. While audio effect classifiers are trained with binary cross-entropy, removal models are trained with a sum of two terms $\mathcal{L} = \alpha\mathcal{L}_{\text{L1}} + \beta\mathcal{L}_{\text{MR-STFT}}$, with $\alpha = 100$ and $\beta = 1$, where $\mathcal{L}_{\text{L1}}$ is the L1 distance in the time domain and $\mathcal{L}_{\text{MR-STFT}}$ is the multi-resolution STFT loss [38, 39].

Approach	DST	DRC	RVB	CHS	DLY	AVG
wav2vec2* [36]	0.720	0.710	0.776	0.651	0.662	0.704
Wav2CLIP* [37]	0.642	0.667	0.850	0.697	0.699	0.720
PANNs* [35]	0.681	0.681	0.841	0.705	0.730	0.732
PANNs	**0.780**	0.771	0.791	0.724	0.680	0.750
+ SpecAug	**0.780**	**0.807**	**0.845**	**0.751**	**0.743**	**0.786**

Table 3: Class-wise accuracy for the audio effect detection task.

5. EXPERIMENTS & RESULTS

5.1. Effect specific models

We report the SI-SDR [40] to investigate performance in the time domain and the multi-resolution STFT error to capture performance in the frequency domain. We train one model for each of the architectures across five different effects, resulting in a total of 25 models for the task in (2). As shown in Table 1, we found that no single architecture performs optimally across all effect removal tasks. Hybrid Demucs outperforms other models in SI-SDR and STFT for distortion and compression, whereas DCUNet performs better on chorus. Although the performance of the models is similar for reverberation and delay, STFT metrics suggest that DCUNet performs better while SI-SDR scores are close. These results align with our informal listening, however it also reveals that effects like chorus and delay remain challenging even for the best-performing models.

5.2. Monolithic removal models

As a first step towards the *general purpose audio effect removal* task, we train Hybrid Demucs and DCUNet as monolithic models to remove multiple effects when only one effect is present at a time, as in (3). We report the results in Table 2, comparing the performance to the effect-specific models from the previous experiment. When training to remove multiple types of effects, we observe that both architectures perform worse as compared to when they are trained to remove only a single effect. This provides evidence for our claim in Sec. 3 that monolithic models may not produce adequate results.

5.3. Audio effect detection

We frame the audio effect detection task as N separate binary classification tasks, where one linear layer followed by a sigmoid generates a prediction for the presence of each effect. We report the class-wise accuracy on held-out data in Table 3. We found superior performance training PANNs from scratch as compared to adapting the pretrained models, and we observed a small benefit (+3.6% accuracy) from SpecAugment. However, our results indicate the multiple effect detection task could be further improved, as the best-performing model achieves 78.6% accuracy across all effects.

Approach	Params.	$N=0$		$N=1$		$N=2$		$N=3$		$N=4$		$N=5$	
		SI-SDR	STFT	SI-SDR	STFT	SI-SDR	STFT	SI-SDR	STFT	SI-SDR	STFT	SI-SDR	STFT
Input	-	Inf	0.000	11.52	0.689	6.24	1.131	3.29	1.508	1.31	1.799	-0.33	2.058
DCUNet	7.7 M	18.53	0.467	11.16	0.743	7.87	0.945	5.42	1.121	3.64	1.265	2.10	1.462
HDemucs-M	84 M	19.72	0.415	11.28	0.728	8.01	0.931	5.77	1.100	4.29	1.223	2.10	1.337
HDemucs-L	334 M	20.78	0.410	11.53	0.725	8.17	0.924	6.08	1.084	4.63	1.212	3.38	1.328
HDemucs-XL	751 M	20.31	0.406	11.54	0.713	8.32	0.902	6.19	1.064	4.73	1.190	3.38	**1.312**
RemFX Oracle	$\leq$192 M	**Inf**	**0.000**	**16.99**	**0.486**	**10.91**	**0.762**	**7.51**	**0.994**	**5.40**	**1.170**	**3.47**	1.360
RemFX All	192 M	21.99	0.234	10.26	0.841	8.44	0.939	6.46	1.084	4.71	1.225	2.99	1.418
RemFX Detect	$\leq$192 M	87.54	0.068	16.67	0.495	10.47	0.786	6.96	1.050	4.80	1.247	2.61	1.486

Table 4: SI-SDR (↑) and STFT (↓) performance in *general purpose audio effect removal* across fixed number of audio effects N.

Effect	Approach	Single Effect		w/ Distractors	
		SI-SDR	STFT	SI-SDR	STFT
AVG	Input	12.17	0.716	10.60	0.692
	Single Effect	16.28	0.612	14.71	0.580
	+ FXAug	**16.59**	**0.554**	**16.61**	**0.514**

Table 5: Average SI-SDR and MR-STFT error across all effects for single effect removal trained with and without FXAug.

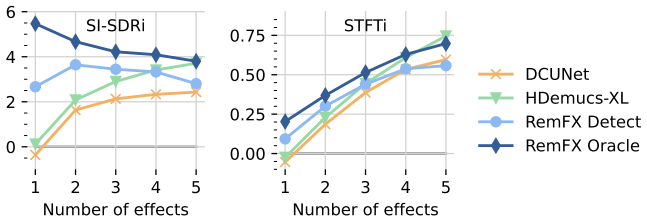

Figure 2: SI-SDR and STFT improvement across a range of effects.

5.4. Audio effect augmentation

We hypothesized that training effect-specific models with only one effect during training would lead to degraded performance at inference. To investigate this and the efficacy of our FXAug approach, we trained a Hybrid Demucs model for each effect, with and without FXAug. We evaluated these models on two test sets: one with only one effect and one with up to four random distractor effects. We report the mean performance across all five effect-specific models in Table 5. We make two conclusions. First, we confirm that distractor effects harm performance for models trained with only one effect (no FXAug). Second, we find that this is remedied by using FXAug, which improves performance in the case of distractors, but also in the case of a single effect. Therefore, we use effect-specific models trained with FXAug in our final RemFX system.

5.5. General purpose audio effect removal

In our final experiment, we investigated the performance of systems on the *general purpose audio effect removal* task using our set of audio effects, applying up to five at a time. We trained monolithic Hybrid Demucs and DCUNet models and compared against variants of RemFX with results in Table 4. These include All: apply all effect-specific models, Oracle: apply respective models given ground truth labels of the effects that are present, and Detect: use the audio effect classifier to determine these labels. For the effect-specific models, we used Hybrid Demucs for distortion and compression, and DCUNet for reverberation, delay, and chorus, along with the best-performing classifier from Table 3, using a threshold of $t = 0.5$. The ordering of the models was randomized for each example, except for Oracle, which used the ground truth ordering.

Number of effects — The case of no effects, $N = 0$, exhibits one of the benefits of RemFX, which will not process the input unless audio effects are detected. On the other hand, the monolithic models produced noticeable degradation across both metrics. For $N = 1$, we found that monolithic models struggle, performing worse even than the input across both metrics, while RemFX Oracle achieved a significant improvement. Even RemFX Detect only had a small performance dip ($< 2\%$) and still outperformed the monolithic models. This trend is similar for $N = 2$. While the monolithic models provided a small improvement compared to the input, RemFX models were superior. As the number of effects increases, the gap between RemFX and the monolithic models decreases, which is shown by Fig. 2. While RemFX outperformed the baselines when fewer effects are present, all approaches exhibited degraded performance for $N = 4$ and $N = 5$, indicating the difficulty of this task.

Model scale — To improve performance of monolithic models, we attempted to further scale Hybrid Demucs. However, we observed minimal improvement in SI-SDR ($\leq$ 1.3 dB) and STFT ($\leq$ 0.04) across N, even when scaling to 751 M parameters. In comparison, RemFX models performed better for $N \leq 3$ and use fewer parameters, ranging from 0 to 192 M, depending on the detected effects.

Ordering & Detection — For RemFX Oracle, we compared using the ground truth and random ordering. We found a slight decrease in SI-SDR ($\leq$ 0.6 dB) and STFT ($\leq$ 0.06) across N when using a random ordering, leading us to conclude the use of random ordering in our RemFX Detect model is an acceptable approach. We also established the importance of the classifier, since RemFX All results in a max decrease of 6.41 dB SI-SDR and 0.346 STFT.

6. CONCLUSION

We introduced a new task, *general purpose audio effect removal*, and investigated several approaches to tackle it. Our findings suggested that monolithic networks fail to generalize across cases with varying number of effects, however our RemFX system yielded improved performance by combining an audio effect detection model with dynamic construction of effect-specific removal models. While the results are promising, our evaluation is still limited in that we considered only five effects, each with a single implementation, and we did not consider more complex signal routing, such as parallel connections. Despite these limitations, our proposed method offers promising results in effect removal and provides a direction for improved audio effect removal systems that are scalable and applicable in real-world scenarios. We open source our code, datasets, and provide pretrained models to facilitate future work.

7. REFERENCES

[1] T. Wilmering, D. Moffat, A. Milo, and M. B. Sandler, "A history of audio effects," *Applied Sciences*, vol. 10, no. 3, 2020.

[2] U. Zölzer *et al.*, *DAFX-Digital audio effects*. John Wiley & Sons, 2002.

[3] H. Yang, S. Firodiya, N. J. Bryan, and M. Kim, "Don't separate, learn to remix: End-to-end neural remixing with joint optimization," in *ICASSP*. IEEE, 2022.

[4] H. Yang *et al.*, "Upmixing via style transfer: a variational autoencoder for disentangling spatial images and musical content," in *ICASSP*. IEEE, 2022.

[5] C. J. Steinmetz, N. J. Bryan, and J. D. Reiss, "Style transfer of audio effects with differentiable signal processing," *Journal of the Audio Engineering Society*, 2022.

[6] J. Koo, M. A. Martinez-Ramirez, W.-H. Liao, S. Uhlich, K. Lee, and Y. Mitsufuji, "Music mixing style transfer: A contrastive learning approach to disentangle audio effects," *arXiv preprint arXiv:2211.02247*, 2022.

[7] C. J. Steinmetz, "Deep learning for automatic mixing: challenges and next steps," in *MDX Workshop at ISMIR*, 2021.

[8] M. A. Martínez-Ramírez, W.-H. Liao, G. Fabbro, S. Uhlich, C. Nagashima, and Y. Mitsufuji, "Automatic music mixing with deep learning and out-of-domain data," in *ISMIR*, 2022.

[9] P. Záviška *et al.*, "A survey and an extensive evaluation of popular audio declipping methods," *IEEE Journal of Selected Topics in Signal Processing*, vol. 15, no. 1, 2020.

[10] A. Bernardini, A. Sarti, *et al.*, "Towards inverse virtual analog modeling," in *DAFx 2019*, 2019.

[11] S. Gorlow and J. D. Reiss, "Model-based inversion of dynamic range compression," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 7, 2013.

[12] K. Lebart, J.-M. Boucher, and P. N. Denbigh, "A new method based on spectral subtraction for speech dereverberation," *Acta Acustica united with Acustica*, vol. 87, no. 3, 2001.

[13] J. Imort, G. Fabbro, M. A. Martínez-Ramírez, S. Uhlich, Y. Koyama, and Y. Mitsufuji, "Distortion audio effects: Learning how to recover the clean signal," in *ISMIR*, 2022.

[14] E. Moliner, J. Lehtinen, and V. Välimäki, "Solving audio inverse problems with a diffusion model," *arXiv preprint arXiv:2210.15228*, 2022.

[15] K. Saito, N. Murata, T. Uesaka, C.-H. Lai, Y. Takida, T. Fukui, and Y. Mitsufuji, "Unsupervised vocal dereverberation with diffusion-based generative models," in *ICASSP*, 2023.

[16] N. Murata, K. Saito, C.-H. Lai, Y. Takida, T. Uesaka, Y. Mitsufuji, and S. Ermon, "Gibbsddrm: A partially collapsed gibbs sampler for solving blind inverse problems with denoising diffusion restoration," *arXiv preprint arXiv:2301.12686*, 2023.

[17] J. Serrà, S. Pascual, J. Pons, R. O. Araz, and D. Scaini, "Universal speech enhancement with score-based diffusion," *arXiv preprint arXiv:2206.03065*, 2022.

[18] J. Su, Z. Jin, and A. Finkelstein, "HiFi-GAN: High-fidelity denoising and dereverberation based on speech deep features in adversarial networks," in *INTERSPEECH*, 2020.

[19] A. Case, *Sound FX: Unlocking the creative potential of recording studio effects*. CRC Press, 2012.

[20] J. Wilkins, P. Seetharaman, A. Wahl, and B. Pardo, "Vocalset: A singing voice dataset." in *ISMIR*, 2018.

[21] Q. Xi, R. M. Bittner, J. Pauwels, X. Ye, and J. P. Bello, "Guitarset: A dataset for guitar transcription." in *ISMIR*, 2018.

[22] A. Liutkus *et al.*, "The 2016 signal separation evaluation campaign," in *LVA/ICA*, 2017.

[23] C. Dittmar and D. Gärtner, "Real-time transcription and separation of drum recordings based on nmf decomposition." in *DAFx*, 2014.

[24] P. Sobot, "Pedalboard," Apr. 2023, 10.5281/zenodo.7817839.

[25] C. J. Steinmetz and J. D. Reiss, "pyloudnorm: A simple yet flexible loudness meter in python," in *150th Convention of the AES*, 2021.

[26] A. Défossez, "Hybrid spectrogram and waveform source separation," *arXiv preprint arXiv:2111.03600*, 2021.

[27] H.-S. Choi, J.-H. Kim, J. Huh, A. Kim, J.-W. Ha, and K. Lee, "Phase-aware speech enhancement with deep complex u-net," in *ICLR*, 2019.

[28] J. Chen, Q. Mao, and D. Liu, "Dual-path transformer network: Direct context-aware modeling for end-to-end monaural speech separation," in *INTERSPEECH*, 2020.

[29] D. Rethage, J. Pons, and X. Serra, "A WaveNet for speech denoising," in *ICASSP*. IEEE, 2018.

[30] C. J. Steinmetz and J. D. Reiss, "Efficient neural networks for real-time modeling of analog dynamic range compression," in *152nd Convention of the AES*, 2022.

[31] F.-R. Stöter, S. Uhlich, A. Liutkus, and Y. Mitsufuji, "Open-unmix-a reference implementation for music source separation," *Journal of Open Source Software*, vol. 4, no. 41, 2019.

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

[33] H. Jürgens, R. Hinrichs, and J. Ostermann, "Recognizing guitar effects and their parameter settings," in *DAFx*, 2020.

[34] M. Comunità, D. Stowell, and J. D. Reiss, "Guitar effects recognition and parameter estimation with convolutional neural networks," *Joural of the Audio Engineering Society*, 2020.

[35] Q. Kong, Y. Cao, T. Iqbal, Y. Wang, W. Wang, and M. D. Plumbley, "Panns: Large-scale pretrained audio neural networks for audio pattern recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 28, 2020.

[36] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," *NeurIPS*, vol. 33, 2020.

[37] H.-H. Wu, P. Seetharaman, K. Kumar, and J. P. Bello, "Wav2clip: Learning robust audio representations from clip," in *ICASSP*. IEEE, 2022.

[38] R. Yamamoto, E. Song, and J.-M. Kim, "Parallel WaveGAN: A fast waveform generation model based on generative adversarial networks with multi-resolution spectrogram," in *ICASSP*, 2020.

[39] C. J. Steinmetz and J. D. Reiss, "auraloss: Audio focused loss functions in PyTorch," in *DMRN+15*, 2020.

[40] J. Le Roux, S. Wisdom, H. Erdogan, and J. R. Hershey, "Sdr–half-baked or well done?" in *ICASSP*. IEEE, 2019.