OPEN ACCESS Freely available online

S. Lee, M. A. Martínez-Ramírez, W.-H. Liao, S. Uhlich, G. Fabbro, K. Lee, and Y. Mitsufuji, "Reverse Engineering of Music Mixing Graphs With Differentiable Processors and Iterative Pruning," *J. Audio Eng. Soc.*, vol. 73, no. 6, pp. 344–365 (2025 Jun.). https://doi.org/10.17743/jaes.2022.0212.

Reverse Engineering of Music Mixing Graphs With Differentiable Processors and Iterative Pruning

SUNGHO LEE,[1,*] MARCO A. MARTÍNEZ-RAMÍREZ,[2] WEI-HSIANG LIAO,[2] STEFAN UHLICH,[3]
(sh-lee@snu.ac.kr) (marco.martinez@sony.com) (weihsiang.liao@sony.com) (stefan.uhlich@sony.com)

GIORGIO FABBRO,[3] KYOGU LEE,[1] AND YUKI MITSUFUJI[2,4]
(giorgio.fabbro@sony.com) (kglee@snu.ac.kr) (yuhki.mitsufuji@sony.com)

[1]*Department of Information and Intelligence, Seoul National University, Seoul, South Korea*
[2]*Sony AI, Tokyo, Japan*
[3]*Sony Europe B.V., Stuttgart, Germany*
[4]*Sony Group Corporation, Tokyo, Japan*

Reverse engineering of music mixes aims to uncover how dry source signals are processed and combined to produce a final mix. In this paper, prior works are extended to reflect the compositional nature of mixing and search for a graph of audio processors. First, a mixing console is constructed, applying all available processors to every track and subgroup. With differentiable processor implementations, their parameters are optimized with gradient descent. Next, the process of removing negligible processors and fine-tuning the remaining ones is repeated. This way, the quality of the full mixing console can be preserved while removing approximately two-thirds of the processors. The proposed method can be used not only to analyze individual music mixes but also to collect large-scale graph data for downstream tasks such as automatic mixing. Especially for the latter purpose, efficient implementation of the search is crucial. To this end, an efficient batch-processing method that computes multiple processors in parallel is presented. Also, the "dry/wet" parameter of each processor is exploited to accelerate the search. Extensive quantitative and qualitative analyses are conducted to evaluate the proposed method's performance, behavior, and computational cost.

0 INTRODUCTION

Production of modern music recordings involves a mixing process, balancing source tracks in various aspects, such as loudness, frequency content, and spatialization. Mixing engineers achieve this by combining and controlling audio processors, such as equalizers and artificial reverberation. Numerous attempts have been made to uncover this complex process. One way is by directly collecting expert knowledge [1–3]. Alternatively, one can develop a computational method that estimates the mixing process from input source tracks and output mix. This strategy, so-called *reverse engineering*, has several advantages. First, it is useful on its own because it allows one to analyze and understand the individual mixes. Furthermore, large-scale mixing data can be collected from various public music data sets [4–6] that provide mixes along with their source tracks. This enables more scalable, data-driven downstream applications. For example, statistical analysis of the mixing practices can be conducted [7–9]. The data can also be used to develop automatic mixing systems [10–14] that could include neural network components.

Reverse engineering of music mixing was initially tackled by Barchiesi and Reiss [15]. By assuming the mixing as a linear process, least squares solutions (i.e., an impulse response and a gain envelope per track) were derived and further converted to the processor parameters. More recently, Colonel and Reiss [16, 17] employed differentiable signal processing [18, 19] and directly optimized the processor parameters with gradient descent. This approach allows more flexible choices of processors and optimization objectives, which could lead to better perceptual matches.

Ideally, the reverse engineering method should be able to comprehensively recover the real-world practice of music mixing. However, these prior works assumed a fixed chain of processors per source track and estimated only their parameters. Therefore, it is desirable to extend these methods to also estimate the compositional aspect of mixing, which can be represented with a *graph* of audio processors.

*To whom correspondence should be addressed, email: sh-lee@snu.ac.kr.

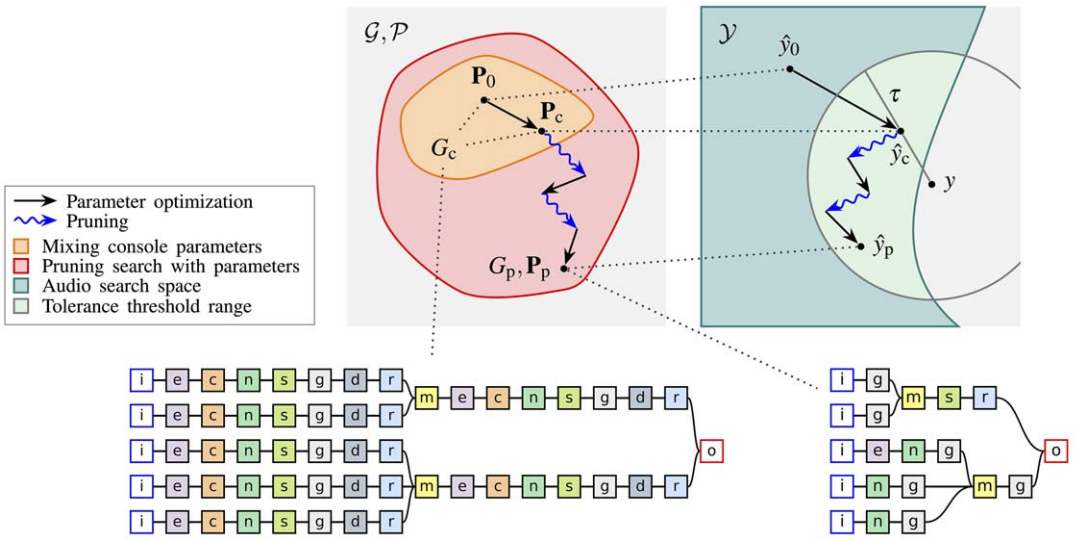


Fig. 1. The proposed mixing graph search method. First, a mixing console G_c is created (example shown in the lower left) and its parameters are also initialized, $\mathbf{P}_0$, which will give a mix $\hat{y}_0$ far from the target mix y. Gradient descent is performed to minimize the distance, obtaining console parameters $\mathbf{P}_c$ and mix $\hat{y}_c$. Next, pruning of the processors and optimization of the remaining processors are repeated. At each pruning stage, the processors are removed as much as possible while not degrading the match quality that the optimized mixing console, G_c and $\mathbf{P}_c$, achieved up to a certain tolerance threshold τ. This is illustrated inside the upper-right circle. An example of pruned graph G_p is shown in the lower right, and it will have its optimized parameters $\mathbf{P}_p$ and corresponding mix $\hat{y}_p$. `c` = compressor; `d` = multitap delay; `e` = equalizer; `g` = gain/panning; `i` = input; `m` = mix; `n` = noisegate; `o` = output; `r` = reverb; `s` = stereo imager.

0.1 Problem Formulation

In this paper, the mixing process is described with a graph $G = (V, E)$, where V and E denote the node and edge set, respectively. Each node $v_i \in V$ represents either a processor or an auxiliary module distinguished by a type attribute t_i. Each edge $e_{ij} \in E$ s directed and represents a "cable" that sends an output signal to another node as input. The graph G is restricted to be acyclic; hence, it is a directed acyclic graph (DAG).

Seven processor types are considered: equalizer `e`, compressor `c`, noisegate `n`, stereo imager `s`, gain/panning `g`, multitap delay `d`, and reverb `r`. Each processor takes an input audio u_i and a parameter vector p_i and computes an output y_i. The auxiliary modules include an input `i` that outputs a dry source s_k, a mix `m` that sums the incoming signals, and an output `o` whose sum of the inputs is the final output $\hat{y}$. A series of nodes is denoted with a string (e.g., `ecns`).

The reverse engineering task is defined as follows: given the dry sources $\mathbf{S} = [s_1, \ldots, s_K]$ and the ground truth mix y, the objective is to identify a graph G along with all processor parameters $\mathbf{P}$ such that processing the sources produces a mix $\hat{y} = G(\mathbf{S}; \mathbf{P})$ that closely approximates the original mix y. To measure the similarity, an audio domain loss function L_a is employed. Additionally, some regularization L_r is needed to restrict the graph and parameters to be in a reasonable range (e.g., being not too large). Therefore, the full optimization objective is given as

$$G^*, \mathbf{P}^* = \arg\min_{G,\mathbf{P}} \left[L_a(\hat{y}, y) + L_r(G, \mathbf{P}) \right]. \quad (1)$$

0.2 Proposed Strategy

Exploring candidate graphs without restrictions may appear appealing. However, the combinatorial nature of graphs results in an extremely large search space, and the need to simultaneously optimize processor parameters further complicates the task. Also, many graph-parameter pairs can achieve similar match quality, making the task ill posed and underdetermined. Therefore, it is beneficial to impose restrictions (e.g., favoring commonly used structures) to narrow the search space and improve practicality.

To this end, the authors propose a search method based on pruning (see Fig. 1). Following the approach of Colonel and Reiss [16, 17], a "mixing console" G_c is created first. To each input source, the mixing console applies a processor chain that consists of all the seven processors in the fixed order. It subgroups the processed sources with mix `m`, applies the chain again, and sums the processed subgroups to obtain a final mix $\hat{y}$. It is assumed that the subgrouping information is known a priori. Note that this structure resembles the traditional hybrid mixing console [20]. The differentiable signal processing [18, 19] is applied to implement the entire mixing in the automatic differentiation framework (i.e., `PyTorch` [21]). This allows for the optimization of all console parameters $\mathbf{P}_c$ with given objectives via gradient descent. After this, the authors proceed to the pruning stage, where the number of processors is reduced as much as possible and the match quality remains within a predefined tolerance τ. The pruning of a node is defined as its removal and the rerouting of its edges in a way that is equivalent to setting them to "bypass." To this

end, the iterative method [22] proposed for neural network pruning is adopted. That is, the remaining parameters are optimized after the pruning, and this process is repeated multiple times.

Casting the graph search into pruning reduces the search space, but entails several tradeoffs. First, the pruning only removes the processors from the predefined processor chains (i.e., it does not consider all other possibilities). As a result, the pruned graph G_{p} does not exceed the match quality of the original mixing console G_{c}. Nevertheless, it aligns with real-world practices by selectively applying appropriate processors, promoting sparsity, and enhancing the interpretability of the estimated mixing process. It also significantly reduces search cost and results in a practical algorithm, which might be challenging with other alternatives [23, 24]. The evaluation results show that about 67% of the processors can be removed while having little perceptual degradation.

Making the search algorithm fast and efficient is crucial to applying it to large-scale mixing data. To achieve this, two strategies are employed. First, the search repeatedly computes the mix, which is a major source of bottleneck. Hence, an algorithm is derived to accelerate this process by computing multiple processors within the graph in parallel. This differs from prior implementations [18, 25] that compute every processor "one by one." Note that the graphs here are much larger and more parallelizable than those from previous works [18, 25, 24, 14], yielding significant speedup with the proposed method.

Second, pruning is another source of bottleneck because it includes many trials, each involving audio processing and loss evaluations. Thus, efficient pruning methods are developed. This process requires identification of a subset of nodes that minimally impact match quality. To this end, each processor's "dry/wet" parameter is interpreted as an approximate importance score and is leveraged to select pruning candidates. This method is combined with a brute-force node-by-node check, deriving a hybrid method that has a reasonable trade-off between the computational cost and resulting sparsity.

The proposed search method was initially presented in a conference paper [26], and some technical details were provided in a companion demonstration paper [27]. This journal article combines these two, aiming to provide a complete description and unified viewpoint of the framework. Furthermore, extended evaluations were conducted. The proposed methods are validated with a diverse set of objective metrics [28–31], making the quantitative analysis more rigorous. Additional experiments and case studies also reveal the behavior of the pruning algorithm in more detail. Subjective listening test results [32, 33] are also reported, revealing how the mixing consoles and their pruned versions perform in perceptual similarity. The results are analyzed to reveal the relationship between subjective similarity and widely used objective metrics. These provide a deeper understanding, including the strengths and weaknesses of the proposed method, hinting at how potential improvements can be made.

0.3 Open-Source Implementation

Two supplementary materials are provided. The first one, relevant to SECS. 2 and 3, covers the efficient computation of audio processing graphs with `PyTorch`. It was initially developed for the presented pruning task and is now fully extended as a general-purpose library.[1] In this paper, the main technical details used for the pruning are discussed. Samples and implementation of the graph pruning (SECS. 4, 5, and 6) are available in the separate material.[2]

1 RELATED WORKS

1.1 Composition of Audio Processors

Although this work focuses on music mixing, it is valuable to review recent works that incorporate the compositional aspects of musical signal processing in a broader scope, such as sound matching [25, 34, 24], effect chain estimation [35–38, 14], and music mixing estimation [13, 14, 17]. Table 1 highlights the key differences among these approaches. These works vary in their tasks, domains, processors, graph structures, and estimation methods. Each task is denoted with $[A] \rightarrow [B]$, where A and B denote given references and estimation targets, respectively. For example, if the references are dry sources and a wet mixture, this task becomes reverse engineering [14, 35, 36, 17]. Some fixed the graph and estimated only the parameters [13, 25, 17, 18, 38]. Others tried to predict the graph [36] or both [14, 35, 24, 34]. Also, with several exceptions [14, 24], most works consider simple compositions (e.g., serial chains) that do not require graph terminology. Yet, those cases are treated as simple graphs. This work is the first graph reverse engineering method based on end-to-end parameter optimization with gradient descent.

1.2 Differentiable Signal Processing

In the audio signal processing literature, "differentiable" has been used as an umbrella term that refers to all methods that calculate the exact or approximate gradients of the audio processor. The most straightforward method is to implement the processors in an automatic differentiation framework (e.g., `PyTorch` [21]). One common application of differentiable processors is to combine them with neural networks. Thus, converting the processors to 1) be "graphics processing unit (GPU)–friendly" and 2) provide meaningful gradients has been an active research topic [17, 39–43, 19]. A notable example is the sampling frequency response of an infinite impulse response (IIR) filter to sidestep the recurrent calculation and speed up the computation [42, 43]. For processors that are not linear time-invariant (LTI), other approximation techniques have been proposed (e.g., replacing the nonlinear recurrent part with an IIR filter [39] or assuming frame-wise LTI to a linear time-varying system [41]).

If all processors are differentiable, so is the entire audio processing graph composed with them due to the chain rule.

[1] https://sh-lee97.github.io/grafx.
[2] https://sh-lee97.github.io/grafx-prune.

Table 1. A brief summary and comparison of previous works on the estimation of compositional audio signal processing.

[25]	Task and domain	Sound matching $[x] \rightarrow [\mathbf{P}]$. The synthesizer parameters $\mathbf{P}$ were estimated to match the reference (target) audio x.
	Processors	Oscillators, envelope generators, and filters that allow parameter modulation as an optional input.
	Graph	Any predefined DAG. For example, a subtractive synthesizer that comprises two oscillators, one amplitude envelope, and one low-pass filter were used in the experiments.
	Method	Trained a single neural backbone for the reference encoding, followed by multiple prediction heads for the parameters. Optimized with a parameter loss and spectral loss, where the latter is calculated with every intermediate output.
[34]	Task and domain	Sound matching $[x] \rightarrow [\mathbf{P}]$. A frequency-modulation (FM) synthesizer matches recordings of monophonic instruments (violin, flute, and trumpet). Estimates parameters of an operator graph that is empirically searched and selected.
	Processors	Differentiable sinusoidal oscillators, each used as a carrier or modulator, predefined frequencies. An additional FIR reverb is added to the FM graph output for postprocessing.
	Graph	DAGs with at most six operators. Different graphs for different target instruments.
	Method	Trained a convolutional neural network that estimates envelopes from the target loudness and pitch.
[24]	Task and domain	Sound matching $[x] \rightarrow [G, \mathbf{P}]$. Similar setup to the above [34] plus additional estimation of the operator graph G.
	Processors	Identical to [34], except for the frequency ratio that can be searched.
	Graph	A subgraph of a supergraph, which resembles a multilayer perceptron (modulator layers followed by a carrier layer).
	Method	Trained a parameter estimator for the supergraph and found the appropriate subgraph G with an evolutionary search.
[35]	Task and domain	Reverse engineering $[s, y] \rightarrow [G, \mathbf{P}]$ of an audio effect chain from a subtractive synthesizer (commercial plugin).
	Processors	Five audio effects: compressor, distortion, equalizer, phaser, and reverb. Nondifferentiable implementations.
	Graph	Chain of audio effects generated with no duplicate types (therefore 32 possible combinations) and random order.
	Method	Trained a next-effect predictor and parameter estimator in a supervised (teacher-forcing) manner.
[36]	Task and domain	Blind estimation $[y] \rightarrow [G]$ and reverse engineering $[s, y] \rightarrow [G]$ of guitar effect chains.
	Processors	Thirteen guitar effects, including nonlinear processors, modulation effects, ambiance effects, and equalizer filters.
	Graph	A chain of guitar effects. Maximum of five processors and a total of 221 possible combinations.
	Method	Trained a convolutional neural network with synthetic data to predict the correct combination.
[37]	Task and domain	Blind estimation $[y] \rightarrow [G, \mathbf{P}, s]$ of audio effect chains, the original dry signal, and all the intermediate signals.
	Processors	Four guitar effects: distortion, delay, chorus, and reverb.
	Graph	A chain of audio effects.
	Method	Trained a hybrid waveform-spectrogram transformer that estimates each processing step (type, parameter, and signal). The full inference is done autoregressively.
[13]	Task and domain	Automatic mixing $[\mathbf{S}] \rightarrow [\mathbf{P}]$. Estimated parameters of fixed processing chains from source tracks ($K \leq 16$).
	Processors	Seven differentiable processors, where four (gain, polarity, fader, and panning) were implemented exactly. A combined effect of the remaining three (equalizer, compressor, and reverb) was approximated with a single pretrained neural network.
	Graph	Tree structure: applied a fixed chain of the seven processors for each track and then summed the chain outputs together.
	Method	Trained a parameter estimator (convolutional neural network) with a spectrogram loss end to end.
[38]	Task and domain	Reverse engineering of music mastering $[s, y] \rightarrow [\mathbf{P}]$.
	Processors	A multiband compressor, graphic equalizer, and limiter. Gradient approximated with a finite difference method.
	Graph	A serial chain of the processors.
	Method	Optimized parameters with gradient descent.
[14]	Task and domain	Blind estimation $[y] \rightarrow [G, \mathbf{P}]$ and reverse engineering $[\mathbf{S}, y] \rightarrow [G, \mathbf{P}]$. Estimates both graph and its parameters for singing voice effect ($K = 1$) or drum mixing ($K \leq 6$).
	Processors	A total of 33 processors, including linear filters, nonlinear filters, and control signal generators. Some are multiple-input multiple-output processors, (e.g., allowing auxiliary modulations). Nondifferentiable implementations.
	Graph	Complex DAG; splits (e.g., multiband processing) and merges (e.g., sum and modulation). Thirty processors maximum.
	Method	Trained a convolutional neural network–based reference encoder and a transformer variant for graph decoding and parameter estimation. Both were jointly trained via direct supervision of synthetic graphs (e.g., parameter loss).

Table 1. *Continued.*

[17]	Task and domain	Reverse engineering $[\mathbf{S}, y] \rightarrow [\mathbf{P}]$ of music mixing. Estimated parameters of a fixed chain for each track.
	Processors	Six differentiable processors: gain, equalizer, compressor, distortion, panning, and reverb.
	Graph	A chain of five processors (all of the above types except the reverb) for each dry track (any other DAG can also be used). The reverb is used for the mixed sum.
	Method	Parameters were optimized with spectrogram loss end to end via gradient descent.
Ours	Task and domain	Reverse engineering $[\mathbf{S}, y] \rightarrow [G, \mathbf{P}]$ of music mixing. Estimated a chain of processors and their parameters for each track and submix where $K \leq 130$.
	Processors	Seven differentiable processors: gain/panning, stereo imager, equalizer, reverb, compressor, noisegate, and delay.
	Graph	A tree of processing chains with a subgrouping structure. Processors can be omitted but should follow the fixed order.
	Method	Joint estimation of the soft masks (dry/wet weights) and processor parameters. Optimized with the spectrogram loss (and additional regularizations) end to end via gradient descent. Accompanied by hard pruning stages.

Therefore, the only remaining practical consideration is optimizing the computation speed, which is tackled here with the batched processing. Finally, one might be interested in differentiation with respect to the graph structure. The proposed method performs this to a limited extent; pruning is a binary operation that modifies the graph structure, which can be relaxed to a continuous dry/wet weight and optimized via gradient descent jointly with other parameters.

1.3 Graph Search

Various research domain exists that develop computational methods for searching graphs with desired properties. For instance, neural network pruning [22, 44] aims to find a smaller, computationally efficient subnetwork that retains the performance of the full network. This is conceptually identical to the approach here. On the other hand, neural architecture search aims to find a neural network architecture that achieves improved performance [45]. In this case, the search space consists of graphs, with each node (or edge) representing one of the primitive neural network layers. One particularly relevant work to this paper is a differentiable architecture search [23], which relaxes the choice of each layer to a categorical distribution and optimizes it via gradient descent. Theoretically, the pruning method in this study can be naturally extended to this approach; one only need to change the two-way choice (prune or not) to $(N+1)$-way (bypass or one of N processors). A differentiable architecture search is clearly more flexible and general. However, it also greatly increases the computational cost because it requires computation of all N processors to compute their weight sum for every node.

Another related domain is the generation/design of molecules with desired chemical properties [46]. One dominant approach for this task is to use reinforcement learning (RL), which estimates each graph by making a sequence of decisions (e.g., adding nodes and edges [47]). RL is an attractive choice because one can be completely free with prior assumptions on graphs and use arbitrary quality measures (rewards) that are not differentiable. However, applying RL to this task has a risk of obtaining nontrivial mixing graphs that are difficult for practitioners to interpret. Also, it may need much larger computational resources to explore the search space sufficiently.

It is noted that another possible strategy for graph estimation is simply to train a neural network as a graph estimator in a supervised manner [14]. However, there is no publicly available ground-truth graphs for music mixes that can be used for training. Hence, they should be synthetically generated with a predefined set of rules. This risks training the model with a data distribution that is different from the real world. Moreover, unlike the prior work [14], which only considered small graph sizes, real-world multitrack mixes typically involve significantly more source tracks and processors, making the synthesis of realistic data even more challenging.

2 DIFFERENTIABLE PROCESSORS

Each processor $v_i \in V$ takes an input $u_i[n]$ and computes an output audio $y_i[n]$ (both are stereo; n denotes a discrete time index) as follows:

$$\bar{y}_i[n] = f_i(u_i[n], \bar{p}_i), \tag{2}$$

$$y_i[n] = w_i \bar{y}_i[n] + (1 - w_i) u_i[n]. \tag{3}$$

In other words, main processing f_i with parameters $\bar{p}_i$ produces an intermediate output $\bar{y}_i$. Then, this output is further mixed with the input with a dry/wet weight $w_i \in [0, 1]$. Separating the dry/wet mix from the other processing is helpful for the later discussion on pruning. The full parameter is denoted as $p_i = (\bar{p}_i, w_i)$ and is used when the distinction between the two is unnecessary. Because the processor v_i is used inside the graph $G = (V, E)$, its input will be given as the sum of other nodes' outputs:

$$u_i[n] = \sum_{j \in \mathcal{N}(i)} y_j[n]. \tag{4}$$

Here, $\mathcal{N}(i)$ denotes a set of node indices that is connected to processor v_i; in other words, $e_{ji} \in E$.

2.1 Gain/Panning

Each input channel is multiplied with a learnable gain. The parameter $\bar{p}_g \in \mathbb{R}^2$ is the concatenation of two loga-

rithmic gain parameters (for left and right channels). The output is given as

$$\bar{y}[n] = \exp(\bar{p}_{\mathtt{g}}) \cdot u[n]. \tag{5}$$

2.2 Stereo Imager

The stereo width (SW) is adjusted by altering the loudness of the side channel (left minus right) using a single logarithmic gain parameter. The mid and side signals are calculated as

$$y_{\mathrm{m}}[n] = u_{\mathrm{l}}[n] + u_{\mathrm{r}}[n], \tag{6}$$

$$y_{\mathrm{s}}[n] = \exp(\bar{p}_{\mathtt{s}}) \cdot (u_{\mathrm{l}}[n] - u_{\mathrm{r}}[n]). \tag{7}$$

Then, these are converted back to the stereo output with

$$\bar{y}_{\mathrm{l}}[n] = (y_{\mathrm{m}}[n] + y_{\mathrm{s}}[n])/2, \tag{8}$$

$$\bar{y}_{\mathrm{r}}[n] = (y_{\mathrm{m}}[n] - y_{\mathrm{s}}[n])/2. \tag{9}$$

2.3 Equalizer

A zero-phase finite impulse response (FIR) filter is employed, which is parameterized by its log-magnitudes $\bar{p}_{\mathtt{e}}$ in the frequency domain. One can obtain the time-domain response with exponentiation followed by inverse fast Fourier transform (FFT) and Hann-windowing $v^{\mathrm{Hann}}[n]$. As a result, the length-N FIR is

$$h_{\mathtt{e}}[n] = v^{\mathrm{Hann}}[n] \cdot \frac{1}{N} \sum_{k=0}^{N-1} \exp \bar{p}_{\mathtt{e}}[k] \cdot w_N^{kn}, \tag{10}$$

where $-(N+1)/2 \le n \le (N+1)/2$ and $w_N = \exp(j \cdot 2\pi/N)$. The same FIR is convolved to the left and right input channels to compute the output:

$$\bar{y}_{\mathrm{x}}[n] = u_{\mathrm{x}}[n] * h_{\mathtt{e}}[n] \quad (\mathrm{x} \in \{\mathrm{l}, \mathrm{r}\}). \tag{11}$$

The FIR length is set to $N = 2047$ (i.e., the parameter $[\bar{p}_{\mathtt{e}}]$ is a 1024-dimensional vector). For all causal convolutions, `FlashFFTConv` [48] is used to speed up the processing and save memory.

2.4 Reverb

A stereo FIR filter, where each FIR is a variant of the time-varying filtered noise [18], is employed. First, two uniform noise signals, $u_{\mathrm{m}}[n]$ and $u_{\mathrm{s}}[n] \sim \mathcal{U}(-1, 1)$, one for the mid channel and the other one for the side channel, respectively, are created (both are 2s long). Next, each noise is converted into its short-time Fourier transform (STFT) $U_{\mathrm{x}}[k, m]$ and multiplied with an independently parameterized magnitude mask $M_{\mathrm{x}}[k, m]$.

$$H_{\mathrm{x}}[k, m] = U_{\mathrm{x}}[k, m] \odot M_{\mathrm{x}}[k, m] \quad (\mathrm{x} \in \{\mathrm{m}, \mathrm{s}\}). \tag{12}$$

Here, k and m denote frequency and time frame indices, respectively. Each mask M_{x} is created with two parameters: a log-magnitude vector $H_{\mathrm{x}}^0[k]$ that corresponds to the initial coloration and another log-magnitude vector $H_{\mathrm{x}}^{\Delta}[k]$ that models the frequency-dependent decay of the reverb:

$$M_{\mathrm{x}}[k, m] = \exp(H_{\mathrm{x}}^0[k] + (m-1)H_{\mathrm{x}}^{\Delta}[k]). \tag{13}$$

The masked STFTs, H_{m} and H_{s}, are transformed back to the time-domain responses with the inverse STFT and further converted to the stereo FIR $h_{\mathtt{r}}[n]$. Channel-wise convolutions between the input $u[n]$ and the FIR yield the desired output $\bar{y}[n]$. The FFT and hop sizes are set to 384 and 192, respectively. Each reverb processor has 768 parameters (two channels, each with two log-magnitude vectors of size 192).

2.5 Compressor

The canonical implementation of the digital feed-forward compressor [49] is followed. First, a mid-channel signal $u_{\mathrm{m}}[n]$ is calculated by summing the left and right channels. Next, the signal is squared and smoothed with the following recursive filter called "ballistics":

$$g_u[n] = \alpha[n] g_u[n-1] + (1 - \alpha[n]) u_{\mathrm{m}}^2[n]. \tag{14}$$

In standard compressors, values of the smoothing coefficients $\alpha[n]$ are set differently for an "attack" and "release" phase (where $g_u[n]$ increases and decreases, respectively). However, this part bottlenecks the computation speed on a GPU. To mitigate this, the authors follow the recent work [39] and set the coefficients to a single parameter α. This simplification makes the process a one-pole IIR filter whose impulse response is simply given as

$$h^{\mathrm{env}}[n] = (1 - \alpha)\alpha^n. \tag{15}$$

Then, the energy envelope $g_u[n]$ can be computed by a simple convolution of the energy signal $u_{\mathrm{m}}^2[n]$ with the above IIR $h^{\mathrm{env}}[n]$ truncated to length N. Next, the compressed energy envelope $G_{\bar{y}}[n]$ is calculated from the log of the input envelope $G_u[n] = \log g_u[n]$. A quadratic knee that interpolates the compression and the bypass region is used. For a given threshold T and half of the knee width W,

$$G_{\mathrm{y}}[n] = \begin{cases} G_{\bar{y}}^{\mathrm{above}}[n] & G_u[n] \ge T + W, \\ G_{\bar{y}}^{\mathrm{mid}}[n] & T - W \le G_u[n] < T + W, \\ G_{\bar{y}}^{\mathrm{below}}[n] & G_u[n] < T - W \end{cases} \tag{16}$$

where, for a given compression ratio R, each term is

$$G_{\bar{y}}^{\mathrm{above}}[n] = T + \frac{G_u[n] - T}{R}, \tag{17}$$

$$G_{\bar{y}}^{\mathrm{mid}}[n] = G_u[n] + \left(\frac{1}{R} - 1\right) \frac{(G_u[n] - T + W)^2}{4W}, \tag{18}$$

and $G_{\bar{y}}^{\mathrm{below}}[n] = G_u[n]$. Finally, the output is given as

$$\bar{y}_{\mathrm{x}}[n] = \exp(G_{\bar{y}}[n] - G_u[n]) \cdot u_{\mathrm{x}}[n] \quad (\mathrm{x} \in \{\mathrm{l}, \mathrm{r}\}). \tag{19}$$

The scalar parameters, α, T, W, and R, form the compressor's parameter vector $\bar{p}_{\mathtt{c}} \in \mathbb{R}^4$.

2.6 Noisegate

Its implementation is identical to the compressor except for the output gain computation. That is $G_{\bar{y}}^{\mathrm{above}}[n] = G_u[n]$ and the remaining is given as

$$G_{\bar{y}}^{\mathrm{mid}}[n] = G_u[n] + (1 - R) \frac{(G_u[n] - T - W)^2}{4W}, \tag{20}$$

$$G_{\tilde{y}}^{\text{below}}[n] = T + R(G_u[n] - T). \tag{21}$$

2.7 Multitap Delay

Some mixes in the dataset used in this paper employ heavy delay (or "echo") effects. Hence, an FIR filter is introduced that can mimic this. The left and right channels use independent FIRs, and they are convolved with the input channel-wise. For simplicity, a single-channel case is considered in the following text and equation. Under this setup, the multitap delay's FIR is given as

$$h_{\mathrm{d}}[n] = \sum_{m=1}^{M} c_m[n] * \delta[n - d_m], \tag{22}$$

where $\delta[n]$ is a unit impulse signal and c_m is a 39-tap FIR implemented the same as the zero-phase equalizer. The authors use $M = 20$ delays, and each delay d_m is constrained to be at every 100-ms range. Not only the filter coefficients, but also each discrete delay length $d_m \in \mathbb{N}$ should be optimized with gradient descent. To achieve this, the property that each delayed impulse corresponds to a complex sinusoid in the frequency domain is utilized. A recent work [50] shows that the complex angular frequency $z_m \in \mathbb{C}$ of such a sinusoid can be optimized using gradient descent if it is allowed to be in the unit disk (i.e., $|z_m| \leq 1$). This finding is leveraged; from the angular frequency z_m, a damped sinusoid is computed. Then, its inverse FFT is used as a surrogate of the delayed unit impulse:

$$\delta[n - d_m] \approx \frac{1}{N} \sum_{k=0}^{N-1} z_m^k w_N^{kn}. \tag{23}$$

Note that this surrogate is not identical to the original delay unless the angular frequency is on the discrete bins of the unit disk. Therefore, it is used only for calculating the gradients with straight-through estimation [51]. Moreover, normalization is applied to the gradient, and a regularization that guides the parameters to increase their radius is added. Empirically, it is observed that this improves the performance. To summarize, each gradient is updated as follows:

$$\frac{\partial L}{\partial z_m^*} \leftarrow \operatorname{sgn}\left(\sum_n \frac{\partial L}{\partial h_{\mathrm{d}}[n]} \frac{\partial \tilde{h}_{\mathrm{d}}[n]}{\partial z_m^*}\right) + \gamma(|z_m| - 1)\operatorname{sgn}(z_m^*), \tag{24}$$

where $\tilde{h}_{\mathrm{d}}[n]$ is the surrogate FIR obtained with the continuous delay, $\operatorname{sgn}(z) = z/|z|$ and γ is a regularization strength set to 0.01. This multitap delay has a parameter vector $\bar{p}_{\mathrm{d}}$ of size 880 (20 delays for each channel; each delay with a real and imaginary part of the sinusoid's angular frequency and 20 log-magnitude bins of the FIR filter).

3 BATCHED PROCESSING ON GRAPHS

An efficient implementation of computing the output y from the source audio $\mathbf{S}$ and graph G and its parameters $\mathbf{P}$ is presented. In practice, each graph G is represented with a vector of node types $\mathbf{T} \in \mathbb{N}^{|V|}$ and a tensor of edge indices $\mathbf{E} \in \mathbb{N}^{2\times|E|}$. The collection of parameters $\mathbf{P}$ is implemented as a dictionary, where each key-value pair corresponds to the node type t and parameters $\mathbf{P}[t] \in \mathbb{R}^{|V_t|\times N_t}$ of that type with a single tensor where $|V_t|$ and N_t denote the number of type-t nodes and parameters, respectively. Note that $\mathbf{w}$ and $\bar{\mathbf{P}}$ are used to denote all the collection of dry/wet weights and the remaining parameters [i.e., $\mathbf{P} = (\bar{\mathbf{P}}, \mathbf{w})$, respectively.

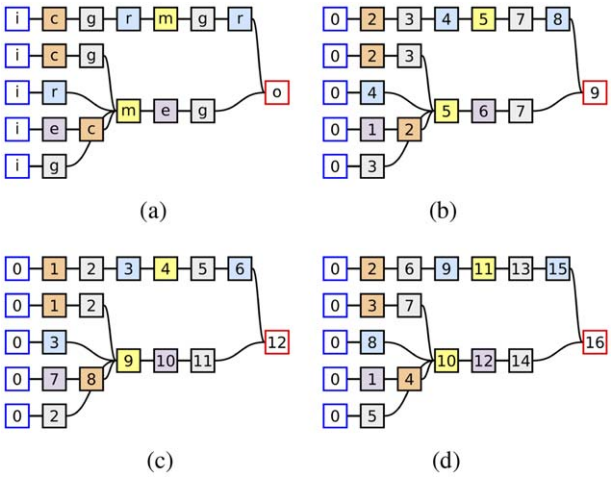


Fig. 2. Various schedules for batched node processing. For each schedule, the processing orders are shown inside the nodes. (a) Target graph; (b) optimal schedule; (c) greedy schedule; (d) one-by-one processing. c = compressor; e = equalizer; g = gain/panning; i = input; m = mix; o = output; r = reverb.

3.1 Node Subset Sequence

Instead of computing each node one by one [25, 18], one can batch process multiple nodes in parallel. Specifically, a sequence of node subsets $V_0, \ldots, V_N \subset V$ that satisfies the following can be used for the batched processing.

(i) It is a *partition*: $\cup_n V_n = V$ and $V_n \cap V_m = \emptyset$ if $n \neq m$.
(ii) It is *causal*: there is no path from $u \in V_n$ to $v \in V_m$ if $n \geq m$.
(iii) Each subset V_n is *homogeneous*: nodes with the same type t_n can be in the subset.

With this sequence, each subset's output $\mathbf{Y}_n$ can be batch computed by gathering the inputs, aggregating them if necessary, and computing the processed result. The first and last subsets, V_0 and V_N, can be set to collect all the input and output nodes, respectively. Then, the number of gather-aggregate-process iterations can be reduced from $|V|$ to N. Fig. 2 shows an example. Although the target graph comprises $|V| = 21$ nodes [Fig. 2(a)], there is a node subset sequence of $N = 9$ that computes one equalizer e, three compressors c, three gain/pannings g, and so on [Fig. 2(b)].

3.2 Type Scheduling

To maximize the batched processing, it is desirable to reduce the length of the subset sequence. When the type t_i of each subset V_i is fixed, one always wants to make the subset as maximal as possible. With this setup, a subset sequence becomes equivalent to a "type string" {e.g.,

Algorithm 1. Batched node processing.

Input: Types $\mathbf{T}$, edges $\mathbf{E}$, parameters $\mathbf{P}$, and inputs $\mathbf{S}$
Output: Output signal y and regularization loss L_r
1: $\bar{\mathbf{T}}$, $N \leftarrow$ ScheduleBatchedProcessing($\mathbf{T}$, $\mathbf{E}$)
2: $\sigma \leftarrow$ OptimizeNodeOrder($\bar{\mathbf{T}}$, $\mathbf{T}$, $\mathbf{E}$)
3: $\mathbf{T}$, $\mathbf{E}$, $\mathbf{P} \leftarrow$ Reorder (σ, $\mathbf{T}$, $\mathbf{E}$, $\mathbf{P}$)
4: $\mathbf{I}^\mathrm{G}$, $\mathbf{I}^\mathrm{P}$, $\mathbf{I}^\mathrm{A}$, $\mathbf{I}^\mathrm{S} \leftarrow$ GetReadWriteIndex($\bar{\mathbf{T}}$, $\mathbf{T}$, $\mathbf{E}$, $\mathbf{P}$)
5: $\mathbf{U}$, $L_\mathrm{r} \leftarrow$ Initialize ($\mathbf{S}$, $\mathbf{T}$), 0
6: **for** $n \leftarrow 1$ to N **do**
7: $\quad\bar{\mathbf{U}}_n \leftarrow$ Gather($\mathbf{U}$, $\mathbf{I}^\mathrm{G}_n$) `(index_select)`
8: $\quad\mathbf{U}_n \leftarrow$ Aggregate($\bar{\mathbf{U}}_n$, $\mathbf{I}^\mathrm{A}_n$) `(scatter)`
9: $\quad\mathbf{P}_n \leftarrow$ Gather($\mathbf{P}[\bar{t}_n]$, $\mathbf{I}^\mathrm{P}_n$) `(slice)`
10: $\quad\mathbf{Y}_n$, $L_{\mathrm{r},n} \leftarrow$ Process($\bar{t}_n$, $\mathbf{U}_n$, $\mathbf{P}_n$)
11: $\quad\mathbf{U} \leftarrow$ Store($\mathbf{U}$, $\mathbf{Y}_n$, $\mathbf{I}^\mathrm{S}_n$) `(slice)`
12: $\quad L_\mathrm{r} \leftarrow L_\mathrm{r} + L_{\mathrm{r},\,n}$
13: **end for**
14: **return** $\mathbf{Y}_N$, L_r

`iecgrmegro` for the previous example [Fig. 2(b)]}. For the mixing consoles and their pruned versions, it is known that the optimal node sequence is a subsequence of `iecnsgdrmecnsgdro`. If such information is not available, one must solve a more general scheduling problem. Given that the search tree for the shortest sequence grows exponentially, a brute-force search becomes computationally impractical for most graphs. As an alternative, a greedy approach can be employed, where the type with the largest number of computable nodes is selected at each step [Fig. 2(c)]. For improved efficiency and accuracy, a more advanced beam search method can be utilized.

3.3 Implementation Details

Algorithm 1 describes the output computation (inside the following parentheses denote the line numbers). First, one computes a list of node types $\bar{\mathbf{T}} \in \mathbb{N}^{N+1}$i.e., a batched node processing schedule (1). Because the computation involves multiple memory reads/writes during (6–13), one changes the node orderings to maximize contiguous memory access (2). This procedure allows memory access via `slice`; see the comments of Algorithm 1. After updating the graph tensors with the new order σ (3), list of indices for the tensor read/writes, $\mathbf{I}^\mathrm{G}$, $\mathbf{I}^\mathrm{P}$, $\mathbf{I}^\mathrm{A}$, and $\mathbf{I}^\mathrm{S}$ are retrieved (4). These preprocessing steps (1–4) can be done on a central processing unit with multiple separate threads (i.e., they do not consume the GPU compute and memory). After the preprocessing, one creates a tensor $\mathbf{U}$ with a shape $|V| \times 2 \times L$ that stores all the node outputs from the processors. Because all the inputs are in the first partition V_0, this tensor is initialized as $\mathbf{U} = \mathbf{S} \oplus \mathbf{0}$ where $\oplus$ denotes concatenation. A regularization loss L_r that aggregates each iteration's losses is also initialized (5).

What remains is the main loop, repeating the processor output computation and necessary reads/writes (6–13). During each n^th iteration, one gathers the previous outputs $\bar{\mathbf{U}}_n$ routed to the current partition nodes by accessing the intermediate tensor $\mathbf{U}$ using the index $\mathbf{I}^\mathrm{G}_n$ with the `index_select` operation (7). If needed, these outputs are aggregated using `scatter` (8). Similarly, the parameter tensor $\mathbf{P}_n$ is retrieved using the corresponding index $\mathbf{I}^\mathrm{P}_n$; the node reordering strategy (3) simplifies this operation to a `slice`, which is more efficient than the usual `index_select`. With the input audio $\mathbf{U}_n \in \mathbb{R}^{|V_n| \times 2 \times L}$ and parameters $\mathbf{P}_n \in \mathbb{R}^{|V_n| \times N_t}$ obtained, the node outputs $\mathbf{Y}_n$ (10) are batch processed. These outputs are then stored back into the intermediate tensor $\mathbf{U}$ using the `slice` index $\mathbf{I}^\mathrm{S}_n$ (11), ensuring they are accessible for subsequent steps. Additionally, the intermediate regularization loss $L_{\mathrm{r},\,n}$ is added in the computation (12). At the end of the iteration, all node outputs are saved in $\mathbf{U}$. The final outputs of the graph are given by the outputs of the last step $y = \mathbf{Y}_N \in \mathbb{R}^{1 \times 2 \times L}$ because the final node partition V_N has the output node (14).

4 MUSIC MIXING CONSOLES

Before exploring the pruning, the match quality of the mixing consoles are evaluated, which will serve as a performance upper bound. In addition to evaluating the full console, the contribution of each processor type to the overall performance is analyzed. To achieve this, the authors begin with a base graph that simply sums all the inputs. Then, each processor type is sequentially added to every processor chain (refer to Tables 2 and 3). Each preliminary graph is then optimized and evaluated for every song in the validation set. This experiment is similar to one conducted in Colonel and Reiss [17], albeit the structure of mixing consoles and the processors differ.

4.1 Optimization

The graph parameters and dry/wet weights are optimized with the following audio domain loss,

$$L_\mathrm{a} = \alpha_\mathrm{lr} L_\mathrm{lr} + \alpha_\mathrm{m} L_\mathrm{m} + \alpha_\mathrm{s} L_\mathrm{s}, \tag{25}$$

where each term L_x is a multiresolution STFT (MRSTFT) loss [52] ($\mathrm{x} \in \{\mathrm{lr}, \mathrm{m}, \mathrm{s}\}$, lr: left/right, m: mid, s: side)

$$L_\mathrm{x} = \sum_{i=1}^{I} \left[\frac{\| \log Y^{(i)}_\mathrm{x} - \log \hat{Y}^{(i)}_\mathrm{x} \|_1}{N} + \frac{\| Y^{(i)}_\mathrm{x} - \hat{Y}^{(i)}_\mathrm{x} \|_F}{\| Y^{(i)}_\mathrm{x} \|_F} \right]. \tag{26}$$

Here, $Y^{(i)}_\mathrm{x}$ and $\hat{Y}^{(i)}_\mathrm{x}$ represent the i^th Mel spectrograms of the target and predicted mix, respectively. The terms N, $\| \cdot \|_1$, and $\| \cdot \|_F$ refer to the number of frames, the l_1 norm, and the Frobenius norm, respectively. FFT sizes of 512, 1024, and 4096 are used, with hop sizes set to 1/4 of their respective FFT sizes. The number of Mel filterbanks is 96 for all scales. Before each STFT, A-weighting is applied [53]. The loss weights are $\alpha_\mathrm{lr} = 0.5$, $\alpha_\mathrm{m} = 0.25$, and $\alpha_\mathrm{s} = 0.25$. The authors use `auraloss` [54] for the implementation. Additionally, the following regularization term is included to encourage gain staging, a common practice to ensure the total energy of the input and output remains approximately equal:

$$L_\mathrm{g} = \sum\nolimits_{v_i \in V_\mathrm{g}} \left| \log \| (\bar{y}_i)_\mathrm{m} \|_2 - \log \| (u_i)_\mathrm{m} \|_2 \right|, \tag{27}$$

where $(\,\cdot\,)_\mathrm{m}$ and $\| \cdot \|_2$ denote mid channel and l_2 norm, respectively. This regularization is applied to all equalizers, reverbs, and multitap delays (denoted as V_g). This approach enables one to remove redundant gains introduced by these

Table 2. Matching performances of the mixing consoles using different processor type configurations for data set `MedleyDB`.

		MRSTFT ↓						MIR Feature Distance ↓				
		L_a	L_{lr}	L_m	L_s	FAD ↓	SI-SDR ↑	d_{RMS}	d_{CR}	d_{SW}	d_{SI}	d_{BS}
Base graph (sum of dry sources)		50.7	1.42	1.42	198	3.57	−6.41	−4.68	0.662	−1.99	−2.22	2.46
+ Gain/panning	`g`	0.550	0.583	0.485	0.549	0.719	−7.20	−5.01	0.484	−3.52	−2.38	−1.56
+ Stereo imager	`sg`	0.541	0.564	0.483	0.553	0.733	−6.97	−5.02	0.488	−3.54	−2.48	−1.54
+ Equalizer	`e sg`	0.450	0.453	0.390	0.504	0.573	−5.25	−5.45	0.415	−3.71	−2.65	−1.62
+ Reverb	`e sg`	0.368	0.361	0.360	0.390	0.405	−3.94	−5.54	0.356	−4.17	−3.55	−1.66
+ Compressor	`ec sg r`	0.315	0.304	0.297	0.356	0.312	−2.95	−6.15	0.173	−4.33	−3.61	−1.52
+ Noisegate	`ec sg r`	0.302	0.288	0.281	0.353	0.264	−2.60	−6.12	0.065	−4.22	−3.55	−1.51
+ Multitap delay (full)	`ecnsg r`	0.296	0.288	0.284	0.324	0.222	−2.66	−6.17	0.093	−4.34	−3.78	−1.46

LTI processors and constrain the parameters to be within a reasonable range. Therefore, the full optimization loss is

$$L(\bar{\mathbf{P}}, \mathbf{w}) = L_a(\bar{\mathbf{P}}, \mathbf{w}) + \alpha_g L_g(\bar{\mathbf{P}}), \tag{28}$$

where the weight of the gain-staging regularization is set to $\alpha_g = 10^{-3}$ [a different notation was used from Eq. (1) to emphasize the optimized parameters]. Parameters are initialized with a Gaussian distribution $\mathcal{N}(0, 10^{-2})$. For dry/wet weights, a logistic sigmoid is further applied to center the initial values to 0.5 and restrict their range to (0, 1). Each console is optimized for 12,000 steps using AdamW optimizer [55] with a 0.01 learning rate. Each step randomly samples a 3.8-s segment, computes the mix $\hat{y}$, and compares it with the corresponding ground truth y (hence, the batch size is 1). The first second of each mix is omitted when calculating the loss; it is used only to "warm up" the processors with long states (e.g., compressors and reverbs).

4.2 Evaluation Metrics

The authors follow the recent work [30] and compare the optimized mix and the ground truth with the following metrics. First, Frechet audio distance (FAD), which calculates the distance between the feature distributions obtained with a pretrained audio encoder is employed. The authors also report the scale-invariant signal-to-distortion ratio (SI-SDR), widely used in source separation literature, which involves sample-wise comparison [29]. Furthermore, the mean square errors of simple analysis features used in music information retrieval (MIR) are measured. These are referred to as "MIR feature distances." The mix and its match are split into disjoint 8-s segments, $y^{(1)}, \ldots, y^{(M)}$ and $\hat{y}^{(1)}, \cdots, \hat{y}^{(M)}$. Then, the mean square errors of features extracted from those segments are calculated. The values are further compressed with logarithm as it was empirically found to correlate better to the subjective similarity. The same set of features in [30] is considered: root mean square (RMS), crest factor (CF), SW, stereo imbalance (SI), and bark spectrum (BS) [31]:

$$d_x = \log_{10}\left[\frac{1}{MK_x}\sum_{m=1}^{M}\left\|F_x(y^{(m)}) - F_x(\hat{y}^{(m)})\right\|_2^2\right], \tag{29}$$

where $x \in \{\text{RMS, CF, SW, SI, BS}\}$, $\|\cdot\|_2$ denotes l_2 norm, and K_x denotes the size of the feature x. Each feature reflects certain aspects of the mix. For example, the CF is defined as a ratio between the peak absolute value and the RMS of the waveform. Therefore, it could be used to compare the mixes and matches in terms of loudness dynamics.

4.3 Data

For each song, dry sources $\mathbf{S}$ and a mixture y are needed to run the optimization. Also, additional subgroup information is needed to construct the initial mixing console. Thus, the `MedleyDB` data set [4, 5] is utilized because as it provides all the necessary components. The `MixingSecrets` library [6] only provides the audio; therefore, the subgroup information is manually created. For evaluation purposes, a random 24-song subset is used for each data set, and the results are reported separately. Finally, to collect the graph data, the authors use the full data and also include a private `internal` data set crowdsourced from multiple

Table 3. Matching performances of the mixing consoles using different processor type configurations for data set `MixingSecrets`.

		MRSTFT ↓						MIR Feature Distance ↓				
		L_a	L_{lr}	L_m	L_s	FAD ↓	SI-SDR ↑	d_{RMS}	d_{CR}	d_{SW}	d_{SI}	d_{BS}
Base graph (sum of dry sources)		7.30	2.16	2.02	22.9	2.78	−13.8	−4.16	1.15	−1.66	−2.22	0.401
+ Gain/panning	`g`	0.876	0.856	0.819	0.973	1.49	−10.0	−4.63	1.21	−2.33	−1.82	−0.842
+ Stereo imager	`sg`	0.847	0.834	0.791	0.928	1.46	−10.3	−4.64	1.20	−2.27	−1.94	−0.855
+ Equalizer	`e sg`	0.699	0.698	0.622	0.780	0.933	−9.62	−5.00	1.14	−2.79	−2.20	−1.23
+ Reverb	`e sg`	0.614	0.601	0.579	0.674	0.642	−6.89	−5.07	1.04	−2.98	−2.74	−1.39
+ Compressor	`ec sg`	0.558	0.542	0.512	0.637	0.526	−6.10	−5.13	0.987	−2.81	−2.70	−1.20
+ Noisegate	`ec sg`	0.548	0.532	0.502	0.625	0.523	−7.04	−4.98	0.927	−2.61	−2.60	−1.17
+ Multitap delay (full)	`ecnsgdr`	0.545	0.529	0.502	0.618	0.506	−6.85	−4.98	0.985	−2.84	−2.66	−1.13

engineers, which comprises Western music recordings. A total of 1129 songs were collected (188, 472, and 579 songs for `MedleyDB`, `MixingSecrets`, and `internal`, respectively). Every dry source track and mix is stereo and resampled to a 30-kHz sampling rate.

4.4 Results

Tables 2 and 3 report the results for the `MedleyDB` and `MixingSecrets` evaluation subsets, respectively. The authors first report the `MedleyDB` results. First, the base graph reports an MRSTFT loss L_a of 50.7. The side-channel term L_s is especially large, reporting 198 because most source tracks are mono (hence, so is the naive sum), whereas the target mixes have wide stereo images. With the gain/pannings and stereo imagers, "rough mixes" with a loss of 0.541 are achieved. Next, the missing details are filled in by adding the remaining processor types. Each type reduces the audio loss, with the full mixing console achieving a value of 0.296. The same trend can be observed in the `MixingSecrets` results, albeit the audio loss is higher (0.545 for the full console), indicating that mixes from this data set are more challenging than `MedleyDB`'s.

In terms of other metrics, FAD is the only one that shows the same consistent improvement for each processor type. The SI-SDR is slightly inconsistent, and the values are very low even with the full console -4.50dB on average. Two causes are suspected: the MRSTFT loss does not model phases, and the dry tracks and target mix are not precisely aligned for some songs. The MIR feature distances are also inconsistent, but each processor tends to improve its relevant metrics. Although the compressor and noisegate reduce the CF distance d_{CR} (-0.183 and -0.108, respectively), they rather increase the BS distance d_{BS} (+0.14 and +0.01, respectively, on `MedleyDB`), possibly due to the properties of these processors that can modify the loudness dynamics but not the overall spectral balance. Similarly, because multitap delay has two FIRs independently parameterized for the left and right channels, they are effective for matching the stereo image, which is shown by the decreased SW distance d_{SW} and SI distance d_{SI} (-0.12 and -0.23, respectively). However, it has a risk of "smearing" the transient and harming the loudness dynamics (an increase of +0.028 in the CF distance d_{CR}). Nevertheless, it is noted that the average of all MIR feature distances on the two subsets consistently decreases, from -1.22 to -2.63.

Further qualitative analysis can be conducted by comparing log-magnitude spectrograms of the targets and mixing console matches and their errors (see the top eight rows of Fig. 10). Again, it is observed that adding each processor type improves the match. Furthermore, each song has different types that are more crucial; for the song `NewSkin` [Fig. 10(a)], there is significant error reduction when reverbs are introduced. However, this is not the case for `RockSteady` [Fig. 10(b)]. Instead, for the latter, different from the average trend, the multitap delays improve the match more than the reverbs. This is not surprising because the original mix introduces heavy delay effects.

5 MUSIC MIXING GRAPH SEARCH

The previous full mixing console G_c serves as an upper bound for matching performance. Now, the goal is to identify a sparser graph that maintains a similar match quality. This is achieved by pruning the console as much as possible and ensuring that the loss increase remains within a predefined tolerance threshold τ. This requirement is illustrated as a circle in Fig. 1. This objective is written as

$$\text{minimize } |V_p| \quad \text{s.t. } \min L_a(G_p) \leq \min L_a(G_c) + \tau, \tag{30}$$

where V_p represents the pruned graph's node set and $|\cdot|$ denotes its cardinality. The term $\min(\cdot)$ emphasizes that the focus here is on the optimized audio loss after pruning. Note that only processors are subject to pruning, whereas auxiliary nodes remain unchanged. Instead of the number of remaining nodes $|V_p|$, a pruning ratio defined as the number of pruned processors over the number of the initial console's processors will be reported.

5.1 Iterative Pruning

Finding the optimal solution V_p^*, a set with the smallest number of nodes, is computationally prohibitive due to the combinatorial nature of the problem (i.e., the number of the candidate graphs is a power of the initial mixing console). To mitigate this, it is assumed that the processors independently contribute to the match and adopt a greedy pruning strategy. Following the iterative approach [22], processors are incrementally removed whenever the tolerance condition is met. Ideally, intermediate pruned graphs should be fine-tuned before assessing the tolerance condition. However, for a reasonable computational complexity this step is omitted, paying the cost of potentially missing some removable processors.

The method used here method is described in Algorithm 2, with specific steps referenced in the following parentheses. The authors start from the previously described optimization of the mixing console $G_c = (V, E)$ and evaluate the final loss (1–3). The loss $L_a^{\min}$ acts as a threshold, with the tolerance τ defining acceptable degradation. Next, pruning and optimizing the remaining parameters and weights [i.e., fine-tuning (5–18)] are alternated. Each pruning stage involves multiple trials in which subsets of candidate processors V_{cand} are sampled from the remaining pool V_{pool} (8) and tested for removability (10). If the result satisfies the constraint, pruning is applied; otherwise, it is reverted (10–13). This process is repeated until the terminal condition (7) is met.

In implementation, pruning trials are simulated by multiplying the weight vector $\mathbf{w}$ with binary masks $\mathbf{m}$ and $\mathbf{m}_{cand}$ (9). After the trials, the actual pruning of nodes is performed so that unnecessary processing can be discarded and further optimization can be accelerated (16). Note that pruning occasionally improves the match quality. In such cases, the pruning threshold is updated accordingly (11).

Algorithm 2. Mixing graph search with iterative pruning.

Input: A mixing console G_c, dry tracks $\mathbf{S}$, and mixture y
Output: Pruned graph G_p and parameters $\mathbf{P} = (\bar{\mathbf{P}}, \mathbf{w})$

$\bar{\mathbf{P}}, \mathbf{w} \leftarrow \text{Initialize}(G_c)$
$\bar{\mathbf{P}}, \mathbf{w} \leftarrow \text{Train}(G_c, \bar{\mathbf{P}}, \mathbf{w}, \mathbf{S}, y)$
$L_a^{\min} \leftarrow \text{Evaluate}(G_c, \bar{\mathbf{P}}, \mathbf{w}, \mathbf{S}, y)$
$G_p \leftarrow G_c$
for $n \leftarrow 1$ to N_{iter} **do**
  $V_{\text{pool}}, \mathbf{m} \leftarrow \text{GetAllProcessors}(V), \mathbf{1}$
  while $\text{TryPrune}(V_{\text{pool}}, \mathbf{w}, \mathbf{m})$ **do**
    $V_{\text{cand}}, \mathbf{m}_{\text{cand}} \leftarrow \text{SampleCandidate}(V_{\text{pool}}, \mathbf{w})$
    $L_a \leftarrow \text{Evaluate}(G_p, \bar{\mathbf{P}}, \mathbf{w} \odot \mathbf{m} \odot \mathbf{m}_{\text{cand}}, \mathbf{S}, y)$
    if $L_a < L_a^{\min} + \tau$ **then**
      $L_a^{\min} \leftarrow \min(L_a^{\min}, L_a)$
      $\mathbf{m} \leftarrow \mathbf{m} \odot \mathbf{m}_{\text{cand}}$
    end if
    $V_{\text{pool}} = \text{UpdatePool}(V_{\text{pool}}, V_{\text{cand}})$
  end while
  $G_p, \bar{\mathbf{P}}, \mathbf{w} \leftarrow \text{Prune}(G_p, \bar{\mathbf{P}}, \mathbf{w}, \mathbf{m})$
  $\bar{\mathbf{P}}, \mathbf{w} \leftarrow \text{Train}(G_p, \bar{\mathbf{P}}, \mathbf{w}, \mathbf{S}, y)$
end for
return $G_p, \bar{\mathbf{P}}, \mathbf{w}$

Algorithm 3. Mixing graph search using dry/wet method.

Input: A mixing console G_c, dry tracks $\mathbf{S}$, and mixture y
Output: Pruned graph G_p and parameters $\mathbf{P} = (\bar{\mathbf{P}}, \mathbf{w})$

$\bar{\mathbf{P}}, \mathbf{w} \leftarrow \text{Initialize}(G_c)$
$\bar{\mathbf{P}}, \mathbf{w} \leftarrow \text{Train}(G_c, \bar{\mathbf{P}}, \mathbf{w}, \mathbf{S}, y)$
$L_a^{\min} \leftarrow \text{Evaluate}(G_c, \bar{\mathbf{P}}, \mathbf{w}, \mathbf{S}, y)$
$G_p \leftarrow G_c$
for $n \leftarrow 1$ **to** N_{iter} **do**
  $T_{\text{pool}}, \mathbf{m} \leftarrow \text{GetProcessorTypeSet}(V), \mathbf{1}$
  for t **in** T_{pool} **do**
    $V_t, \mathbf{w}_t \leftarrow \text{Filter}(V, t), \text{Filter}(\mathbf{w}, t)$
    $N_t, r_t \leftarrow |V_t|, 0.1$
  end for
  while $T_{\text{pool}} \neq \emptyset$ **do**
    $t \leftarrow \text{SampleType}(T_{\text{pool}})$
    $\bar{V}_t, \bar{\mathbf{m}} \leftarrow \text{GetLeastWeightNodes}(V_t, \mathbf{w}_t, \lfloor N_t r_t \rceil)$
    $L_a \leftarrow \text{Evaluate}(G_p, \bar{\mathbf{P}}, \mathbf{w} \odot \mathbf{m} \odot \bar{\mathbf{m}}, \mathbf{S}, y)$
    if $L_a < L_a^{\min} + \tau$ **then**
      $L_a^{\min} \leftarrow \min(L_a^{\min}, L_a)$
      $\mathbf{m} \leftarrow \mathbf{m} \odot \bar{\mathbf{m}}$
      $V_t \leftarrow V_t \setminus \bar{V}_t$
    else
      if $\lfloor N_t r_t \rceil > 1$ **then**
        $r_t \leftarrow r_t / 2$
      else
        $T_{\text{pool}} \leftarrow T_{\text{pool}} \setminus \{t\}$
      end if
    end if
  end while
  $G_p, \bar{\mathbf{P}}, \mathbf{w} \leftarrow \text{Prune}(G_p, \bar{\mathbf{P}}, \mathbf{w}, \mathbf{m})$
  $\bar{\mathbf{P}}, \mathbf{w} \leftarrow \text{Train}(G_p, \bar{\mathbf{P}}, \mathbf{w}, \mathbf{S}, y)$
end for
return $G_p, \bar{\mathbf{P}}, \mathbf{w}$

5.2 Candidate Sampling

Implementation-wise, the design choices are to decide how to sample a candidate set V_{cand} (8, 14), decide when to terminate the trials (7), and do some preprocessing (6; see colored lines in Algorithm 2). The following three approaches will be explored.

First, a simple brute-force method can be used; every processor can be tried one by one: $V_{\text{cand}} = \{v\}$ where $v \sim V_{\text{pool}}$. This is repeat ed until every processor in the pool V_{pool} has been tried. This granularity likely achieves high sparsity but demands a high computational cost; $|V_{\text{pool}}|$ trials should be run for each pruning stage.

Second, to improve efficiency from the above method, an informed guess of each node's importance is required. Intuitively, one can use each dry/wet weight w_i for approximate importance because $w_i = 0$ is equivalent to pruning v_i. This observation leads to the following variant: the dry/wet method, described in Algorithm 3 (the colored lines highlight the correspondence with Algorithm 2). For each pruning iteration, a set of remaining types T_{pool} and the number of remaining processors N_t for each type t are calculated. Then, for each trial, the authors sample a remaining type t (12) and choose the smallest weight type t processors as candidates $\bar{V}_t$ (13). The proportion of chosen candidates is controlled with a per-type ratio r_t initialized to 0.1. Then, an iterative search is performed to find the maximum proportion for each type. When the trial succeeds, those candidates are pruned and search for that type t (18) is continued. If the trial fails, the following is performed: if the number of candidates is larger than 1, halve the ratio for the next search of the type t is halved. Otherwise, the search of the type t is terminated by removing it from the pool T_{pool}. This method resembles the binary search but uses a more conservative initial proportion $r_t = 0.1$ other than 0.5. This avoids excessive pruning of early sampled types. Also, treating each type separately is motivated by the observation that the range of dry/wet values could be different for different types.

Finally, solely relying on the weight values could neglect some large-weight processors that can be pruned. This is mitigated by combining the above two methods, running the brute-force method for every fourth iteration, and running the dry/wet method for the remaining. The authors call this a hybrid method, which is, along with the tolerance of $\tau = 0.01$, the default choice throughout the experiments.

5.3 Optimization

The authors keep the same objectives used in the mixing console training and add sparsity regularization L_p, a l_1 norm of the weight $\mathbf{w}$, to promote the pruning:

$$L(\bar{\mathbf{P}}, \mathbf{w}) = L_a(\bar{\mathbf{P}}, \mathbf{w}) + \alpha_g L_g(\bar{\mathbf{P}}) + \alpha_p L_p(\mathbf{w}). \tag{31}$$

The optimization is begun by training the console for 6,000 steps. Following this, $N_{\text{iter}} = 12$ rounds of pruning are performed, with each round including 500 steps of fine-tuning. This results in the same total optimization steps as in the previous experiments. Although the full console optimization steps are halved, potentially leading to an increase

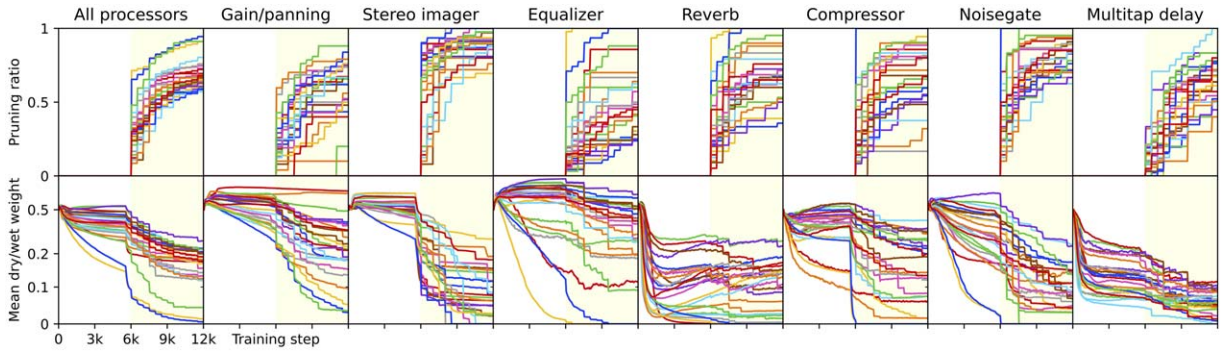


Fig. 3. Process of iterative pruning (hybrid, $\tau = 0.01$). Twenty-four random-sampled songs are shown; each color represents an individual song. The upper and lower rows show the pruning ratios and mean dry/wet weights. The yellow-shaded regions show the pruning phase.

in loss, this approach is justified due to strict resource constraints. During the initial 4,000 steps of the pruning phase, the sparsity coefficient α_p is increased linearly from 0 to 10^{-4}.

6 RESULTS

6.1 Sparsity and Quality Degradation

Same as the mixing console experiments, results for `MedleyDB` and `MixingSecrets` data sets are reported separately (Tables 4 and 5, respectively). In the default setting, the average audio loss L_a is 0.301 for the `MedleyDB` subset, which is a 0.005 increase compared with the full consoles within the range of the tolerance threshold of $\tau = 0.01$. However, for the `MixingSecrets` subset, the audio loss increase is 0.018, higher than the tolerance. This outcome is expected due to the shorter training time for the full console. The average pruning ratio, denoted as μ, is 0.606 and 0.622, respectively, with the equalizer `e` and stereo imager `s` being the most and least retained types, respectively (e.g., 0.805 and 0.483 for `MedleyDB`, respectively). Similar to the mixing console experiments, metrics other than FAD have noisy results, although the changes in values are very small (e.g., at most 0.51dB for SI-SDR).

6.2 Choice of Tolerance

Even with a very low tolerance of $\tau = 0.001$, the number of processors (average pruning ratio of $\mu = 0.442$ and 0.433 for the respective subsets) can be nearly halved. However, setting τ too high, such as $\tau = 0.1$, results in highly sparse graphs but significantly degrades the match quality. The default $\tau = 0.01$ has a good balance, maintaining the match and achieving reasonable sparsity. This can be verified qualitatively with the spectrogram error plots; see bottom three rows of Fig. 10 (and the latter subjective listening test results, e.g., Figs. 6 and 7). There is no noticeable degradation in performance from the full consoles to the pruned graphs with $\tau = 0.001$ and $\tau = 0.01$. Also, refer to the captions in Fig. 8, which provide the pruning ratios and audio losses for the pruned graphs of the song `RockSteady` (note that the full console has an audio loss of 0.523).

6.3 Pruning Process

Fig. 3 shows how the pruning progresses. During the first 6,000 steps of initial console optimization, the dry/wet weights adapt freely to optimize the audio loss L_a because there is no sparsity regularization L_p. Then, as the pruning phase begins, the weight values start to decrease. Each graph's sparsity increases gradually while its weights adapt over time. Note that this trend is different for different processor types, supporting the per-type candidate selection of the dry/wet and hybrid methods.

6.4 Sampling Method Comparison

The three proposed candidate sampling variants under the same tolerance value $\tau = 0.01$ are compared. The brute-force method, having the highest granularity, also achieves the highest sparsity by removing 62.6 and 65.4% of the processors on the `MedleyDB` and `MixingSecrets` data sets, respectively. As fewer processors remain, it also has the worst match quality: the MRSTFT loss L_a increases by 0.009 and 0.021 from the mixing console, respectively. The opposite results are obtained with the dry/wet method. It prunes less and degrades the quality less. Similar results can be observed with most other validation metrics, whereas results with MIR feature distances are more mixed.

To investigate the differences in sparsity, the authors examine the relationship between each dry/wet weight w_i and the loss increase caused by pruning the corresponding processor v_i defined as follows:

$$\Delta_i = L_a(G \setminus \{v_i\}) - L_a(G). \tag{32}$$

Fig. 4 shows scatterplots for two random-sampled songs. Each point (w_i, Δ_i) represents a processor after initial console training. Ideally, maximizing sparsity with the dry/wet method would require a monotonic relationship between w_i and Δ_i. However, this is not observed, and only a weak positive correlation exists. Also, different types are clustered in different regions, again justifying the per-type candidate selection. Still, the weakness of the dry/wet method cannot be completely removed, leading the authors to the hybrid strategy as a compromise.

The three methods differ not only in sparsity but also in the trade-offs between sparsity and match performance.

Table 4. Pruning results with different strategies and tolerance τ for data set `MedleyDB`.

			Pruning Ratio ↑										MIR Feature Distance ↓				
	τ	L_a ↓	μ	g	s	e	r	c	n	d	FAD ↓	SI-SDR ↑	d_{RMS}	d_{CR}	d_{SW}	d_{SI}	d_{BS}
Mixing console	···	0.296	···	···	···	···	···	···	···	···	0.222	−2.66	−6.17	0.093	−4.34	−3.78	−1.46
Brute-force	0.01	0.305	0.626	0.343	0.811	0.540	0.772	0.669	0.711	0.532	0.262	−3.41	−5.94	0.147	−4.14	−3.54	−1.51
Dry/wet	0.01	0.302	0.561	0.315	0.791	0.416	0.736	0.568	0.669	0.433	0.229	−1.87	−6.02	0.066	−4.23	−3.62	−1.55
Hybrid	0.001	0.295	0.442	0.216	0.728	0.259	0.612	0.503	0.540	0.237	0.227	−2.18	−6.11	0.108	−4.36	−3.77	−1.51
	0.01	0.301	0.606	0.322	0.805	0.483	0.741	0.666	0.708	0.519	0.247	−3.03	−5.97	0.096	−4.15	−3.58	−1.52
	0.1	0.374	0.829	0.586	0.926	0.828	0.918	0.804	0.840	0.898	0.406	−4.38	−5.47	0.335	−3.78	−3.07	−1.57

Table 5. Pruning results with different strategies and tolerance τ for data set `MixingSecrets`.

			Pruning Ratio ↑										MIR Feature Distance ↓				
	τ	L_a ↓	μ	g	s	e	r	c	n	d	FAD ↓	SI-SDR ↑	d_{RMS}	d_{CR}	d_{SW}	d_{SI}	d_{BS}
Mixing console	···	0.545	···	···	···	···	···	···	···	···	0.506	−6.85	−4.98	0.985	−2.84	−2.66	−1.13
Brute-force	0.01	0.566	0.654	0.498	0.823	0.429	0.710	0.605	0.766	0.748	0.548	−7.13	−5.06	0.991	−2.83	−2.54	−1.19
Dry/wet	0.01	0.561	0.566	0.472	0.818	0.222	0.623	0.539	0.744	0.546	0.513	−6.79	−4.96	1.02	−2.68	−2.48	−1.13
Hybrid	0.001	0.550	0.433	0.345	0.765	0.134	0.421	0.429	0.552	0.383	0.523	−6.66	−4.94	1.00	−2.68	−2.58	−1.11
	0.01	0.563	0.622	0.490	0.848	0.344	0.637	0.599	0.789	0.646	0.519	−6.35	−5.05	0.995	−2.78	−2.61	−1.22
	0.1	0.647	0.843	0.703	0.914	0.751	0.864	0.831	0.927	0.909	0.805	−7.98	−4.82	1.03	−2.78	−2.33	−1.19

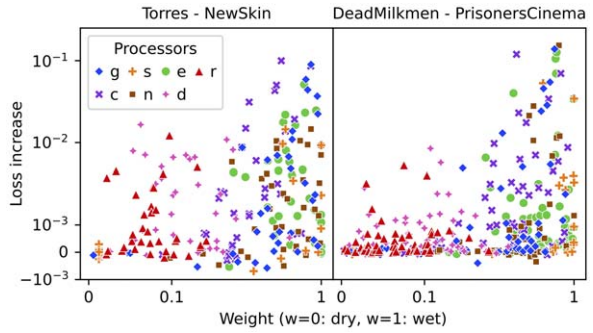


Fig. 4. Each node's weight and loss increase when pruned.

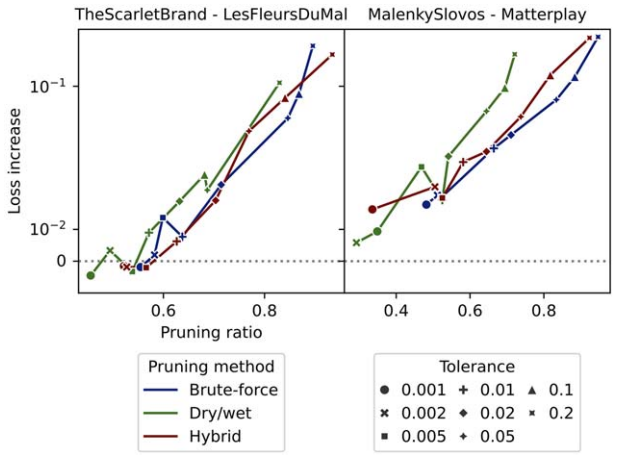


Fig. 5. Loss increases from the mixing console and pruning ratios for different pruning methods and tolerances.

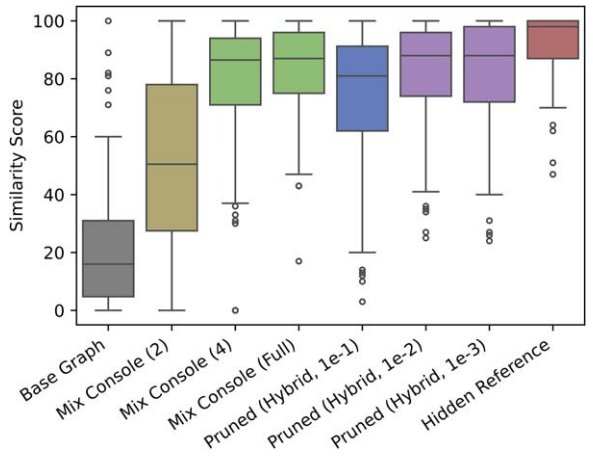


Fig. 6. Subjective listening test results. Each dot corresponds to stimuli that a participant evaluated.

One can observe this by running the methods with more fine-grained tolerance values, from 0.001 to 0.2, and analyzing the loss-sparsity relationship. Fig. 5 shows the result. The brute-force method tends to find graphs with better matches, even with the same graph size (lower right region in each subplot). However, in terms of the search time, the brute-force, dry/wet, and hybrid methods took an average of 56, 29, and 36 min for each song, respectively, using

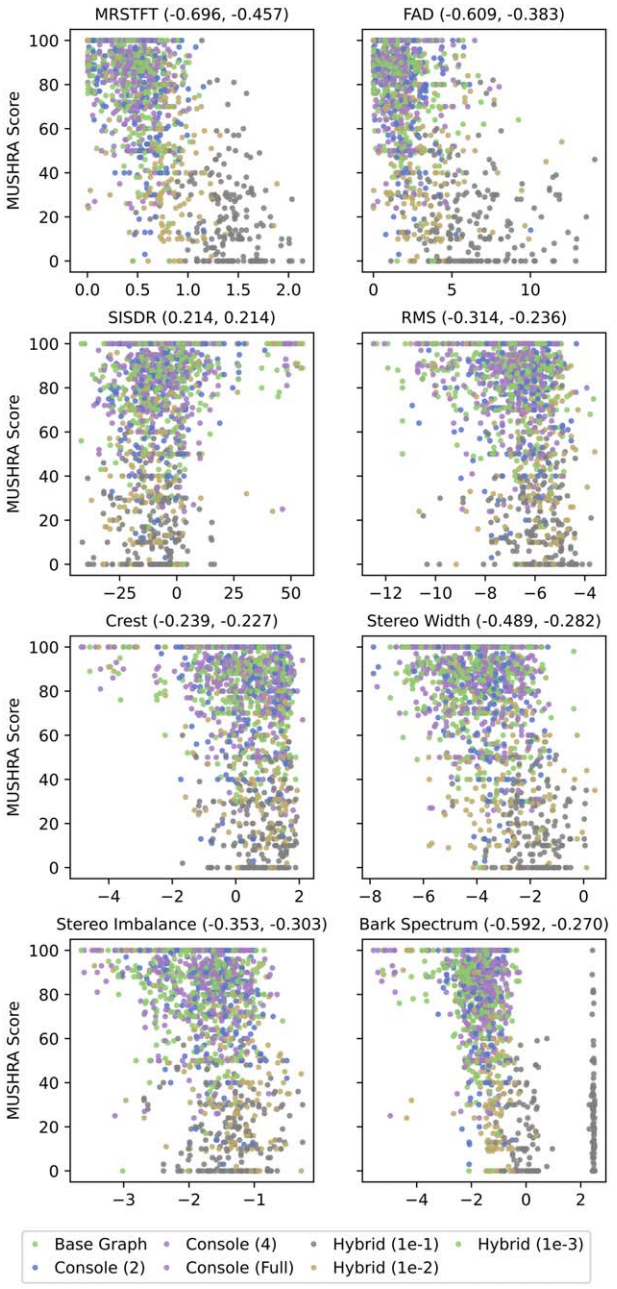


Fig. 7. Scatter plots of MUSHRA scores and objective metrics.

a single RTX 3090 GPU. Note that this difference comes from the different number of candidate trials. Furthermore, it remains unclear whether this difference in the trade-off is perceptually significant, raising the question of whether the finer search is worth the cost.

6.5 Subjective Listening Test

To further investigate the performance of these methods, a Multiple Stimuli with Hidden Reference and Anchor (MUSHRA) test [32] was conducted. Participants were asked to compare the original mix (reference) and the others with scores ranging from 0 to 100. A total of eight stimuli were evaluated, including 1) the hidden reference, 2) naive summation as a low anchor, and the mixing consoles with 3) two and 4) four processor types (as in Table

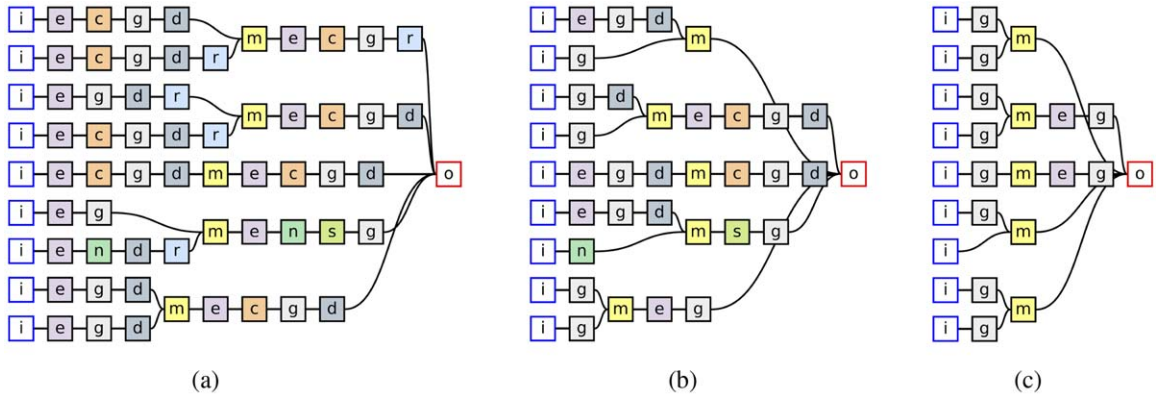


Fig. 8. Pruning results (hybrid method) with various tolerances. Song: Rock Steady by Tabla Breakbeat Science. (a) $\tau = 0.001$; $\rho = 0.45$, $L_a = 0.525$. (b) $\tau = 0.01$; $\rho = 0.74$, $L_a = 0.525$. (c) $\tau = 0.1$; $\rho = 0.85$, $L_a = 0.611$. c = compressor; d = multitap delay; e = equalizer; g = gain/panning; i = input; m = mix; n = noisegate; o = output; r = reverb; s = stereo imager.

2) as mid anchors. The remaining four are the 5) full mixing console and its pruned versions with the hybrid method and the tolerance τ of 6) 0.1, 7) 0.01, and 8) 0.001. Eighteen subjects participated in the test: nine were professional mixing engineers and six were amateurs (4.5 years of experience on average). The remaining three had no experience in music mixing, but they are researchers in musical signal processing and familiar with the MUSHRA-style test. Each participant went through two training and 12 main sessions. Each session was a random 8-s excerpt from the validation set. The authors used WebMUSHRA [33] to conduct the test remotely.

Fig. 6 summarizes the results. The hidden reference scored 92.5 ± 1.3 (95% confidence interval). As expected, the base graph received the lowest score of 21.1 ± 2.7. Note that some high-score outliers exist; this is because their corresponding excerpts were "easy," where the naive summations sufficiently match the targets. The mixing console with two types (gain/panning and stereo imager) scored 51.2 ± 4.0, indicating that matching the overall loudness and stereo balance is insufficient. By adding the equalizer and reverb, the score improved significantly to 80.5 ± 2.6. The full mixing console scored 83.6 ± 2.1, which is the best score among tested stimuli but still significantly lower than the hidden reference (8.9 points). The pruned graphs with the tolerance τ of 0.01 and 0.001 received almost identical scores of 82.8 ± 2.3 and 82.5 ± 2.3, respectively. Also, their score decrease from the mixing console was very small (1.1 maximum). However, setting the tolerance to 0.1 resulted in a clear degradation and scored 74.9 ± 3.0. These results align well with the previous analysis: the pruning has an almost negligible effect on the perceptual similarity if the tolerance value τ is set appropriately.

It could be helpful to investigate how well the loss function and evaluation metrics correlate with subjective similarity. Fig. 7 shows scatter plots between the MUSHRA score and the loss/metrics. For each pair, two Pearson correlation coefficients were calculated: one with the full test results and the other one that discarded the base graph samples. The latter was motivated by the fact that the base graph

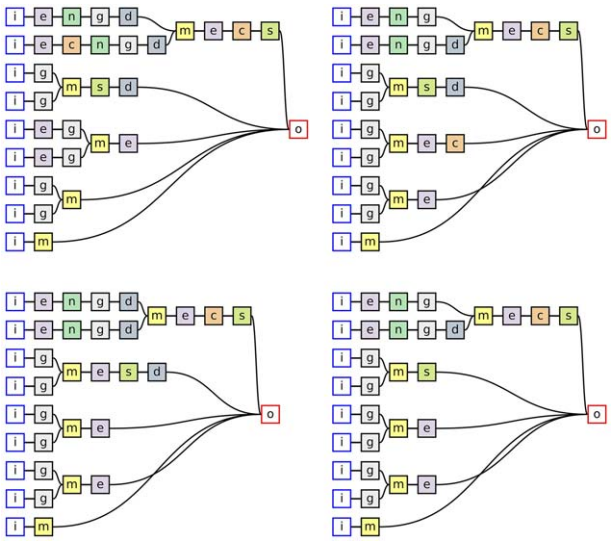


Fig. 9. Graphs obtained with multiple pruning runs (default setting). Song: GirlsOnABridge by EthanHein. c = compressor; d = multitap delay; e = equalizer; g = gain/panning; i = input; m = mix; n = noisegate; o = output; s = stereo imager.

results are, in many cases, too different from the target and not of interest. These Pearson correlation coefficient values are also reported in Fig. 7 (see the parentheses). The results show that the MRSTFT loss shows the strongest linear correlation, making it a suitable objective for optimization and pruning. The second best is the FAD score, possibly due to its ability to capture diverse features of the music mixes, even if it was not pretrained for that purpose. The MIR feature distances have slightly weaker correlations, which are expected because each feature reflects certain specific aspects of the mix. The SI-SDR has the weakest correlation, possibly because of the aforementioned misalignment and sample-level match not being perceptually important.

After the test, the subjects were asked what kind of differences they could find between the reference and the other stimuli, especially the full consoles and the pruning with

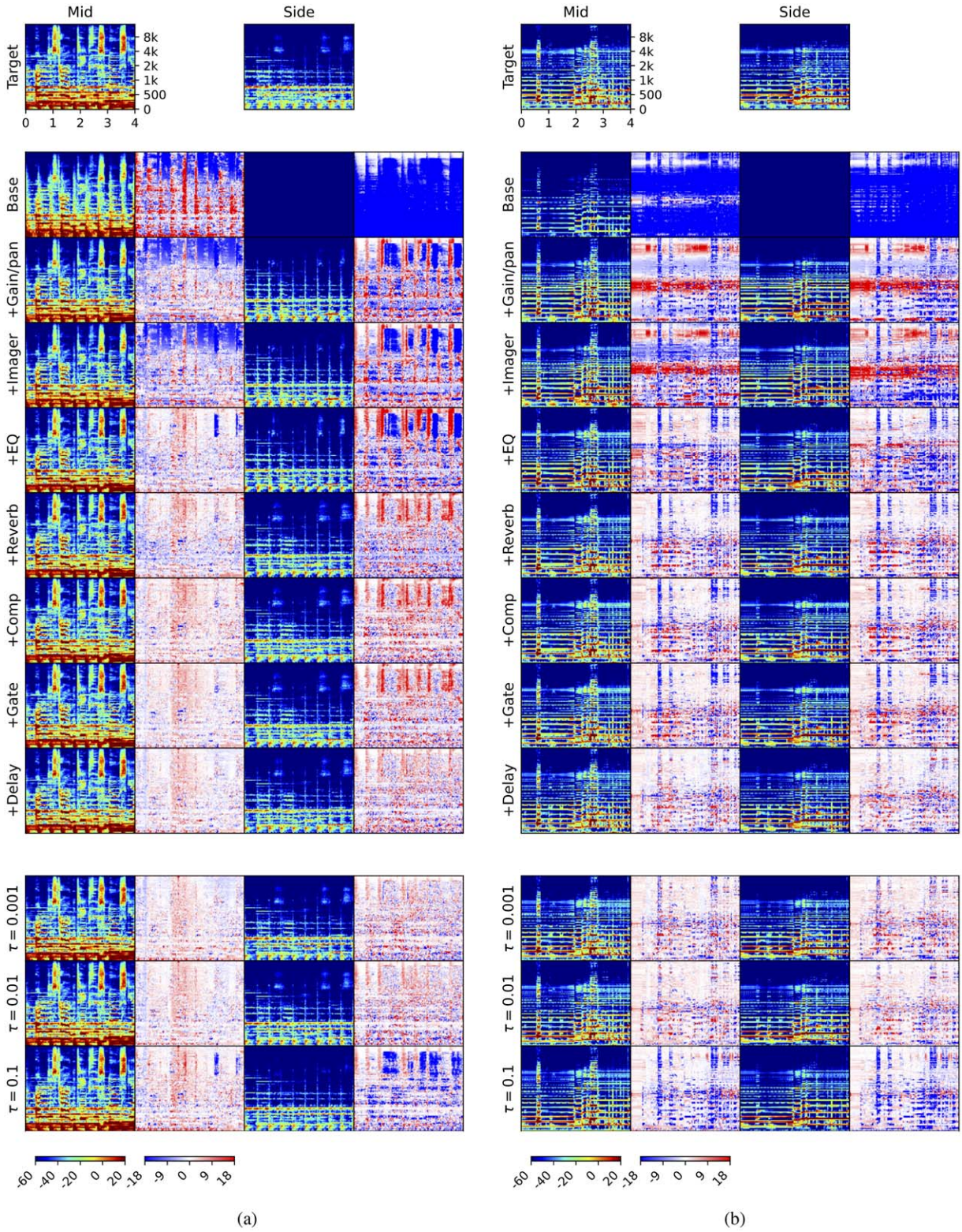


Fig. 10. Log-magnitude spectrograms of target mixes and matches of mixing console with different processor configurations and pruned graphs. Refer to the supplementary page for more samples. (a) New Skin by Torres; (b) Rock Steady by Tabla Breakbeat Science.

$\tau \leq 0.01$. One common response was the slight mismatch in the loudness dynamics and spatial effects. This applied to both the full consoles and their pruned versions, indicating that the processors or optimization objectives could be further improved. Also, the participants were able to catch some smearing artifacts, which might have been caused by the reverbs and multitap delays while trying to correct the misalignments. Finally, an informal AB test was also conducted to compare the three candidate selection methods with the same tolerance $\tau = 0.01$. However, no statistically

Table 6. Pruning results of the full data set (default setting).

	Pruning Ratio ↑								FAD ↓	SI-SDR ↑	MIR Feature Distance ↓				
	L_a ↓	μ	g	s	e	r	c	n	d			d_{RMS}	d_{CR}	d_{SW}	d_{SI}	d_{BS}
MedleyDB	0.431	0.631	0.365	0.842	0.444	0.688	0.735	0.773	0.570	0.488	−4.11	−5.75	−.237	−3.67	−3.20	−1.32
MixingSecrets	0.625	0.637	0.498	0.876	0.334	0.635	0.633	0.798	0.685	0.621	−12.2	−5.03	.254	−2.83	−2.82	−1.01
Internal	0.434	0.755	0.705	0.871	0.550	0.728	0.854	0.857	0.723	0.280	0.28	−5.88	−.387	−3.72	−3.45	−1.47
Total	0.506	0.692	0.574	0.868	0.451	0.686	0.751	0.821	0.685	0.443	−5.15	−5.54	−.120	−3.37	−3.17	−1.27

significant results were found. This is possible because their differences are smaller than, e.g., the differences between the hybrid method with $\tau = 0.01$ and $\tau = 0.001$.

6.6 Characteristics of Iterative Pruning

Here, example cases are provided to illustrate the behavior of the pruning method. First, one can qualitatively check that the pruning indeed removes the unnecessary processors. Recall that adding reverbs to the song `RockSteady` did not significantly improve performance [Fig. 10(b)]. It is reasonable to anticipate that those reverbs will be pruned with a moderate tolerance τ accordingly. Fig. 8 shows that this is the case; only five reverbs remain among 14 when $\tau = 0.001$ and 0 for $\tau = 0.01$. When $\tau = 0.1$, only the gain/pannings and equalizers are left, and the others for the details are eliminated, significantly degrading the match.

It should be noted that the greedy pruning tends to yield suboptimal results. For example, it could fail to detect redundant processors. In Fig. 8(b), the bottom two inputs are processed with three gain/pannings; at least one of them can be removed by modifying the parameters of the remaining ones. Although this specific case is easy to detect and correct, it hints that sparser graphs might exist that are harder to find.

Another notable behavior is that each pruning run of the same song results in a slightly different graph. The pruning of GirlOnABridge was run ten times; four of them are shown in Fig. 9. The resulting graphs have the number of processors ranging from 19 to 22, with an audio loss ranging from 0.668 to 0.674. The main cause of this variance is the stochasticity of the iterative pruning; candidates sampled early are more likely to satisfy the tolerance condition and be pruned, whereas the remaining will adapt and be further optimized, changing their "importance scores."

6.7 Consistency Analysis

It is worth investigating how similar the pruned graph G is to the ground truth. However, the latter is unavailable. As a remedy, one can try a "consistency check," where the pruned result is treated as a synthetic ground truth. Then, the pruning can be rerun with the matched mix $\hat{y}$ as a target and similarity of the obtained graphs $\tilde{G}$ can be measured with the original G. Because of the restrictions on the graph structure (sequential chain, fixed order, at most one processor instance per type), comparing the two graphs, G and $\tilde{G}$, is equivalent to comparing the set of processors for each input and subgroup (i.e., multiple binary classification tasks).

Table 7 summarizes the evaluation results; it is reported that the per-type and microaverage metrics (denoted as μ). The average accuracy is 0.902 (i.e., the pruning correctly predicts about nine out of ten whether a certain source track or subgroup mix is processed with a given processor type (or not). In terms of the F1 score, the microaverage result is 0.872. The multitap delay `d` shows the lowest result (F1 score of 0.722), possibly due to its subtle effect. Note that the precision is consistently lower than recall across all types except for the equalizer `e` (0.054 by average). This

Table 7. Processor detection performance of the pruning method (default setting) with consistency check.

	μ	g	s	e	r	c	n	d
Accuracy ↑	0.902	0.888	0.968	0.923	0.956	0.914	0.909	0.757
Precision ↑	0.846	0.848	0.871	0.954	0.931	0.848	0.773	0.682
Recall ↑	0.900	0.955	0.903	0.917	0.931	0.935	0.874	0.768
F1 Score ↑	0.872	0.898	0.887	0.935	0.931	0.889	0.820	0.722

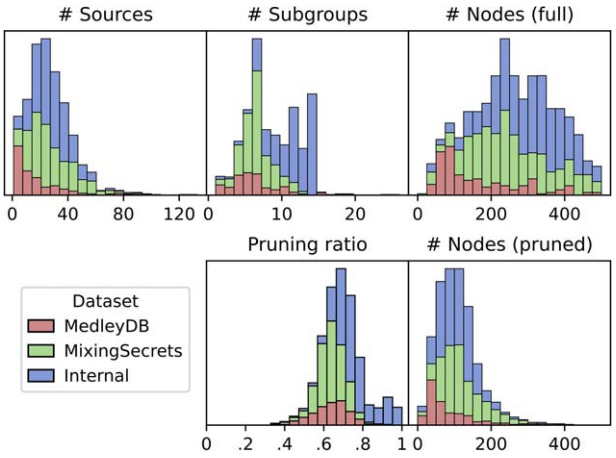


Fig. 11. Statistics of the consoles and pruned graphs (full data). Per–data set results are stacked to form the full histograms.

is not so surprising; as the pruning operates with a certain tolerance, it is biased toward false negatives rather than false positives.

It should be noted that the mean audio loss result L_a of this experiment was 0.112, which is much lower than the main result (0.422). This is because each target mix $\hat{y}$ is already projected to the subspace that the graph can express (shown as a teal-shaded area in Fig. 1). Nevertheless, the loss value is not negligible. Two reasons are suspected: the pruned graph $\tilde{G}$ is slightly different from the original G, and the gradient descent struggles to find better parameters.

6.8 Full Results

Finally, the authors pruned the full data set ensemble and collected the graph data for the downstream tasks. The evaluation results are reported in Table 6, and they are consistent with the evaluation subset results. Overall statistics of the songs, original mixing consoles, and their corresponding pruned graphs are shown in Fig. 11. Among the data sets, `MedleyDB` has the fewest source tracks, averaging 17.6 per song. In contrast, the `internal` data set has the most (28.8), closely followed by `MixingSecrets` (27.9). The `internal` data set also tends to have more subgroups, leading to larger mixing consoles. It is suspected that these contributed to the higher sparsity of its pruned graphs; more processors were initially used to match the mixes, resulting in more redundant ones that could be pruned. On average, 72.1 processors (108.5 nodes) remained per song after pruning. Considering that each full mixing console initially contained an average of 247.6 processors (280.1 nodes), this corresponds to a pruning ratio of 0.692.

7 CONCLUSION

The reverse engineering of the music mix was formulated as a search of the audio processing graph and its processor parameters. Then, the search was restricted to the iterative pruning of a predefined mixing console and a computationally feasible algorithm was derived. Extensive evaluation results indicate that, with an appropriate tolerance, pruning rarely degrades the match quality. This was supported by the almost identical MUSHRA score of the default pruning setup to that of the full mixing console.

On the other hand, there was a clear score difference between the hidden reference and the matches, including ones from the full mixing console. This indicates that improving the console can have a significant impact on the resulting match quality. The processors were chosen, such as the zero-phase FIR equalizer and STFT mask-based reverb, due to their simplicity and fast computation on GPU. However, alternatives, such as parametric equalizers [42] and artificial reverberation [43], exist. Also, when the dynamic processors were added to the mixing consoles, the listening test results showed only a slight improvement. Approximated ballistics could be a potential cause, as recently reported [56]. The processor set could be expanded to include effects such as saturation [40] or modulation [41]. Additionally, the method did not account for time-varying parameters (e.g., automation), leading to audible mismatches, such as the inability to replicate fade-outs.

The structure of the mixing consoles could also be modified to better reflect real-world practices. Send-return loops could be added, as could postequalizers for compressors, and even multi-input or multioutput processors, such as those with sidechains or crossover filters. Although already available subgroup information was used to construct a mixing console, it is possible to estimate it with audio feature analysis [57]. One could also jointly optimize the subgrouping information. However, current framework should be extended (e.g., allowing learnable edge weights) to accommodate such graph connectivity.

Another critical component of the method is the optimization objective. The analysis with the subjective listening test results showed that the current MRSTFT loss correlated the most with the MUSHRA score among all tested metrics. However, it also has been reported to miss certain perceptual features [19, 58]. Exploring alternative objectives [59] could address these shortcomings.

The resulting graph's sparsity and the search cost were also of interest due to the application of graph data collection. In this context, three pruning variants with different granularities in search were explored. The real-world results show that a more detailed search produces more accurate

graphs of the match-sparsity tradeoff. However, it is not clear whether this difference is important enough to justify the extra cost, given the limited computation budget. It is noted that the final experiment, which consisted of more than a thousand individual optimization runs, took about three days, even with 10 RTX 3090 GPUs.

Although this method finds a sparse and interpretable mixing graph, it is still unclear how closely the pruning method finds the actual underlying graph. The authors only have the following conservative conclusions. This method, at least, tries to find a minimal graph that closely follows the target. Also, it shows reasonable reconstruction accuracy (0.872 F1 score) for the in-domain graphs (obtained with the same pruning method), as shown in the consistency check experiment. Additionally, the graph's plausibility is supported by the design of mixing consoles, their processors, and several qualitative case studies.

It is possible to extend the current pruning method in various way. First, differentiability is not required for the pruning objective, allowing broader design choices, such as the MIR feature distances that were used for the evaluation. Secondly, this method tends to prune processors for short-span tracks because of their limited contribution. However, their removal can sometimes be noticeable, and methods for resolving this are desirable. Third, adopting advanced neural network pruning techniques [44] could improve the candidate node selections. Finally, domain-specific methods, (e.g., merging LTI processors into a single one) can be applied, which could further increase sparsity.

Other than the aforementioned extensions, a radically different approach could be taken, (e.g., adopting the neural architecture search by relaxing the prior restrictions on graph structures [23, 24] and RL [46, 47]. Although these are promising, the main challenge will be balancing various aspects, such as the graph's plausibility, match quality, and sparsity, and the search method's computation cost.

This work focused on the graph search algorithm and its use for collecting the graph data. The next step will be applying this data to downstream applications. Prior works on statistical analysis of music mixes [7–9] can be extended by directly assessing the mixing graphs, not just the audio features. The collected graphs can also be treated as pseudolabels and neural networks can be trained to estimate the graphs (and their corresponding parameters). Exiting neural network-based systems can be modified (e.g., automatic mixing [11, 12, 60, 61, 13, 62] and style transfer [63]) to output the graphs, which allow the end users to interpret and control the outputs further for their needs. Although graphs were collected from an ensemble of multiple data sets, the amount of data could still be insufficient for the neural network training. Mitigating potential overfitting and improving the generalization performance are left as future work.

8 REFERENCES

[1] P. D. Pestana and J. D. Reiss, "Intelligent Audio Production Strategies Informed by Best Practices," in *Proceedings of the AES 53rd International Conference on Semantic Audio* (2014 Jan.), paper S2-2.

[2] F. Everardo, "Towards an Automated Multitrack Mixing Tool Using Answer Set Programming," in *Proceedings of 14th Sound and Music Computing Conference* (Espoo, Finland) (2017 Jul.).

[3] D. Ronan, H. Gunes, and J. D. Reiss, "Analysis of the Subgrouping Practices of Professional Mix Engineers," presented at the *142nd Convention of the Audio Engineering Society Convention* (2017 May), paper 9700.

[4] R. M. Bittner, J. Salamon, M. Tierney, M. Mauch, C. Cannam, and J. P. Bello, "MedleyDB: A Multitrack Dataset for Annotation-Intensive MIR Research." in *Proceedings of the 15th International Society for Music Information Retrieval (ISMIR) Conference*, vol. 14, pp. 155–160 (2014 Oct.).

[5] R. M. Bittner, J. Wilkins, H. Yip, and J. P. Bello, "MedleyDB 2.0: New Data and a System for Sustainable Data Collection," in *Proceedings of the 24th International Society for Music Information Retrieval (ISMIR)*, p. 36 (New York, NY) (2016 Aug.)

[6] M. Senior, *Mixing Secrets for the Small Studio* (Routledge, Oxfordshire, UK, 2018), 2nd ed. https://doi.org/10.4324/9781315150017.

[7] A. Wilson and B. Fazenda, "101 Mixes: A Statistical Analysis of Mix-Variation in a Dataset of Multi-Track Music Mixes," presented at the *139th Convention of the Audio Engineering Society* (2015 Oct.–Nov.), paper 9398.

[8] A. Mourgela, E. Quinton, S. Bissas, J. D. Reiss, and D. Ronan, "Exploring Trends in Audio Mixes and Masters: Insights From a Dataset Analysis," *arXiv preprint arXiv:2412.03373* (2024).

[9] A. Wilson and B. Fazenda, "Variation in Multitrack Mixes: Analysis of Low-Level Audio Signal Features," *J. Audio Eng. Soc.*, vol. 64, no. 7/8, pp. 466–473 (2016 Jul./Aug.). https://doi.org/10.17743/jaes.2016.0029.

[10] B. De Man, J. Reiss, and R. Stables, "Ten Years of Automatic Mixing," in *Proceedings of 3rd Workshop on Intelligent Music Production* (Salford, UK) (2017 Sep.).

[11] E. Perez-Gonzalez and J. Reiss, "Automatic Gain and Fader Control for Live Mixing," in *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pp. 1–4 (New Palz, NY) (2009 Oct). https://doi.org/10.1109/aspaa.2009.5346498.

[12] B. De Man and J. D. Reiss, "A Knowledge-Engineered Autonomous Mixing System," presented at the *135th Convention of the Audio Engineering Society* (2013 Oct.), paper 8961.

[13] C. J. Steinmetz, J. Pons, S. Pascual, and J. Serrà, "Automatic Multitrack Mixing With A Differentiable Mixing Console Of Neural Audio Effects," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 71–75 (Toronto, Canada) (2021 Jun.). https://doi.org/10.1109/icassp39728.2021.9414364.

[14] S. Lee, J. Park, S. Paik, and K. Lee, "Blind Estimation of Audio Processing Graph," in *Pro-*

ceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), pp. 1–5 (Rhodes Island, Greece) (2023 Jun.). https://doi.org/10.1109/icassp49357.2023.10096581.

[15] D. Barchiesi and J. Reiss, "Reverse Engineering of a Mix," *J. Audio Eng. Soc.*, vol. 58, no. 7/8, pp. 563–576 (2010).

[16] J. T. Colonel and J. Reiss, "Reverse Engineering of a Recording Mix With Differentiable Digital Signal Processing," *J. Acoust. Soc. Am.*, vol. 150, no. 1, pp. 608–619 (2021 Jul./Aug.). https://doi.org/10.1121/10.0005622.

[17] J. T. Colonel and J. Reiss, "Reverse Engineering a Nonlinear Mix of a Multitrack Recording," *J. Audio Eng. Soc.*, vol. 71, no. 9, pp. 586–595 (2023 Sep.). https://doi.org/10.17743/jaes.2022.0105.

[18] J. Engel, L. H. Hantrakul, C. Gu, and A. Roberts, "DDSP: Differentiable Digital Signal Processing," in *Proceedings of the International Conference on Learning Representations (ICLR)* (Virtual) (2020 Apr.)

[19] B. Hayes, J. Shier, G. Fazekas, A. McPherson, and C. Saitis, "A Review of Differentiable Digital Signal Processing for Music & Speech Synthesis," *Front. Signal Process.*, vol. 3, p. 1284100 (2024 Jan.). https://doi.org/10.3389/frsip.2023.1284100.

[20] "The mixing console — Split, Inline and Hybrids," https://steemit.com/sound/@jamesub/the-mixing-console-split-inline-and-hybrids (accessed Feb. 26, 2024).

[21] A. Paszke, S. Gross, F. Massa, et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)* (Vancouver, Canada) (2019 Dec).

[22] G. Castellano, A. M. Fanelli, and M. Pelillo, "An Iterative Pruning Algorithm for Feedforward Neural Networks," *IEEE Trans. Neural Networks*, vol. 8, no. 3, pp. 519–531 (1997 May). https://doi.org/10.1109/72.572092.

[23] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable Architecture Search," in *Proceedings of the International Conference on Learning Representations (ICLR)s of the International Conference on Learning Representations (ICLR)* (New Orleans, LA) (2019 May).

[24] Z. Ye, W. Xue, X. Tan, Q. Liu, and Y. Guo, "NAS-FM: Neural Architecture Search for Tunable and Interpretable Sound Synthesis Based on Frequency Modulation," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, p. 5869–5877 (Macao, SAR) (2023 Aug.). https://doi.org/10.24963/ijcai.2023/651.

[25] N. Uzrad, O. Barkan, A. Elharar, et al., "DiffMoog: A Differentiable Modular Synthesizer for Sound Matching," *arXiv preprint arXiv:2401.12570* (2024).

[26] S. Lee, M. Martínez-Ramírez, W.-H. Liao, et al., "Searching for Music Mixing Graphs: A Pruning Approach," in *Proceedings of the 27th International Conference on Digital Audio Effects (DAFx)*, pp. 147154 (Guildford, UK) (2024 Sep.)

[27] S. Lee, M. Martínez-Ramírez, W.-H. Liao, et al., "GRAFX: An Open-Source Library for Audio Processing Graphs in PyTorch," in *Proceedings of the 27th International Conference on Digital Audio Effects (DAFx)*, pp. 475–478 (Guildford, UK) (2024 Sep.).

[28] K. Kilgour, M. Zuluaga, D. Roblek, and M. Sharifi, "Frèchet Audio Distance: A Metric for Evaluating Music Enhancement Algorithms," *arXiv preprint arXiv:1812.08466* (2018).

[29] J. Le Roux, S. Wisdom, H. Erdogan, and J. R. Hershey, "SDR–Half-Baked or Well Done?" in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 626–630 (Brighton, UK) (2019 May). https://doi.org/10.1109/icassp.2019.8683855.

[30] S. Vanka, C. Steinmetz, J.-B. Rolland, J. Reiss, and G. Fazekas, "Diff-MST: Differentiable Mixing Style Transfer," in *Proceedings of the 25th International Society for Music Information Retrieval Conference (ISMIR)* (San Francisco, CA) (2024 Nov.).

[31] B. Man, B. Leonard, R. King, and J. D. Reiss, "An Analysis and Evaluation of Audio Features for Multitrack Music Mixtures," in *Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR)* , pp. 137–142 (Taipei, Taiwan) (2014 Oct.).

[32] ITU-R, "Method for the Subjective Assessment of Intermediate Quality Level of Audio Systems," Recommendation ITU-R BS.1534-3 (2015 Oct.).

[33] M. Schoeffler, S. Bartoschek, F.-R. Stöter, et al. "webMUSHRA — A Comprehensive Framework for Web-based Listening Tests," *J. Open Res. Softw.*, vol. 6, p. 8 (2018 Feb.). https://doi.org/10.5334/jors.187.

[34] F. Caspe, A. McPherson, and M. Sandler, "DDX7: Differentiable FM Synthesis of Musical Instrument Sounds," in *Proceedings of the 23rd International Society for Music Information Retrieval Conference (ISMIR)*, pp. 608–616 (Bengaluru, India) (2022 Dec.).

[35] C. Mitcheltree and H. Koike, "SerumRNN: Step by Step Audio VST Effect Programming," in *Proceedings of the 10th International Conference on Artificial Intelligence in Music, Sound, Art and Design (EvoMUSART)*, pp. 218–234 (2021 Apr.). https://doi.org/10.1007/978-3-030-72914-1_15.

[36] J. Guo and B. McFee, "Automatic Recognition of Cascaded Guitar Effects," in *Proceedings of the 26th International Conference on Digital Audio Effects (DAFx)* (Copenhagen, Denmark) (2023 Sep.).

[37] O. Take, K. Watanabe, T. Nakatsuka, et al., "Audio Effect Chain Estimation and Dry Signal Recovery from Multi-Effect-Processed Musical Signals," in *Proceedings of the 27th International Conference on Digital Audio Effects (DAFx)* (Guildford, UK) (2024 Sep.).

[38] M. A. Martínez-Ramírez, O. Wang, P. Smaragdis, and N. J. Bryan, "Differentiable Signal Processing With Black-Box Audio Effects," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)* (Toronto, Canada) (2021 Jun.). https://doi.org/10.1109/icassp39728.2021.9415103.

[39] C. J. Steinmetz, N. J. Bryan, and J. D. Reiss, "Style Transfer of Audio Effects with Differentiable Signal Processing," *J. Audio Eng. Soc.*, vol. 70, no. 9, pp. 708–721 (2022 Sep.). https://doi.org/10.17743/jaes.2022.0025.

[40] J. T. Colonel, M. Comunità, and J. Reiss, "Reverse Engineering Memoryless Distortion Effects with Differentiable Waveshapers," presented at the *153rd Convention of the Audio Engineering Society* (2022 Oct.), paper 10626.

[41] A. Carson, S. King, C. V. Botinhao, and S. Bilbao, "Differentiable Grey-Box Modelling of Phaser Effects Using Frame-Based Spectral Processing," in *Proceedings of the 26th International Conference on Digital Audio Effects (DAFx)* (Copenhagen, Denmark) (2023 Sep.).

[42] S. Nercessian, "Neural Parametric Equalizer Matching Using Differentiable Biquads," in *Proceedings of the 23rd International Conference on Digital Audio Effects (DAFx)*, pp. 265–272 (Vienna, Austria) (2020 Sep.).

[43] S. Lee, H.-S. Choi, and K. Lee, "Differentiable Artificial Reverberation," *IEEE/ACM Trans. Audio Speech Lang. Process.*, vol. 30, pp. 2541–2556 (2022 Jul.). https://doi.org/10.1109/taslp.2022.3193298.

[44] Y. He and L. Xiao, "Structured Pruning for Deep Convolutional Neural Networks: A Survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 5, p. 2900–2919 (2024 May). https://doi.org/10.1109/tpami.2023.3334614.

[45] T. Elsken, J. H. Metzen, and F. Hutter, "Neural Architecture Search: A Survey," *J. Mach. Learn. Res.*, vol. 20, no. 55, pp. 1–21 (2019 Mar.).

[46] D. C. Elton, Z. Boukouvalas, M. D. Fuge, and P. W. Chung, "Deep Learning for Molecular Design—A Review of the State of the Art," *Mol. Syst. Des. Eng.*, vol. 4, no. 4, pp. 828–849 (2019 Aug.). https://doi.org/10.1039/c9me00039a.

[47] J. You, B. Liu, Z. Ying, V. Pande, and J. Leskovec, "Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation," in *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS)*, p. 31 (Montreal, Canada) (2018 Dec.).

[48] D. Y. Fu, H. Kumbong, E. Nguyen, and C. Ré, "FlashFFTConv: Efficient Convolutions for Long Sequences with Tensor Cores," in *Proceedings of the 11th International Conference on Learning Representations (ICLR)* (Kigali, Rwanda) (2023 May).

[49] D. Giannoulis, M. Massberg, and J. D. Reiss, "Digital Dynamic Range Compressor Design—A Tutorial and Analysis," *J. Audio Eng. Soc.*, vol. 60, no. 6, pp. 399–408 (2012 Jun.).

[50] B. Hayes, C. Saitis, and G. Fazekas, "Sinusoidal Frequency Estimation by Gradient Descent," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 1–5 (Rhoades Island, Greece) (2023 May). https://doi.org/10.1109/icassp49357.2023.10095188.

[51] Y. Bengio, N. Léonard, and A. Courville, "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation," *arXiv preprint arXiv:1308.3432* (2013).

[52] 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 *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 6199–6203 (Barcelona, Spain) (2020 May). https://doi.org/10.1109/icassp40776.2020.9053795.

[53] A. Wright and V. Välimäki, "Perceptual Loss Function for Neural Modeling of Audio Systems," in Proceedings of the *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 251–255 (Barcelona, Spain) (2020 May). https://doi.org/10.1109/icassp40776.2020.9052944.

[54] C. J. Steinmetz and J. D. Reiss, "auraloss: Audio Focused Loss Functions in PyTorch," in *Proceedings of the Digital Music Research Network One-Day Workshop (DMRN+15)Digital Music Research Network One-Day Workshop (DMRN+15)* (London, UK) (2020 Dec.).

[55] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," in *Proceedings of the International Conference on Learning Representations (ICLR)* (New Orleans, LA) (2019 Dec.), paper 939.

[56] C. J. Steinmetz, T. Walther, and J. D. Reiss, "High-Fidelity Noise Reduction with Differentiable Signal Processing," presented at the *155th Convention of the Audio Engineering Society* (2023 Oct.), paper 153.

[57] D. Ronan, H. Gunes, D. Moffat, and J. D. Reiss, "Automatic Subgrouping of Multitrack Audio," in *Proceedings of the 18th International Conference on Digital Audio Effects (DAFx)* (Trondheim, Norway) (2015 Nov–Dec.).

[58] J. Turian and M. Henry, "I'm Sorry for Your Loss: Spectrally-Based Audio Distances Are Bad at Pitch," in *Proceedings of the "I Can't Believe It's Not Better!" Conference on Neural Information Processing Systems (NeurIPS) Workshop* (2020 Dec.).

[59] C. Vahidi, H. Han, C. Wang, M. Lagrange, G. Fazekas, and V. Lostanlen, "Mesostructures: Beyond Spectrogram Loss in Differentiable Time–Frequency Analysis," *J. Audio Eng. Soc.*, vol. 71, no. 9, pp. 577–585 (2023 Sep.). https://doi.org/10.17743/jaes.2022.0103.

[60] 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 *Proceedings of the 23rd International Society for Music Information Retrieval Conference (ISMIR)*, pp 411–418 (Bengaluru, India) (2022 Dec.).

[61] D. Koszewski, T. Görne, G. Korvel, and B. Kostek, "Automatic Music Signal Mixing System Based on One-Dimensional Wave-U-Net Autoencoders," *EURASIP J. Audio Speech Music*, vol. 2023, no. 1, p. 1 (2023 Jan.). https://doi.org/10.1186/s13636-022-00266-3.

[62] C. J. Steinmetz, S. S. Vanka, M. A. Martínez-Ramírez, and G. Bromham, *Deep Learning for Automatic Mixing* (ISMIR, Victoria, Canada, 2022 Dec.). https://dl4am.github.io/tutorial/landing-page.html.

[63] J. Koo, M. A. Martínez-Ramírez, W.-H. Liao, et al., "Music Mixing Style Transfer: A Contrastive Learning Approach to Disentangle Audio Effects," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 1–5 (Rhoades Island, Greece) (2023 Jun.). https://doi.org/10.1109/icassp49357.2023.10096458.

THE AUTHORS


Sungho Lee


Marco A. Martínez-Ramírez


Wei-Hsiang Liao


Stefan Uhlich


Giorgio Fabbro


Kyogu Lee


Yuki Mitsufuji

Sungho Lee is a Ph.D. student in digital content and information studies with Music and Audio Research Group, Seoul National University, Seoul, South Korea.

•

Marco A. Martínez-Ramírez is a research scientist at Sony AI in Tokyo, where he is part of the Music Foundation Model Team. He received his Ph.D. from the Centre for Digital Music at Queen Mary University of London. Marco also has a background in music production and mixing engineering.

•

Wei-Hsiang Liao received his Ph.D. in informatics, telecommunications, and electronics from the University of Pierre and Marie Curie (currently Sorbonne University). He joined Sony Group Corporation in 2017 and now serves as a senior research scientist at Sony AI. His research interests include signal processing, acoustics modeling, and representation learning.

•

Stefan Uhlich received his Dipl.-Ing. and Ph.D. in electrical engineering from the University of Stuttgart in 2006 and 2012, respectively. He was a research assistant at the university's Chair of System Theory and Signal Processing (2007–2011), focusing on statistical signal processing and parameter estimation. Since 2011, he has been with Sony R&D working on music source separation and deep learning.

•

Giorgio Fabbro received his M.Sc. in informatics from the Technische Universität München in 2020. Since then, he has joined Sony R&D as an engineer, working on music source separation and deep learning.

•

Kyogu Lee received a Ph.D. degree in computer-based music theory and acoustics from Stanford University, Stanford, CA, USA. He is currently a professor in the Department of Intelligence and Information, Seoul National University, Seoul, South Korea, and leads the Music and Audio Research Group.

•

Yuki Mitsufuji is a distinguished engineer at Sony Group Corporation and lead research scientist at Sony AI, holding a Ph.D. in information science and technology from the University of Tokyo. Currently, Dr. Mitsufuji serves as the vice president of artificial intelligence (AI) research at Sony AI and head of the Creative AI Laboratory at Sony R&D.