A. RELATED WORKS

A.1. Composition of audio processors

Most audio processors are designed to modify some specific properties of their input signals, e.g., magnitude response, loudness dynamics, and stereo width. As such, combining multiple processors is a common practice to achieve the full effect. Following the main text, we will use the terminology "graph" to represent this composition, although some previous works considered simple structures that allow a more compact form, e.g., a sequence. Now, we outline the previous attempts that tried to estimate the processing graph or its parameters from reference audio. These works differ in task and domain, processors, graph structure, and estimation methods. For example, if the references are dry sources and a wet mixture, this task becomes reverse engineering [1, 2, 3, 4]. In terms of the prediction targets, some fixed the graph and estimated only the parameters [5, 6, 4, 7, 8]. Others tried to predict the graph [3] or both [1, 2, 9, 10]. Table 1 summarizes such differences.

A.2. Differentiable signal processing

Differentiable processor — Exact implementation or approximation of processors in an automatic differentiation framework, e.g., `pytorch` [11], enables parameter optimization via gradient descent. Numerous efforts have been focused on converting existing audio processors to their differentiable versions [4, 12, 13, 14, 15, 16]; refer to the recent review [17] and references therein for more details. In many cases, these processors are combined with neural networks, whose computation is done in GPU. Thus, converting the audio processors to be "GPU-friendly" has been an active research topic. For example, for a linear time-invariant (LTI) system with a recurrent structure, we can sample its frequency response to approximate its infinite impulse response (IIR) instead of directly running the recurrence; the former is faster than the latter [15, 16]. However, it is nontrivial to apply a similar trick to nonlinear, recurrent, or time-varying processors. Typically, further simplifications and approximations are employed, e.g., replacing the nonlinear recurrent part with an IIR filter [12] or assuming frame-wise LTI to a linear time-varying system [14]. Sometimes, we can only access input and output signals. In such a case, one can approximate the gradients with finite difference methods [8, 18] or use a pretrained auxiliary neural network that mimics the processors [5]. In the literature, these are also referred to as "differentiable;" hence, it is rather an umbrella term encompassing all methods that obtain the output signals or gradients within a reasonable amount of time. Nevertheless, our work limits the focus to the implementations in the automatic differentiation framework.

Audio processing graph — Now, consider a composition of multiple differentiable processors; the entire graph remains differentiable due to the chain rule. However, the following practical considerations remain. If we fix the processing graph prior to the optimization and the graph size is relatively small, we can implement the "differentiable graph" following the existing implementations [7, 6]. That is, we compute every processor one by one in a predefined topological order. However, we have the following additional requirements. First, the pruning changes the graph during the optimization. Therefore, our implementation must take a graph and its parameters along with the source signals as input arguments for every forward pass. Note that this feature is also necessary when training a neural network that predicts the parameters of any given graph [1]. Second, the size of our graphs is much larger than the ones from previous works [7, 6, 9, 1]. In this case, the one-by-one computation severely bottlenecks the computation speed. Therefore, we derived a flexible and efficient graph computation algorithm (i) that can take different graphs for each forward pass as input and (ii) performs batched processing of multiple processors within a graph, utilizing the parallelism of GPUs. The details of this method are described in our companion paper [19]. Finally, we note that other than the differentiation with respect to the input signals $\mathbf{S}$ and parameters $\mathbf{P}$, one might be interested in differentiation with respect to the graph structure G. The proposed pruning method performs this to a limited extent; deletion of a node v_i is a binary operation that modifies the graph structure. We relaxed this to a continuous dry/wet weight w_i and optimized it with the audio loss L_{a} and regularization L_{p}.

A.3. Graph search

Several independent research efforts in various domains exist that search for graphs that satisfy certain requirements. For example, neural architecture search (NAS) aims to find a neural network architecture that achieves improved performance [20]. 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 ours is a differentiable architecture search (DARTS) [21], which relaxes the choice of each layer to a categorical distribution and optimizes it via gradient descent. Theoretically, our method can be naturally extended to this approach; we only need to change our 2-way choice (prune or not) to $(N+1)$-way (bypass or select one of N processor types). DARTS is clearly more flexible and general, allowing an arbitrary order of processors. However, it also greatly increases the computational cost, as we must compute all N processors to compute their weight sum for every node. For example, if we want to keep the mixing console structure and allow arbitrary processor choices, the memory complexity becomes $O(N^2)$ instead of the current $O(N)$. In other words, we must pay additional costs to increase the size of the search space. This cost increase is especially critical to us since we have to find a graph for every song. Another popular related domain is the generation/design of molecules with desired chemical properties [22]. 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 [23]. RL is an attractive choice since we can be completely free with prior assumptions on graphs, and we can use arbitrary quality measures that are not differentiable. We also note that RL can be used for NAS [24]. However, applying RL to our task has a risk of obtaining nontrivial mixing graphs that are difficult for practitioners to interpret; we may need a soft regularization penalty that guides the generation process towards familiar structures, e.g., ones like the pruned mixing consoles. Also, it may need much larger computational resources to explore the search space sufficiently.

B. DRY/WET PRUNING ALGORITHM

Algorithm 1 describes the details of the dry/wet method. For a simpler description, we modified the initialization to include per-type node sets and weights as in line 6-10. The termination condition is given in line 11. The trial candidate sampling is implemented in line 12-13. The candidate pool update is expanded to handle the trial successes and failures separately, shown in line 18 and 20-25.

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

[6]	*Task & 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 pre-defined directed acyclic graph (DAG). For example, a subtractive synthesizer that comprises 2 oscillators, 1 amplitude envelope, and 1 lowpass 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.
[10]	*Task & 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 & selected.
	Processors	Differentiable sinusoidal oscillators, each used as a carrier or modulator, pre-defined frequencies. An additional FIR reverb is added to the FM graph output for post-processing.
	Graph	DAGs with at most 6 operators. Different graphs for different target instruments.
	Method	Trained a convolutional neural network that estimates envelopes from the target loudness and pitch.
[9]	*Task & domain*	Sound matching $[x] \rightarrow [G, \mathbf{P}]$. Similar setup to the above [10] plus additional estimation of the operator graph G.
	Processors	Identical to [10], except for the frequency ratio that can be searched.
	Graph	A subgraph of a supergraph, which resembles a multi-layer 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.
[2]	*Task & domain*	Reverse engineering $[s, y] \rightarrow [G, \mathbf{P}]$ of an audio effect chain from a subtractive synthesizer (commercial plugin).
	Processors	5 audio effects: compressor, distortion, equalizer, phaser, and reverb. Non-differentiable 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.
[3]	*Task & domain*	Blind estimation $[y] \rightarrow [G]$ and reverse engineering $[s, y] \rightarrow [G]$ of guitar effect chains.
	Processors	13 guitar effects, including non-linear processors, modulation effects, ambiance effects, and equalizer filters.
	Graph	A chain of guitar effects. Maximum 5 processors and a total of 221 possible combinations.
	Method	Trained a convolutional neural network with synthetic data to predict the correct combination.
[5]	*Task & domain*	Automatic mixing $[\mathbf{S}] \rightarrow [\mathbf{P}]$. Estimated parameters of fixed processing chains from source tracks ($K \leq 16$).
	Processors	7 differentiable processors, where 4 (gain, polarity, fader, and panning) were implemented exactly. A combined effect of the remaining 3 (equalizer, compressor, and reverb) was approximated with a single pre-trained neural network.
	Graph	Tree structure: applied a fixed chain of the 7 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.
[8]	*Task & domain*	Reverse engineering of music mastering $[s, y] \rightarrow [\mathbf{P}]$
	Processors	A multi-band 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.
[1]	*Task & 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 processors are multiple-input multiple-output (MIMO), e.g., allowing auxiliary modulations. Non-differentiable implementations.
	Graph	Complex DAG; splits (e.g., multi-band processing) and merges (e.g., sum and modulation). 30 processors max.
	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).
[4]	*Task & domain*	Reverse engineering $[\mathbf{S}, y] \rightarrow [\mathbf{P}]$ of music mixing. Estimated parameters of a fixed chain for each track.
	Processors	6 differentiable processors: gain, equalizer, compressor, distortion, panning, and reverb.
	Graph	A chain of 5 processors (all 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 & 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	7 differentiable processors: gain/panning, stereo imager, equalizer, reverb, compressor, noisegate, and delay.
	Graph	A tree of processing chains with a subgrouping structure (any other DAG can also be used). 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.

Table 2: *Per-dataset results of the mixing consoles with different processor type configurations.*

		MedleyDB				MixingSecrets				Internal			
		L_a	L_{lr}	L_m	L_s	L_a	L_{lr}	L_m	L_s	L_a	L_{lr}	L_m	L_s
Base graph		50.7	1.45	1.42	198	7.30	2.16	2.02	22.9	1.12	.951	.940	1.63
+ Gain/panning	`g`	.550	.583	.485	.550	.876	.856	.819	.973	.642	.619	.597	.734
+ Stereo imager	`sg`	.541	.564	.483	.553	.847	.834	.791	.928	.538	.616	.595	.727
+ Equalizer	`e sg`	.450	.453	.390	.504	.700	.698	.622	.780	.522	.497	.467	.626
+ Reverb	`e sg r`	.368	.361	.360	.390	.614	.601	.579	.674	.463	.451	.432	.517
+ Compressor	`ec sg r`	.315	.304	.297	.356	.558	.542	.512	.637	.396	.377	.347	.482
+ Noisegate	`ecnsg r`	.302	.288	.281	.353	.548	.532	.502	.625	.393	.374	.343	.480
+ Multitap delay (full)	`ecnsgdr`	.296	.288	.284	.324	.545	.529	.502	.618	.385	.369	.338	.465

Table 3: *Per-dataset results of the pruning.* `MC`*: mixing console.* `BF`*: brute-force,* `DW`*: dry/wet, and* `H`*: hybrid.*

		MedleyDB									MixingSecrets									Internal								
	τ	L_a	ρ	ρ_g	ρ_s	ρ_e	ρ_r	ρ_c	ρ_n	ρ_d	L_a	ρ	ρ_g	ρ_s	ρ_e	ρ_r	ρ_c	ρ_n	ρ_d	L_a	ρ	ρ_g	ρ_s	ρ_e	ρ_r	ρ_c	ρ_n	ρ_d
`MC`	–	.296	–	–	–	–	–	–	–	–	.545	–	–	–	–	–	–	–	–	.385	–	–	–	–	–	–	–	–
`BF`	.01	.305	.63	.35	.81	.54	.77	.67	.71	.53	.566	.65	.50	.82	.43	.71	.61	.77	.75	.402	.80	.76	.92	.61	.81	.85	.87	.80
`DW`	.01	.302	.56	.32	.79	.42	.74	.57	.56	.43	.561	.57	.47	.82	.22	.62	.54	.74	.55	.397	.74	.74	.82	.49	.70	.87	.86	.61
`H`	.001	.295	.44	.22	.73	.26	.61	.50	.54	.24	.550	.43	.35	.76	.13	.42	.43	.55	.38	.388	.59	.48	.77	.40	.57	.78	.77	.39
	.01	.302	.61	.32	.81	.48	.74	.69	.71	.52	.563	.62	.49	.85	.34	.64	.60	.79	.65	.400	.77	.73	.93	.56	.75	.85	.86	.74
	.1	.375	.83	.59	.93	.83	.92	.80	.84	.90	.648	.84	.70	.91	.75	.86	.83	.93	.91	.474	.93	.90	.99	.84	.92	.93	.95	.96

Algorithm 1 *Music mixing graph search (dry/wet method).*

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

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

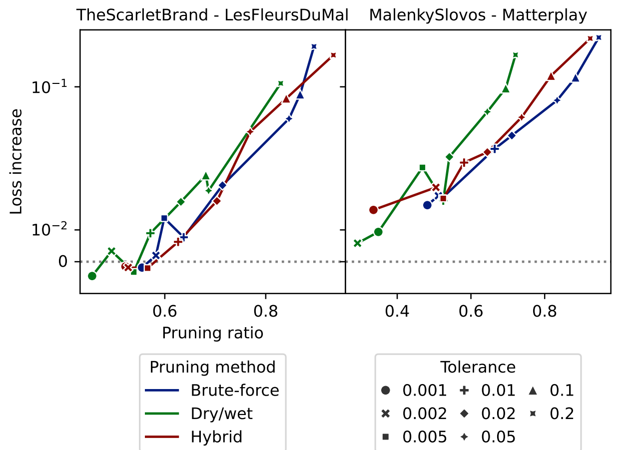


Figure 1: *Loss increases from the mixing console and pruning ratios for different pruning methods and tolerances.*

C. SUPPLEMENTARY RESULTS

- Table 2 and 3 report the per-dataset results on the mixing consoles and graph pruning, respectively.
- Figure 1 compares the pruning methods on 2 random-sampled songs using 7 tolerance settings from 0.001 to 0.2.
- Figure 2 shows multiple graphs obtained by pruning the same console (song) repeatedly.
- Refer to Figure 3-5 for more pruned graphs obtained with the default setting — hybrid method and $\tau = 0.01$.
- Figure 7-9 show more spectrogram plots.

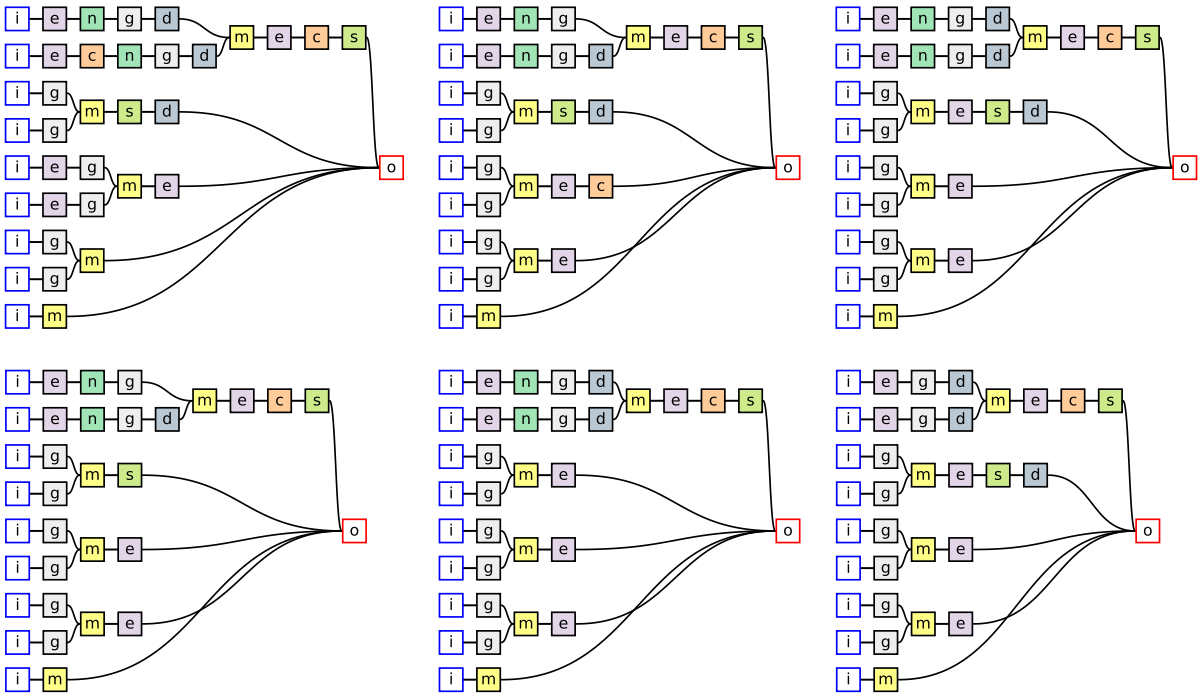
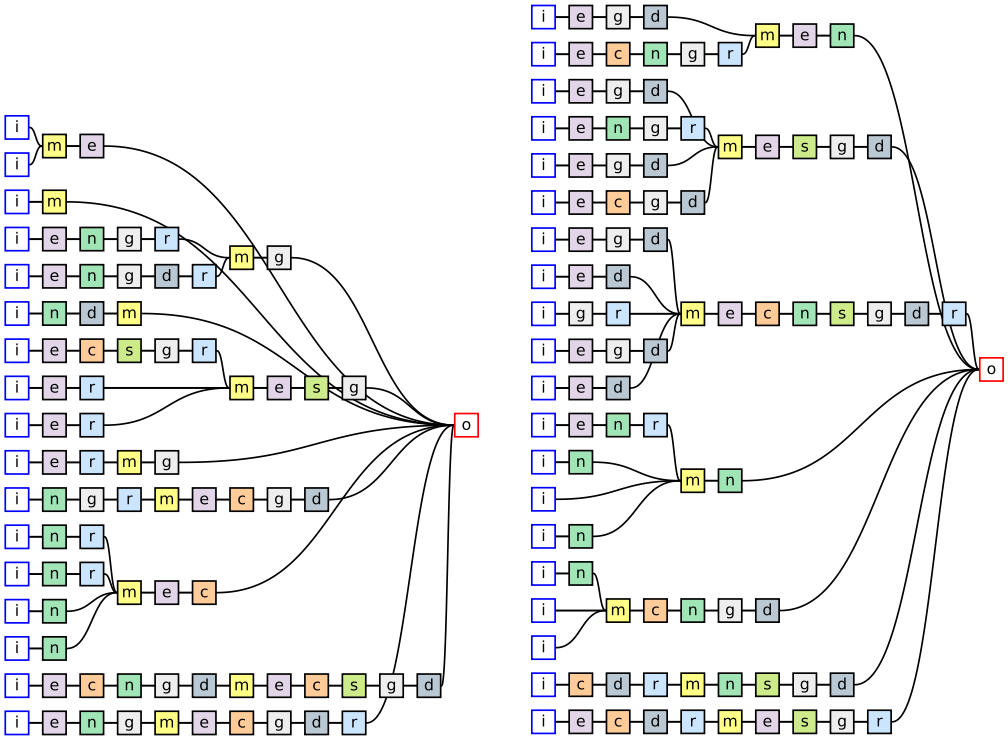
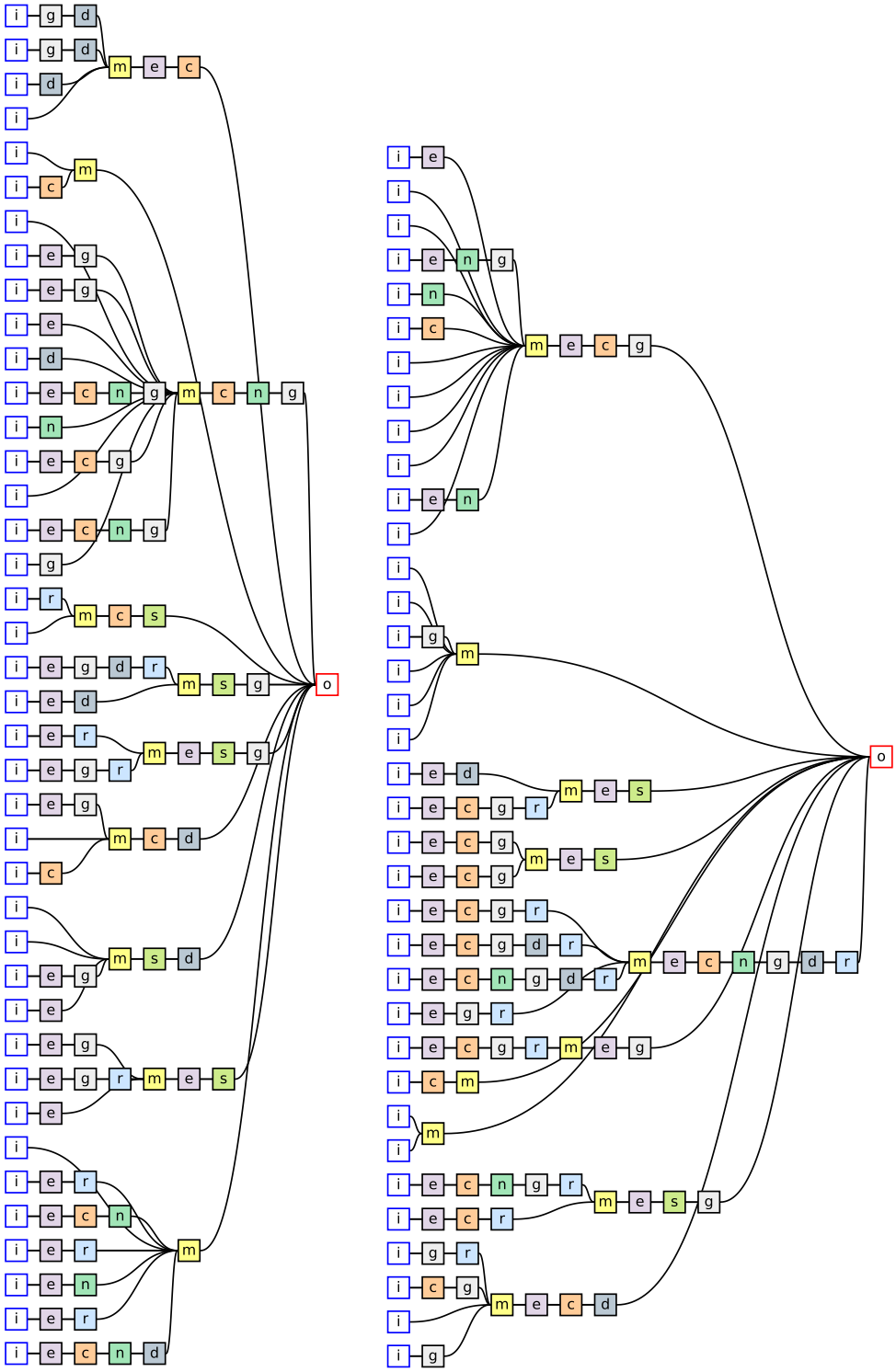


Figure 2: *Each pruning run (default setting) yields a slightly different graph. Song:* `EthanHein_GirlOnABridge`.



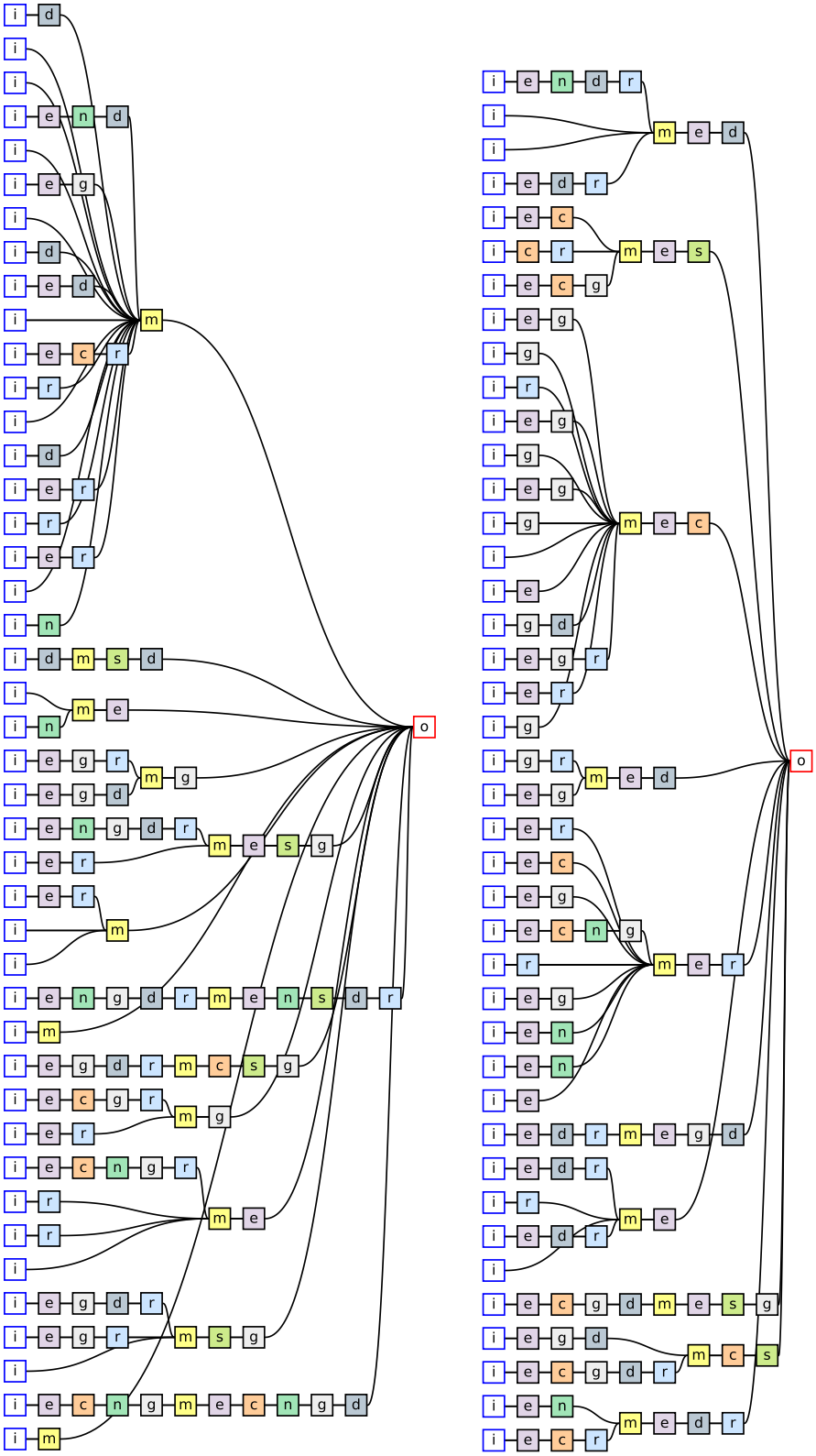
(a) `Internal_part1_65536` (b) `DonCamilloChoir_TrudeTheBumblebee`

Figure 3: *Example pruned graphs (default setting). the number of tracks:* $K \leq 20$.



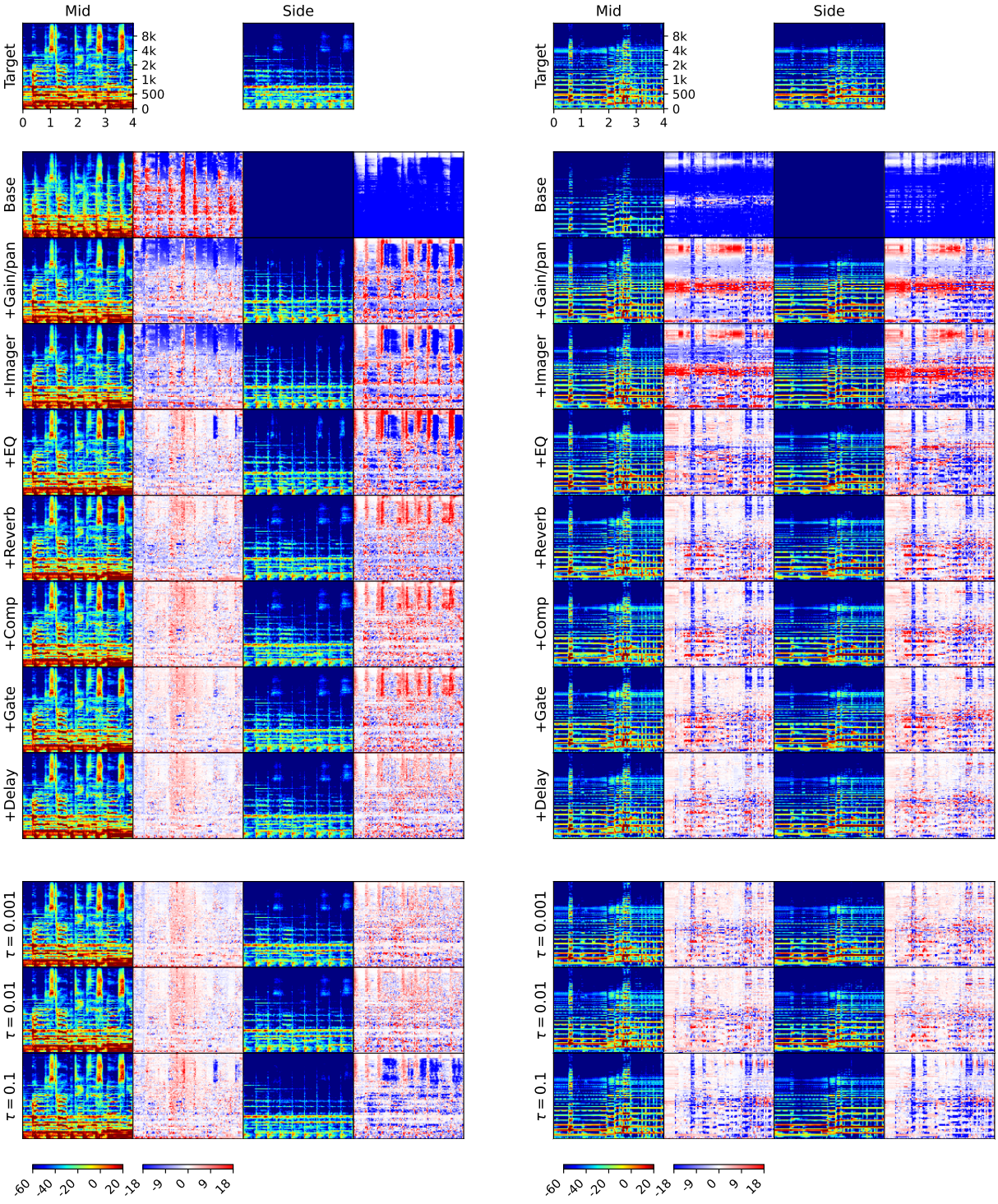
(a) `LittleTybee_TheAlchemist` (b) `RaftMonk_Tiring`

Figure 4: *Example pruned graphs (default setting). the number of tracks:* $K > 20$.



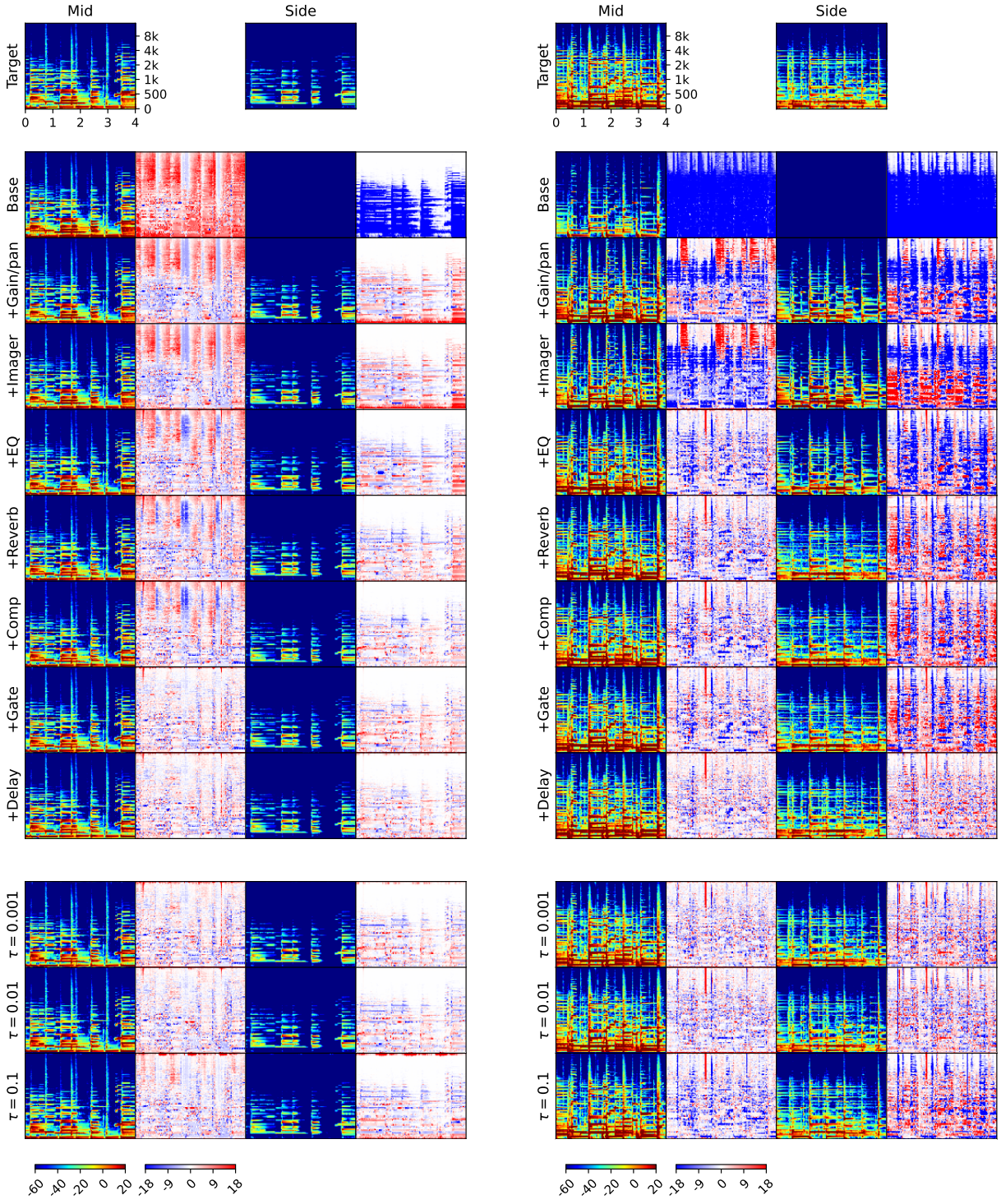
(a) `Internal_part2_67692` (b) `StevenClark_Bounty`

Figure 5: *Example pruned graphs (default setting). the number of tracks:* $K > 20$ *(continued).*



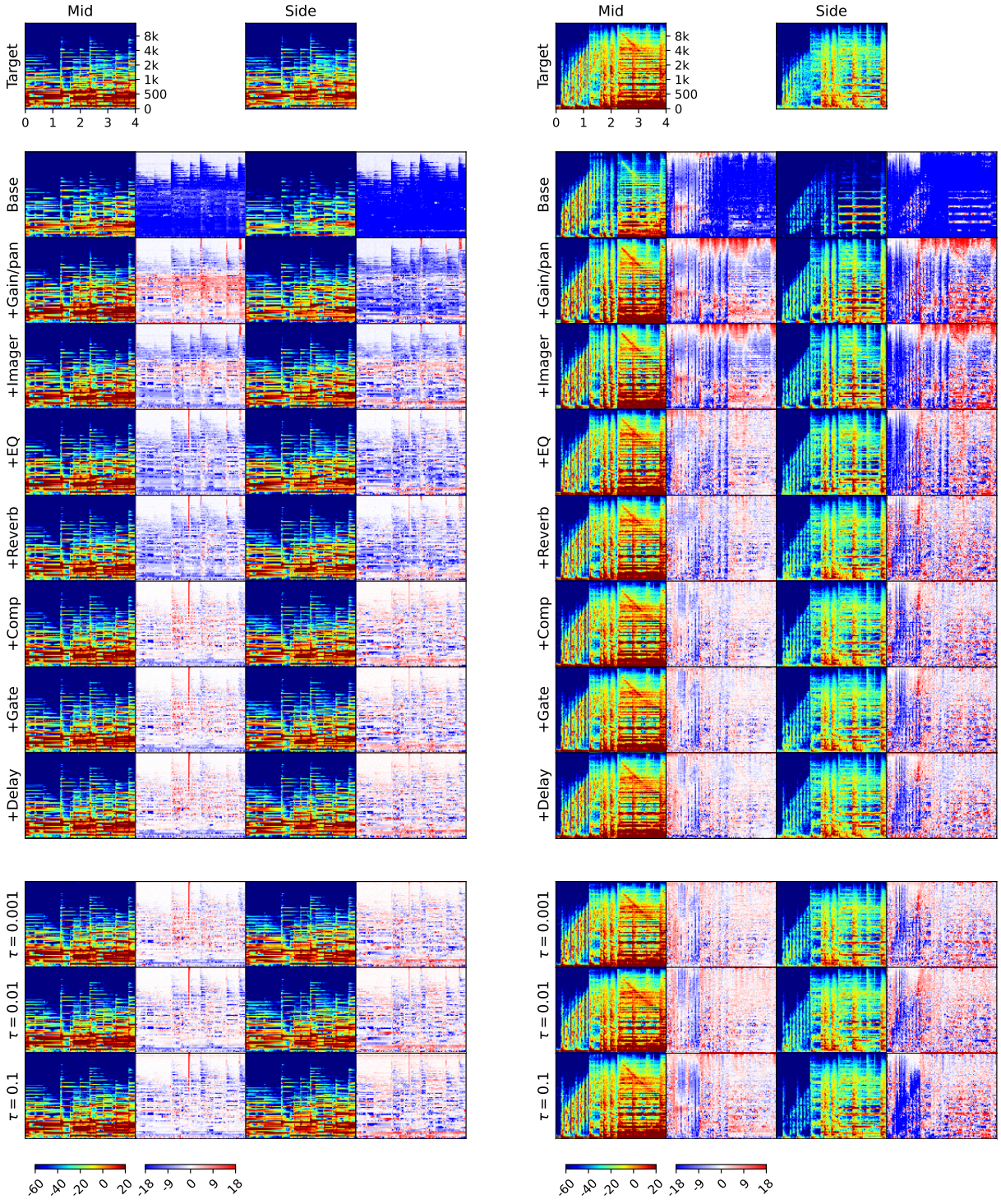
(a) `Torres_NewSkin` (b) `TablaBreakbeatScience_RockSteady`

Figure 6: *Matching of target music mixes with mixing consoles and their pruned versions:* `MedleyDB` *dataset.*



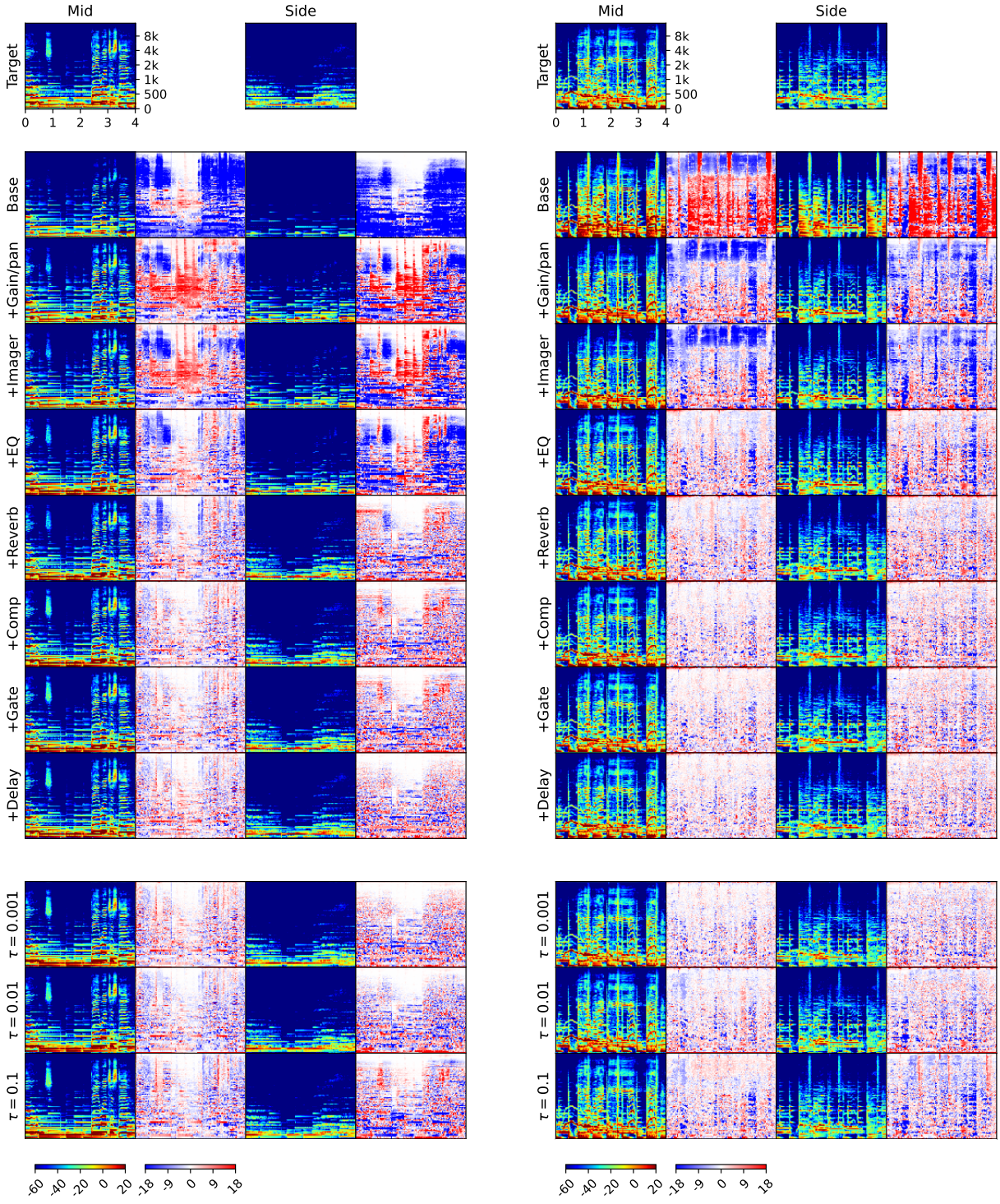
(a) `MusicDelta_SwingJazz` (b) `ChrisJacoby_BoothShotLincoln`

Figure 7: *Matching of target music mixes with mixing consoles and their pruned versions:* `MedleyDB` *dataset.*



(a) `HowlProject_IfIWereABell` (b) `IanDearden_TeraniaCreekWalking`

Figure 8: *Matching of target music mixes with mixing consoles and their pruned versions:* `MixingSecrets` *dataset.*



(a) `Internal_66680` (b) `Internal_67954`

Figure 9: *Matching of target music mixes with mixing consoles and their pruned versions:* `Internal` *dataset.*

D. REFERENCES

[1] S. Lee, J. Park, S. Paik, and K. Lee, "Blind estimation of audio processing graph," in *IEEE ICASSP*, 2023.

[2] C. Mitcheltree and H. Koike, "SerumRNN: Step by step audio VST effect programming," in *Artificial Intelligence in Music, Sound, Art and Design*, 2021.

[3] J. Guo and B. McFee, "Automatic recognition of cascaded guitar effects," in *DAFx*, 2023.

[4] J. Colonel, "Music production behaviour modelling," 2023.

[5] C. J. Steinmetz, J. Pons, S. Pascual, and J. Serrà, "Automatic multitrack mixing with a differentiable mixing console of neural audio effects," in *IEEE ICASSP*, 2021.

[6] N. Uzrad *et al.*, "DiffMoog: a differentiable modular synthesizer for sound matching," *arXiv:2401.12570*, 2024.

[7] J. Engel, L. H. Hantrakul, C. Gu, and A. Roberts, "DDSP: differentiable digital signal processing," in *ICLR*, 2020.

[8] M. A. Martínez-Ramírez, O. Wang, P. Smaragdis, and N. J. Bryan, "Differentiable signal processing with black-box audio effects," in *IEEE ICASSP*, 2021.

[9] 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," *arXiv:2305.12868*, 2023.

[10] F. Caspe, A. McPherson, and M. Sandler, "DDX7: Differentiable FM synthesis of musical instrument sounds," in *ISMIR*, 2022.

[11] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," *NeurIPS*, 2019.

[12] C. J. Steinmetz, N. J. Bryan, and J. D. Reiss, "Style transfer of audio effects with differentiable signal processing," *JAES*, vol. 70, no. 9, 2022.

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

[14] A. Carson, S. King, C. V. Botinhao, and S. Bilbao, "Differentiable grey-box modelling of phaser effects using frame-based spectral processing," in *DAFx*, 2023.

[15] S. Nercessian, "Neural parametric equalizer matching using differentiable biquads," in *DAFx*, 2020.

[16] S. Lee, H.-S. Choi, and K. Lee, "Differentiable artificial reverberation," *IEEE/ACM TASLP*, vol. 30, 2022.

[17] B. Hayes, J. Shier, G. Fazekas, A. McPherson, and C. Saitis, "A review of differentiable digital signal processing for music & speech synthesis," *Frontiers in Signal Process.*, 2023.

[18] C. J. Steinmetz, T. Walther, and J. D. Reiss, "High-fidelity noise reduction with differentiable signal processing," in *AES Convention 155*, 2023.

[19] S. Lee *et al.*, "GRAFX: an open-source library for audio processing graphs in PyTorch," in *DAFx (Demo)*, 2024.

[20] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, vol. 20, no. 55, 2019.

[21] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *ICLR*, 2019.

[22] 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," *Molecular Systems Design & Engineering*, vol. 4, no. 4, 2019.

[23] J. You, B. Liu, Z. Ying, V. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," *NeurIPS*, 2018.

[24] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv:1611.01578*, 2016.