

Ph.D. DISSERTATION

SOLVING INVERSE PROBLEMS OF
AUDIO SYSTEMS WITH
DIFFERENTIABLE PROCESSING
AND GENERATIVE MODELING

미분 가능한 신호 처리 및 생성 모델링을 활용한
오디오 시스템의 역문제 해결

SUNGHO LEE

AUGUST 2026

DEPARTMENT OF INTELLIGENCE AND INFORMATION
GRADUATE SCHOOL OF CONVERGENCE SCIENCE AND
TECHNOLOGY
SEOUL NATIONAL UNIVERSITY

공학박사학위논문

미분 가능한 신호 처리 및 생성 모델링을
활용한 오디오 시스템의 역문제 해결

SOLVING INVERSE PROBLEMS OF AUDIO
SYSTEMS WITH DIFFERENTIABLE PROCESSING
AND GENERATIVE MODELING

2026년 8월

서울대학교 융합과학기술대학원
지능정보융합학과
이 성 호

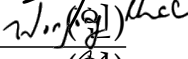
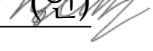
SOLVING INVERSE PROBLEMS OF AUDIO SYSTEMS WITH DIFFERENTIABLE PROCESSING AND GENERATIVE MODELING

미분 가능한 신호 처리 및 생성 모델링을 활용한 오디오
시스템의 역문제 해결

지도교수 이 교 구
이 논문을 공학박사 학위논문으로 제출함
2026년 5월

서울대학교 융합과학기술대학원
지능정보융합학과
이 성 호

이성호의 공학박사 학위 논문을 인준함
2026년 7월

위 원 장:	곽 노 준	
부위원장:	이 교 구	
위 원:	이 원 종	
위 원:	남 주 한	
위 원:	정 다 샘	

Abstract

Audio signals are rarely delivered to listeners as they are produced; they pass through transformations imposed by physical environments, electronic devices, and computational processes. We refer to the objects or structures that impose these transformations as systems, and recovering such systems from observed audio constitutes an inverse problem that arises across audio applications, from reproducing room acoustics in virtual reality to reverse engineering a music mix. This thesis develops methods for solving such inverse problems using two complementary paradigms: differentiable signal processing, which enables gradient-based optimization through classical audio systems, and generative modeling, which learns conditional distributions over system descriptions given observations to address the inherent ambiguity of inverse problems. We study two domains: room acoustics, where systems are well approximated as linear and time-invariant, and musical signal processing, where systems are compositional graphs of diverse processors. Across both domains, we make their representative systems differentiable, apply generative modeling, and explore hybrids of the two approaches, and thereby show that domain knowledge and data-driven methods can be combined to produce interpretable, structured solutions while preserving the scalability of learning-based approaches.

keywords: Inverse problem, differentiable processing, generative modeling, audio post-production, room acoustics, audio effects, music production.

student number: 2020-22847

Contents

Abstract	i
Contents	ii
List of Tables	ix
List of Figures	xi
1 Introduction	1
1.1 Inverse Problems	2
1.1.1 Definitions	2
1.1.2 Linear System	4
1.1.3 Ill-posedness and Ambiguity	5
1.2 Audio Systems	6
1.2.1 Linear Time-Invariant (LTI) Systems	7
1.2.2 Linear Time-Varying (LTV) Systems	8
1.2.3 Nonlinear Systems	9
1.2.4 Compositional Systems	11
1.3 Tasks of Interest	13
1.3.1 Room Acoustic System Estimation	13
1.3.2 Audio Processing Graph Estimation	15
1.4 Key Challenges	16
1.5 Methodology	18
1.5.1 Differentiable Processing	18
1.5.2 Generative Models	22

1.5.3	Hybrid Approaches	23
1.6	Contributions and Thesis Organization	24
1.6.1	Chapter-by-Chapter Contributions	24
1.6.2	Broader Impact	29
1.6.3	Publications	29
2	Differentiable Artificial Reverberation	31
2.1	Introduction	31
2.2	Related Works	33
2.2.1	Differentiable Digital Signal Processing	33
2.2.2	ARP Estimation	33
2.2.3	Modeling Reverberation in the Deep Learning Framework	35
2.3	Frequency-Sampling Method for Differentiable IIR Filters	35
2.3.1	Reliability of the Frequency-Sampling Method	36
2.3.2	Differentiable Dense IIR Filter with State Variable Filters	37
2.3.3	Differentiable Parametric Equalizer	39
2.4	Differentiable Artificial Reverberation	40
2.4.1	Differentiable Filtered Velvet Noise	40
2.4.2	Differentiable Advanced Filtered Velvet Noise	46
2.4.3	Differentiable Delay Network	46
2.5	Artificial Reverberation Parameter Estimation	51
2.5.1	Reference Encoder	52
2.5.2	ARP-Groupwise Projection Layers	52
2.5.3	Nonlinear Activation Functions and Bias Initialization	52
2.5.4	Loss Function	53
2.6	Experimental Setup	56
2.6.1	Data	56
2.6.2	Training	57
2.6.3	Evaluation Metrics	57
2.6.4	Subjective Listening Test	58
2.6.5	Baselines	59
2.7	Evaluation Results	60
2.7.1	Comparison with the Baseline Models	60

2.7.2	Reliability of the DAR Models	65
2.7.3	Performance Difference Between Target Tasks	66
2.7.4	AR Model Efficiency and Estimation Performance	66
2.8	Conclusion	68
2.A	Ablations/Comparisons of the AR/DAR Models	69
2.A.1	AR/DAR Model Ablations	69
2.A.2	Comparisons on Different Coloration Filters	70
2.A.3	Effects of the Frequency-Sampling Resolution	71
2.B	Details on the Baseline Methods	71
2.B.1	ARP Match Models	71
2.B.2	CNN Decoder Models	73
2.C	Details on the Frequency-Sampling Method	73
2.C.1	Proof of Equation (2.6)	73
2.C.2	Proof of Equation (2.9)	74
3	Differentiable Acoustic Radiance Transfer	75
3.1	Introduction	75
3.2	Background	77
3.2.1	Acoustic Rendering Equation	77
3.2.2	Acoustic Radiance Transfer	81
3.3	Methods	85
3.3.1	Kernel Decomposition	85
3.3.2	Material Parameterization	88
3.3.3	Radiance Computation	89
3.3.4	Injection and Detection	91
3.3.5	Learnable Directivity	93
3.4	Experimental Setup	93
3.4.1	Benchmarks	93
3.4.2	Metrics	96
3.4.3	Implementation Details	96
3.4.4	Optimization	97
3.4.5	Baselines	100
3.5	Results	101

3.5.1	Main Results	101
3.5.2	Qualitative Analysis	103
3.5.3	Material Parameterization Comparison	106
3.6	Further Analysis	107
3.6.1	Effect of Measurement Quantity	107
3.6.2	Sensitivity to Geometry Distortion	107
3.6.3	Ablation Study	110
3.6.4	Computational Cost	111
3.7	Conclusion	113
3.A	Delay Network Interpretation of ART	115
3.B	Derivations	117
3.B.1	Visibility-Material Decomposition (3.25)	117
3.B.2	Material Matrix-Based Injection (3.40)	118
3.C	Aliasing Suppression	119
3.D	RIR Estimation with DART Echograms	120
3.E	Geometric Resolution	121
4	Autoregressive Modeling of Neural Discrete Representations for Room	
	Impulse Response Estimation	123
4.1	Introduction	123
4.2	Discrete Representation Learning with RQ-VAE	125
4.2.1	Input Representation	125
4.2.2	Autoencoding	126
4.2.3	Residual Quantization	126
4.2.4	Differences to Existing Neural Audio Codecs	127
4.3	Autoregressive Token Generation with Axial Transformers	127
4.3.1	Factorization Strategies	128
4.3.2	Transformer Architecture	131
4.3.3	Inference	132
4.4	Conditional Generation with Reference Encoder	132
4.4.1	Reference Encoder	132
4.4.2	Acoustic Matching	134
4.5	Experiments	134

4.5.1	Data	134
4.5.2	Metrics	136
4.5.3	Training	137
4.6	Evaluation Results	137
4.6.1	Autoencoding	137
4.6.2	Analysis-Synthesis	137
4.6.3	Blind Estimation	139
4.6.4	Acoustic Matching	140
4.6.5	Decoding Strategy Comparison	140
4.6.6	Ablation Study and Baselines	143
4.7	Hybrid Extension: Efficient Acoustic Scene Transfer with Differentiable Decoding	143
4.7.1	Factorized Codec with a Filtered-Noise Late Reverberation Decoder	144
4.7.2	Time-Frequency Delay Pattern	146
4.7.3	Efficient CPU Inference	147
4.8	Conclusion	148
5	Differentiable and Efficient Computation of Audio Processing Graphs	149
5.1	Introduction	149
5.2	Graph Definition and Representation	150
5.2.1	Mutable and Tensor Representations	151
5.3	Batched Graph Rendering	152
5.3.1	Type scheduling	153
5.3.2	Implementation details	154
5.4	Music Mixing Applications	155
5.4.1	Mixing Graph Construction	155
5.4.2	Speed benchmark	155
5.5	Conclusion	156
6	Reverse Engineering of Music Mixing Graphs with Differentiable Processors and Iterative Pruning	158
6.1	Introduction	158

6.1.1	Problem Formulation	159
6.1.2	Proposed Strategy	160
6.1.3	Open-Source Implementation	162
6.2	Related Works	165
6.2.1	Composition of Audio Processors	165
6.2.2	Graph Search	165
6.3	Differentiable Processors	166
6.4	Music Mixing Consoles	170
6.4.1	Optimization	171
6.4.2	Evaluation Metrics	172
6.4.3	Data	175
6.4.4	Results	175
6.5	Music Mixing Graph Search	176
6.5.1	Iterative Pruning	177
6.5.2	Candidate Sampling	178
6.5.3	Optimization	182
6.6	Results	183
6.6.1	Sparsity and Quality Degradation	183
6.6.2	Choice of Tolerance	183
6.6.3	Pruning Process	185
6.6.4	Sampling Method Comparison	185
6.6.5	Subjective Listening Test	187
6.6.6	Characteristics of Iterative Pruning	191
6.6.7	Consistency Analysis	194
6.6.8	Full Results	195
6.7	Conclusion	197
7	Blind Estimation of Audio Processing Graphs	200
7.1	Introduction	200
7.2	Training Data Generation	201
7.2.1	Prototype Graph Sampling	202
7.2.2	Adaptive Parameter Randomization	203
7.2.3	Properties of Synthetic Graphs	204

7.3	Blind Estimation System	205
7.3.1	Reference Encoder	205
7.3.2	Prototype Decoder	205
7.3.3	Parameter Estimator	208
7.3.4	Training	209
7.4	Experimental Setup	209
7.4.1	Data	209
7.4.2	Metric	210
7.4.3	Baselines	211
7.5	Evaluation Results	213
7.5.1	Sanity Check	213
7.5.2	Performance Analysis	214
7.5.3	Blind Estimation Strategy Comparison	215
7.6	End-to-End Refinement with Differentiable Processing	216
7.6.1	Fine-Tuning with Differentiable Rendering	216
7.6.2	Refinement Results	218
7.7	Conclusion	220
8	Conclusions	221
8.1	Thesis Summary	221
8.2	Discussion	222
8.2.1	Expressivity and Model Choice	222
8.2.2	Data Availability and Generalization	223
8.2.3	Objectives for Optimization and Evaluation	224
8.2.4	Alternative Estimation Strategies	226
8.2.5	System-Aware Representation Learning	226
8.2.6	Audio Post-Production Workflows and Deployment	227
	References	229
	초 록	266
	감사의 글	267

List of Tables

1.1	Notation for the inverse-problem framework and its audio instantiations	14
1.2	Thesis summary and organization	25
2.1	Target ARP tensors and nonlinear activation functions	54
2.2	Ablation on the AR/DAR model configurations	69
2.3	Differentiable FVN with various coloration filters and parameterization approaches	70
2.4	Effects of frequency-sampling resolution and regularization on the match loss	71
3.1	Main benchmark results	99
3.2	Computation time of each operation in DART	112
3.3	Ablation study of DART	114
3.4	Impulse response prediction results	122
3.5	Effects of varying spatial and directional resolutions on DART (UC)	122
4.1	IR estimation tasks and usable speech/singing dataset categories	136
4.2	Evaluation results of the proposed methods and baselines	138
6.1	A summary and comparison of previous works on the estimation of compositional audio signal processing	163
6.2	Matching performance of mixing console configurations (MedleyDB)	173
6.3	Matching performance of mixing console configurations (MixingSecrets)	174

6.4	Pruning results with different strategies and tolerance τ (MedleyDB)	180
6.5	Pruning results with different strategies and tolerance τ (MixingSe- crets)	180
6.6	Processor detection performance with consistency check	195
6.7	Pruning results of the full dataset	196
7.1	List of the processors and their configurations	202
7.2	Evaluation results of the proposed method and other baselines . . .	217
7.3	Differentiable processing refinement on the singing voice task . . .	219

List of Figures

1.1	Audio system modeling and description	3
1.2	An example audio processing graph G	13
1.3	Two key methods for audio system estimation	19
2.1	The proposed ARP estimation framework	32
2.2	The frequency-sampling method with a various number of sam- pling points N	38
2.3	Modified FVN, AFVN's RIR approximation, and their differentiable implementation strategy	42
2.4	EDR of IRs generated by differentiable FVN, AFVN, their original AR counterparts, and EDR errors introduced during the AR-to-DAR conversion process	43
2.5	Modifications to the original FVN and AFVN	45
2.6	Our DN in the time domain	48
2.7	EDR of differentiable DN, its original DN with two versions (LTI and time-varying), and EDR errors between those three	50
2.8	Architecture of the ARP estimation network	55
2.9	Baseline methods for the experiments	59
2.10	ARP estimation results on various tasks with the AR/DAR models	61
2.11	EDR plots of a reference RIR (in short T_{30} range) and its estimations.	62
2.12	EDR plots of a reference RIR (in mid T_{30} range) and its estimations.	63
2.13	EDR plots of a reference RIR (in long T_{30} range) and its estimations.	64
2.14	Reliability of the DAR models	65
2.15	Magnitude responses of the coloration filters of the AR models . . .	67

3.1	The acoustic rendering equation (ARE) and acoustic radiance transfer (ART)	78
3.2	Overview of DART	82
3.3	Decomposition of the reflection kernel	87
3.4	Room geometries of the Hearing Anything Anywhere (HAA) dataset	94
3.5	Room geometries of the Coupled Room (CR) dataset	95
3.6	Echogram residuals on the CR dataset, unseen split	102
3.7	Optimized reflection coefficients of parametric DART, CR unseen split	104
3.8	Spatial energy distribution predicted by DART	105
3.9	Example learned scattering patterns of DART	108
3.10	Test results with varying measurements and geometric distortion .	109
4.1	The proposed method	125
4.2	RQ-VAE encoder and resampling filter banks	127
4.3	Coarse-to-fine modeling of IRs with RQ-VAE	128
4.4	AudioLM-like token generation strategy	129
4.5	RQ-Transformer-like token generation strategy	130
4.6	VALL-E-like token generation strategy	131
4.7	Axial transformer architecture	132
4.8	Reference encoder architecture	133
4.9	Reverberation parameter statistics of the IR datasets	135
4.10	Energy decay relief waterfalls of ground-truth RIRs, RQ-VAE reconstructions, and RQ-Transformer estimations	139
4.11	Scatter plot of estimated versus ground-truth reverberation parameters across test RIRs	141
4.12	Ground-truth versus predicted per-octave-band magnitude responses, same test set	142
4.13	Scatter plots between MUSHRA-like subjective similarity scores and objective losses or reverberation parameter errors	144
4.14	Hybrid codec architecture	145
5.1	Various schedules for batched node processing	153
5.2	Music mixing graph construction with GRAFX	156

5.3	Speed benchmark results of various scheduling methods	157
6.1	The proposed mixing graph search method	161
6.2	Example full and pruned mixing graphs	162
6.3	Process of iterative pruning (hybrid, $\tau = 0.01$)	181
6.4	Pruning results (hybrid) with various tolerances	184
6.5	Each node's weight and loss increase when pruned	186
6.6	Loss increases from the mixing console and pruning ratios for dif- ferent pruning methods and tolerances	187
6.7	Subjective listening test results	188
6.8	Scatter plots of MUSHRA scores and objective metrics	189
6.9	Scatter plots of MUSHRA scores and objective metrics (cont.)	190
6.10	Representative log-magnitude spectrograms	192
6.11	Representative log-magnitude spectrograms (cont.)	193
6.12	Graphs obtained with multiple pruning runs (default setting)	194
6.13	Statistics of the consoles and pruned graphs (full data)	197
7.1	Properties of the synthetic audio processing graphs	204
7.2	The proposed blind estimation framework	205
7.3	Blind estimation with Tokenized Graph Transformer	206
7.4	An inference result from the proposed method on the drum mixing estimation task	214
7.5	Inference results from the proposed method on the singing voice effect estimation task	215
7.6	End-to-end refinement with differentiable processing	216

Chapter 1

Introduction

Sound is rarely delivered to listeners as is. A voice reverberates through its surrounding environment before reaching a microphone; a guitarist shapes their tone through a chain of pedals; and a mixing engineer balances dozens of tracks to produce a final record. In each case, one or more physical, electronic, or computational entities transform an input signal into the audio we ultimately hear. We refer to these entities as *systems*.

Recovering the unobserved causes of transformed audio is a long-standing challenge in audio signal processing. Much of the literature focuses on estimating the input signal itself, as in denoising, dereverberation, or restoration from nonlinear distortion. This thesis instead focuses on the equally important task of estimating the system, or some properties of it, responsible for the observed transformation.

What are the acoustic properties of the room captured in this recording?

What processing chain was used to produce this guitar tone?

How was this song mixed and mastered?

Such questions arise in a wide range of practical scenarios. For instance, in film post-production, dialogue recorded on set is often re-recorded in a studio and then made to sound *as if* it were captured in the original environment, which requires estimating the acoustic properties of that environment from the original recording. In music production, a producer may wish to replicate the sonic character of a

reference track, or an educator may want to analyze and decompose the mixing decisions behind a finished record.

Unlike a signal, a system induces a mapping over signals; it therefore lives in a different representation space and calls for dedicated methods. This thesis investigates these *inverse problems of audio systems* using two modern techniques: *differentiable processing* and *generative modeling*.

1.1 Inverse Problems

1.1.1 Definitions

We begin with a general formulation of an inverse problem [1, 2]. Let x denote an input (or source) signal and y the corresponding observed (transformed) output, related by

$$y = \mathcal{F}(x). \quad (1.1)$$

We call $\mathcal{F}$ the *true process*, mapping input signals to their observed outputs. In general, $\mathcal{F}$ is unknown, high-dimensional, or analytically intractable (and so, in general, is the input x). To make $\mathcal{F}$ computationally tractable, we introduce a *proxy model*:

$$\hat{y} = f(x; H), \quad (1.2)$$

where H is the *system description*, a finite-dimensional mathematical object that characterizes the system (e.g., a single number, vector, matrix, integration kernel, set of system parameters, program code, or natural-language specification), and $f(\cdot; H)$ is the *forward map*, a computational procedure that applies H to an input and produces an estimate of the output. The input x and output y can be a vector, multiset, natural language, or another data type; the forward map can be implemented as an analytical function, a general numerical procedure, or a neural network.

Figure 1.1 summarizes the separation between a true process, its proxy forward map, and the system description. In this formulation, the description H specifies *what* the system is, while f specifies *how* it acts on signals. For example, a matrix H is a description, while the matrix–vector product $\hat{y} = Hx$ is the corre-

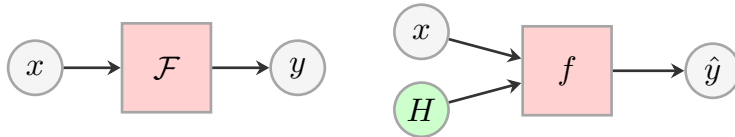


Figure 1.1: **Audio system modeling and description.** (Left) The true process $\mathcal{F}$ maps the input x to the observed output y ; (right) the proxy forward map f uses the input x and system description H to produce $\hat{y} = f(x; H)$.

sponding forward map. As another example, the true process $\mathcal{F}$ may be acoustic wave propagation in a room. A proxy may describe this process using a partial differential equation (PDE), with H representing boundary or material properties and f denoting the numerical discretization and solver.

The gap between $\mathcal{F}$ and the proxy $f(\cdot; H)$ is a deliberate modeling choice. A more expressive description space for H , or a more expressive forward map f , can cover a wider range of behaviors of $\mathcal{F}$. A compact choice restricts what is representable and thereby acts as an implicit prior. Expressivity, compactness, interpretability, and controllability may trade off in practice, but none of these relationships is necessary.

When the distinction between the true process and its proxy is not central to the discussion, we suppress it and use f to denote both.

The inverse problem is to estimate one or more unobserved variables from the available *observation* $\mathcal{O}$, the collection of variables that are accessible. Common variants include:

- $\mathcal{O} = (H, y)$: the usual (informed) inverse problem, where the system is known and we seek to recover the input x .
- $\mathcal{O} = (x, y)$: both input and output are known, i.e., *system identification* or *reverse engineering*.
- $\mathcal{O} = y$: only the output is observed, i.e., *blind estimation*. One may wish to recover x , H , or both jointly.

Our primary focus is on the system inverse problem, i.e., recovering the description H from the available observation. We elaborate on specific task variants in the audio domain in Section 1.3.

1.1.2 Linear System

Before addressing audio systems, we illustrate the framework on a simple linear system. Throughout this subsection, we take $x \in \mathbb{R}^M$ and $y \in \mathbb{R}^N$ to be finite-dimensional vectors, and the description H to be a real matrix that acts on the input by matrix–vector multiplication:

$$y = Hx. \quad (1.3)$$

Given K input–output pairs sampled from this system, collected as columns of $\mathbf{X} \in \mathbb{R}^{M \times K}$ and $\mathbf{Y} \in \mathbb{R}^{N \times K}$, our goal is to estimate H . The number of pairs K shapes the problem: when $K < M$ the system is underdetermined and additional priors are needed to identify a unique H ; when $K \geq M$ and $\mathbf{X}$ has full row rank, the data alone constrain the solution. The quality of an estimate $\hat{H}$ is judged by a *goodness-of-fit metric* $d(y, \hat{y})$ between the observed output y and the model output $\hat{y} = \hat{H}x$. Given K input–output pairs, a best-fitting estimate is therefore

$$\hat{H} \in \arg \min_H \sum_{k=1}^K d(y_k, Hx_k). \quad (1.4)$$

The classical choice is the squared L_2 norm:

$$d(y, \hat{y}) = \|y - \hat{y}\|_2^2. \quad (1.5)$$

For a linear system $y = Hx$, this yields the well-known pseudoinverse solution:

$$\hat{H} = \mathbf{Y}\mathbf{X}^\dagger = \mathbf{Y}\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}. \quad (1.6)$$

Having both inputs and outputs available, i.e., the reverse engineering setting ($\mathcal{O} = (\mathbf{X}, \mathbf{Y})$), is what makes this closed-form solution tractable; in the blind setting ($\mathcal{O} = y$), $\mathbf{X}$ is unavailable, so recovering the input, the system, or both requires additional assumptions or priors on x , H , or their joint distribution.

The pseudoinverse is the minimum-norm least-squares solution under the L_2 metric. What is defined to be similar or distant is ultimately domain-specific, and the choice of metric shapes the problem: it determines the uniqueness of the

solution, the tractability of finding it analytically, and the computational landscape of the search. Some combinations admit closed-form solutions, as the L_2 metric does with a linear forward map. More generally, a metric may be differentiable or non-differentiable, while the resulting objective depends jointly on the metric, the forward map, and its parameterization. The resulting objective may be convex or non-convex, with the latter being more common.

1.1.3 Ill-posedness and Ambiguity

Most real-world inverse problems, including those studied in this thesis, are ill posed: an exact description may not exist, a best-fitting description may not be unique, and the estimate may depend unstably on the observation.

Proxy mismatch can prevent the forward model from reproducing the observation exactly, leaving a nonzero irreducible residual even for a best-fitting description. Even when a best-fitting description exists, it may not be unique. Such ambiguity arises when the available evidence does not distinguish between descriptions, either because the observations are limited or uncertain, or because distinct descriptions produce equivalent behavior under the forward map. This may result in multiple descriptions attaining the same minimum, or in several descriptions becoming practically indistinguishable because they fit the observation nearly equally well. In the latter case, small changes in the observation may also lead to substantially different estimates.

Ambiguity can sometimes be removed in specific cases. For example, when it arises solely from a known equivalence between descriptions, such as permutation symmetry, one can choose a canonical representation or treat equivalent descriptions as the same system. In general, however, the observation alone may not determine which of several plausible descriptions should be selected. Additional assumptions or prior information can then be used in two complementary ways: regularized point estimation uses them to select and stabilize a single description, whereas probabilistic inference uses them to characterize the relative plausibility of different descriptions.

Regularized point estimation augments the data-fit objective with a preference

over descriptions:

$$\hat{H} = \arg \min_H [d(y, f(x; H)) + \lambda r(H)], \quad (1.7)$$

where $r(H)$ favors descriptions with the desired properties and λ controls its influence. The regularizer can select among descriptions that fit the observation equally or nearly equally well and can reduce the sensitivity of the estimate to perturbations of the observation. Such preferences need not take the form of an explicit penalty: the description space of H and the structure of the forward map f can also restrict which solutions are admissible.

Probabilistic inference instead represents uncertainty over descriptions:

$$p(H \mid \mathcal{O}) \propto p(\mathcal{O} \mid H) p(H), \quad (1.8)$$

where the likelihood $p(\mathcal{O} \mid H)$ describes how the observation may arise from a given description, and the prior $p(H)$ expresses which descriptions are plausible before observing $\mathcal{O}$. The resulting posterior retains the relative plausibility of multiple descriptions rather than requiring one of them to be selected. Depending on what is observed, additional assumptions may also be required about the input x or its relationship with H , particularly in blind settings.

The two formulations are closely related. When the data-fit term corresponds to a negative log-likelihood and the regularizer to a negative log-prior, the regularized point estimate can be interpreted as a maximum a posteriori estimate. Their outputs are nevertheless different: regularized point estimation returns a single description shaped by the chosen preferences, whereas probabilistic inference represents the uncertainty that remains over plausible descriptions.

1.2 Audio Systems

We now instantiate the framework for audio. Following the standard digital signal processing convention, a system, filter, or operator f maps one or more input signals to one or more output signals. Unless stated otherwise, signals are discrete-time sequences such as $x[n]$ and $y[n]$, where $n \in \mathbb{Z}$ is a discrete-time index.

The same ideas have continuous-time counterparts, but the thesis primarily uses discrete-time notation.

1.2.1 Linear Time-Invariant (LTI) Systems

A system is *linear* if it satisfies the following superposition:

$$f(a_1x_1 + a_2x_2)[n] = a_1 f(x_1)[n] + a_2 f(x_2)[n]. \quad (1.9)$$

It is *time-invariant* if delaying the input by any amount delays the output by the same amount:

$$f(x[\cdot - k])[n] = f(x)[n - k] \quad \text{for any time shift } k \in \mathbb{Z}. \quad (1.10)$$

A system that satisfies both is referred to as *linear time-invariant (LTI)*. Subsequent subsections treat *linear time-varying (LTV)* systems, which keep linearity but drop Equation (1.10), and *nonlinear* systems, which drop Equation (1.9).

An LTI system admits an *impulse response representation*: its response to a unit impulse, denoted by the (possibly infinite-length) sequence $h[n]$, fully characterizes the input–output behavior. Many distinct realizations (e.g., finite impulse response (FIR) filters, infinite impulse response (IIR) filters, sparse delay lines, and other digital-filter structures) can share the same impulse response and therefore the same input–output behavior. Under this representation, the forward map is the *discrete-time convolution*:

$$y[n] = (h * x)[n] = \sum_k h[k] x[n - k]. \quad (1.11)$$

Equivalently, the discrete-time Fourier transform (DTFT) turns convolution into pointwise multiplication,

$$h[n] * x[n] \xrightarrow{\text{DTFT}} H(e^{j\omega}) X(e^{j\omega}), \quad (1.12)$$

where $H(e^{j\omega}) = \sum_n h[n] e^{-j\omega n}$ is the system’s *frequency response*, summarizing its magnitude and phase behavior at each frequency. The duality between time-

domain convolution and frequency-domain multiplication makes filter design and analysis natural in either representation [3, 4].

Assuming a causal system with one-sided indexing $n \geq 0$, zero initial conditions, and a finite displayed truncation, convolution can equivalently be written as a matrix–vector product with a lower-triangular Toeplitz matrix:

$$\begin{bmatrix} y[0] \\ y[1] \\ y[2] \\ \vdots \end{bmatrix} = \begin{bmatrix} h[0] & 0 & 0 & \cdots \\ h[1] & h[0] & 0 & \cdots \\ h[2] & h[1] & h[0] & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ \vdots \end{bmatrix}. \quad (1.13)$$

Alternatively, we may use a parametric model f with parameters P :

$$y \approx f(x; P) = h(P) * x. \quad (1.14)$$

Many common LTI processors are described in this parametric form, including digital filters and equalizers (EQs), with P being filter coefficients, pole-zero locations, or higher-level controls such as cutoff frequency, gain, or Q value [5, 6].

The primary LTI domain considered in this thesis is room acoustics. Because h is itself a signal, an LTI system can be estimated as a signal-domain inverse problem.

1.2.2 Linear Time-Varying (LTV) Systems

When a system is linear but time-varying, i.e., its response to a fixed input changes with time, the impulse response becomes a two-dimensional kernel $h[n, k]$ indexed by observation time n and lag k . The forward map generalizes to

$$y[n] = \sum_k h[n, k] x[n - k]. \quad (1.15)$$

The LTV description is therefore substantially higher-dimensional than the one-dimensional LTI impulse response, and remains so even under framewise approximations that treat the system as locally LTI within short frames. In matrix form,

this corresponds to a general (non-Toeplitz) matrix:

$$\begin{bmatrix} y[0] \\ y[1] \\ y[2] \\ \vdots \end{bmatrix} = \begin{bmatrix} h[0,0] & 0 & 0 & \cdots \\ h[1,1] & h[1,0] & 0 & \cdots \\ h[2,2] & h[2,1] & h[2,0] & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ \vdots \end{bmatrix}. \quad (1.16)$$

The LTI system is a special case where all rows are shifted copies of a single vector. In practice, many audio systems are approximately LTI over short durations but exhibit slow time variation, e.g., a room whose door is opening, a moving source or listener, a time-varying filter, or modulation effects such as chorus or flanger [7]. A canonical example in audio is short-time Fourier transform (STFT)-based filtering [8]: the signal is decomposed into time-frequency frames $X[k, m]$ (where k is the frequency-bin index and m is the frame index), each frame is multiplied by a complex mask $M[k, m]$, and the masked frames are inverted back to the time domain. The mask varies with m , so the effective impulse response changes from frame to frame, while the operation remains linear within each frame.

1.2.3 Nonlinear Systems

A nonlinear system violates linearity (Equation (1.9)): the response depends on the amplitude and content of the input, so a single impulse response no longer characterizes the input–output behavior. The simplest case is a *memoryless nonlinearity*, applied sample by sample:

$$y[n] = \sigma(x[n]). \quad (1.17)$$

When σ admits a Maclaurin expansion [9], the output is a sum of integer-power terms:

$$y[n] = \sum_{k=0}^{\infty} \frac{\sigma^{(k)}(0)}{k!} x[n]^k. \quad (1.18)$$

Each $x[n]^k$ has a spectrum equal to the k -fold convolution of the input spectrum $X(e^{j\omega})$:

$$x^k[n] \xleftrightarrow{\text{DTFT}} \underbrace{(X * X * \dots * X)}_{k \text{ copies}}(e^{j\omega}). \quad (1.19)$$

This effect is known as *intermodulation*: products in the time domain correspond to convolutions in the frequency domain, generating harmonics and combination components from a multi-component input. Many audio effects exploit this behavior deliberately: distortion and saturation enrich the harmonic spectrum via intermodulation, while dynamic range processors such as compressors use input-dependent gain to shape level over time [10].

Systems with memory (or state) require richer descriptions. *Block-oriented* models combine linear filters and memoryless nonlinearities. For instance, the *Wiener–Hammerstein* structure cascades two linear filters around a memoryless nonlinearity [11],

$$y[n] = (h_2 * \sigma(h_1 * x))[n], \quad (1.20)$$

with the *Wiener* (h_2 absent) and *Hammerstein* (h_1 absent) structures as special cases. The *Volterra series* [12] provides a general polynomial description with memory,

$$y[n] = \sum_{p=1}^{\infty} \sum_{k_1, \dots, k_p} h_p[k_1, \dots, k_p] x[n - k_1] \cdots x[n - k_p], \quad (1.21)$$

which subsumes the block-oriented forms at the cost of high-dimensional Volterra kernels h_p . The more general the description, the wider the behavior it can represent, but at the cost of size and interpretability.

A subtler family arises when the filtering operation itself looks linear but its parameters depend on the input. A *dynamic range compressor*, for instance, applies a gain determined by the smoothed input level. Because this gain depends on the input and its recent history, the system is nonlinear and has memory [10, 13].

Because superposition does not hold, a single impulse response measurement is insufficient: nonlinear system identification generally requires observing the response across a variety of input amplitudes and content.

1.2.4 Compositional Systems

The system classes above (LTI, LTV, nonlinear) describe the input–output behavior of a single processor. This thesis treats specific LTV and nonlinear systems mainly through their *composition*: rather than identifying arbitrary LTV or nonlinear behavior in isolation, we focus on the case where many such processors are connected in a graph. A mixing engineer, for instance, chains an equalizer into a compressor into a reverb send, and a mastering chain stacks several processors in series and parallel [14, 15].

Composition is orthogonal to the LTI/LTV/nonlinear distinction: a single graph may contain processors of any of those classes. We formalize such a system as an *audio processing graph* $G = (V, E)$ with the following structure: *heterogeneous*, in that the nodes may be processors of different types (filters, dynamic range processors, modulation effects, and so on); *directed*, in that each edge points from a producing node’s output port to a consuming node’s input port, following signal flow; *acyclic* at the graph level, forming a directed acyclic graph (DAG). The most general case is a heterogeneous multigraph with cycles, but graph-level cycles create algebraic dependencies that complicate topological execution; this thesis therefore treats feedback as occurring *inside* individual processor nodes (e.g., IIR filters, recursive reverberators) while the graph-level structure remains acyclic. Where multi-channel routing requires it, two nodes may be connected by multiple parallel edges distinguished by their port or channel indices, making the graph a *multigraph*; this extension is adopted only where needed. Each node $v_i \in V$ is a processor whose categorical type t_i is part of G and whose processor parameters p_i contain settings such as gain, cutoff frequency, or threshold. The processor takes input signals and produces output signals:

$$\{y_i^{(1)}[n], \dots, y_i^{(N)}[n]\} = f_i(\{u_i^{(1)}[n], \dots, u_i^{(M)}[n]\}, p_i), \quad (1.22)$$

where M and N are the numbers of input and output channels, respectively. Here, input and output channels refer to graph inlets and outlets, not necessarily to the number of audio channels carried by each signal. Each edge $e_{ij} \in E$ carries a type attribute $t_{ij} = (k, l)$, a tuple of output and input channel indices specifying

which output port of node j connects to which input port of node i . An edge may additionally carry a parameter p_{ij} , such as a gain. When multiple edges are connected to the same input, the incoming signals are summed:

$$u_i^{(l)}[n] = \sum_{(j,k) \in \mathcal{N}^+(i,l)} y_j^{(k)}[n], \quad (1.23)$$

where $\mathcal{N}^+(i, l)$ denotes the collection of nodes and channel indices that send their output signals to the l -th input of node i .

The full system description is $H = (G, P)$, where G contains the connectivity, processor types t_i , and routing types t_{ij} , while $P = \{p_i\}_{i \in V} \cup \{p_{ij}\}_{(i,j) \in E}$ contains the node parameters and, when present, the edge parameters. These parameters are typically continuous. The forward map executes the graph in topological order, which is well-defined thanks to the graph-level acyclicity assumed above, producing the output:

$$\hat{y} = f(x; G, P). \quad (1.24)$$

When G is understood as an executable graph, we also write the same mapping as $\hat{y} = G(x; P)$.

Figure 1.2 shows an example graph. An input audio signal x , e.g., a vocal recording, enters an `[in]` node, is split by a `[crossover]` into low- and high-frequency branches, processed by separate `[distortion]` nodes, summed by `[mix]`, and sent to `[out]` to produce $\hat{y} = f(x; G, P)$. Also, a control node `[stereo_lfo]`, a low-frequency oscillator (LFO), modulates the crossover frequency of the `[crossover]` processor, illustrating how such graphs combine interpretable routing, processor selection, and parameter control.

The primary compositional domain considered in this thesis is *musical signal processing*, including audio effects, mixing, and mastering.

Crucially, composition only introduces new input–output structure when it is *non-reducible*. Cascading two LTI impulse responses yields another impulse response (their convolution $h_1 * h_2$), so for input–output behavior the graph structure carries no information beyond the equivalent single response; the graph may still be useful as an editable decomposition that exposes individual stages to a practitioner. By contrast, composing an LTI system with a nonlinear processor, or two

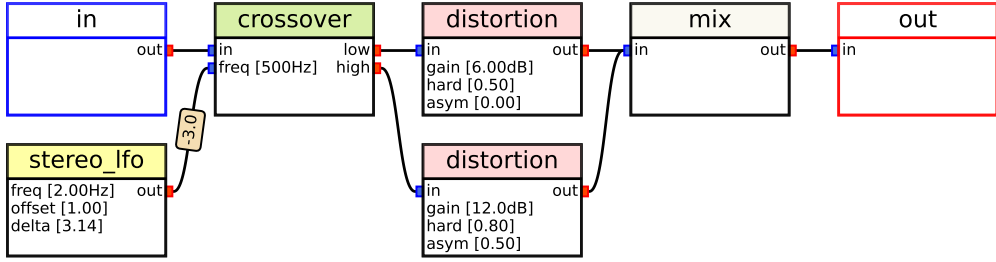


Figure 1.2: An example audio processing graph G .

nonlinear processors, produces a system whose behavior depends on the *order* and *connectivity* of the components. In such cases, the graph topology G becomes a meaningful, and necessary, target of estimation, alongside the parameters P . Estimating (G, P) in this way describes a complex system as a structured arrangement of interpretable processors, in contrast to modeling the same audio-to-audio mapping with a single black-box neural network [16, 17].

Table 1.1 consolidates the notation introduced so far for the general inverse-problem framework and its instantiations in the two audio domains we now consider.

1.3 Tasks of Interest

We now outline the two broad problem areas addressed in this thesis.

1.3.1 Room Acoustic System Estimation

Sound propagation in an enclosed space can be reasonably well modeled by the linear acoustic wave equation [18], a PDE whose closed-form solution exists only in idealized special cases (e.g., a point source radiating in free space, or a perfectly rectangular room with simple boundaries); for the irregular geometries and frequency-dependent boundaries of practical rooms, no analytical solution is generally available, and tractable proxies are required. The most common proxy is the LTI assumption: when the source and receiver positions and orientations are held fixed and the room itself is static, the point-to-point acoustic response can be rep-

Table 1.1: Notation for the inverse-problem framework and its audio instantiations.

Concept	Symbol	General	Room Acoustics	Music Production
True process	$\mathcal{F}$	Real-world process	Sound propagation	Engineer's process
Description	H	Object we estimate	RIR h	Graph + parameters (G, P)
Forward map	$f(\cdot; H)$	Applies H to input	Convolution $h * x$	Graph execution $G(x; P)$
Input	x	Source signal	Dry speech	Dry recording(s)
Output	y	Processed signal	Reverberant speech	Processed audio
Observation	$\mathcal{O}$	Available information	y or (x, y)	y or (x, y)
Inverse problem	$\mathcal{O} \rightarrow H$	Recover description	Estimate RIR	Recover graph + settings
Metric	$d(\cdot, \cdot)$	Notion of goodness	RT difference	Spectral loss

resented as an LTI system characterized by its room impulse response (RIR) $h[n]$. Estimating the RIR, or some parametric representation of it, from observed audio is a long-standing problem with applications ranging from automatic dialogue replacement (ADR) in film post-production to spatial audio rendering in virtual and augmented reality and acoustic design of performance spaces [19, 20, 21].

Depending on the application, different proxies are appropriate. An artificial reverberation model, an efficient recursive filter that emulates the perceptual effect of reverberation rather than simulating sound propagation directly, provides a compact, interpretable description whose parameters control perceptually meaningful attributes such as reverberation time (RT) and coloration, suitable when controllability is important [22, 23, 24]. The RIR itself, treated as a free-form signal, provides maximum expressivity within the LTI assumption, but it does not by itself provide compact semantic controls. A geometric acoustics model, such as acoustic radiance transfer (ART) [25, 26], instead describes how acoustic energy propagates through the space, parameterized by surface material and scattering properties, and enables *position-dependent* prediction from sparse measurements. This is a different inverse problem from estimating a single point-to-point RIR, since its target is an acoustic field over many source–receiver configurations rather than one fixed pair.

Two practical setups motivate this thesis. In ADR and foreign-language dubbing, the re-recorded take must be processed to match the on-set acoustic context; the technical core is estimating or matching the on-set reverberation from available reference audio. In spatial audio for virtual and augmented reality, the listener moves continuously through a synthetic or scanned three-dimensional environment, so a single fixed RIR is insufficient; the requirement is a forward acoustic model that can render plausible reverberation at arbitrary source–receiver positions [27, 28].

1.3.2 Audio Processing Graph Estimation

In music production, the system of interest is a compositional signal processing chain: a graph of audio processors (G, P) that transforms dry recordings into a finished mix or a processed audio signal. Unlike room acoustics, the system

description is inherently mixed: it comprises both the discrete graph topology (which processors, how they are connected) and continuous parameters (gain, EQ, compression settings).

The problem takes two forms, depending on what is observed. When both the dry sources and the mix are available, recovering the processing graph amounts to reverse engineering of the mixing decisions [29, 30, 31]: the recovered (G, P) is a model-dependent explanation of how the dry stems map to the mix, useful for remixing, remastering, educational analysis, and the construction of pseudo-labels for downstream automatic mixing research where reliable real-world graph annotations are otherwise unavailable. When only the processed output is observed, the problem becomes blind estimation of the graph, which is substantially harder, since the estimator must factor out source-dependent content while inferring processing cues; it is nonetheless relevant for analyzing finished commercial recordings whose dry stems are inaccessible.

Recovering such graphs enables several practical use cases. Educational analysis can expose plausible routings and processor hypotheses for a reference mix; large-scale empirical study becomes possible when these estimates are treated as model-dependent evidence rather than ground truth [32, 33]; and production-assistive tools can compare an in-progress mix against estimated reference graphs to surface candidate differences for human review.

1.4 Key Challenges

The inverse problems studied in this thesis are governed by the joint choice of a system description H , a forward map $f(\cdot; H)$, and an inference procedure that estimates H from the available observation $\mathcal{O}$. This choice determines not only the achievable fidelity, but also what information can be recovered, how the recovered system can be edited, and whether the method can be trained or deployed at practical scale. The following challenges therefore cut across both domains considered in this thesis: room-acoustic systems and compositional music-production systems.

- *Expressivity*: can the proxy $f(\cdot; H)$ approximate the true process $\mathcal{F}$ well? Any

behavior outside the range of $f(\cdot; H)$ over the allowed space of H remains an irreducible error of the modeling choice.

- *Controllability*: once we have an estimate $\hat{H}$, can a practitioner read it, edit it, and reuse it? This is distinct from how accurately $\hat{H}$ reproduces the observed audio: a free-form RIR is reusable for convolution but exposes no acoustic controls, while a parametric reverberator exposes controls at the cost of a smaller behavior set.
- *Optimization landscape and objectives*: even if f is expressive enough, can we actually find an $\hat{H}$ that minimizes the chosen objective $\mathcal{L}$? The landscape of $\mathcal{L}(H)$ is shaped jointly by the description, the forward map, the input/output distributions over (x, y) , and the optimization procedure itself; feedback structures, recursive parameterizations, and discrete components make it non-trivial. $\mathcal{L}$ is also a proxy for the desired notion of correctness, so low loss does not guarantee perceptual or system-level quality.
- *Data efficiency and robustness*: does the method work with the data actually available? For example, measured RIR datasets cover only a limited range of rooms, source–receiver configurations, and measurement conditions [34, 35], while large-scale datasets pairing real mixes with complete, verified processing graphs and parameters are not available. Estimators must therefore be data-efficient, use synthetic supervision, or both, and remain robust to noisy, biased, or mismatched evidence.
- *Scalability*: does the estimator remain useful as the data, computational budget, description resolution, or structural search space grows? For example, finer geometric discretization increases the number of simulation variables, longer token sequences increase generation cost, and larger processor sets expand the combinatorial search space.
- *Hardware/software compatibility*: do the available hardware, software environments, and APIs permit the desired forward model to be implemented correctly and efficiently for training and deployment? Sample-wise recursion and feedback may map poorly to graphics processing units (GPUs),

while recursive filters, discrete choices, and graph routing may not integrate directly with automatic differentiation frameworks. They can therefore require frequency-domain or batched reformulations, constrained parameterizations, surrogate gradients, or specialized sparse operations [36, 37, 38, 39], which may introduce approximation or alter the system being trained.

1.5 Methodology

Audio system inverse problems present several challenges, and this thesis organizes them around two complementary methodological perspectives: *differentiable processing*, which makes an explicit forward system amenable to gradient-based optimization, and *generative modeling*, which learns a distribution over plausible system descriptions. Figure 1.3 illustrates their distinct estimation mechanisms: panel (a) depicts direct gradient-based fitting from an input–output pair, whereas panel (b) depicts sampling in the blind setting, where y is the observed audio.

1.5.1 Differentiable Processing

Gradient-based adaptation of signal processing parameters predates modern automatic differentiation frameworks [40]. Exact and approximate gradients were developed for recursive filter structures [41, 42, 43, 44] and for neural networks whose units contained FIR or IIR filters [45, 46]. Before general-purpose automatic differentiation, these methods typically required a backward computation derived for the particular processor or architecture.

General-purpose automatic differentiation frameworks removed this requirement. Once a processing operation is written as a composition of differentiable primitives, its derivatives are obtained automatically and compose with those of any connected neural network. The differentiable digital signal processing (DDSP) framework [47] applied this to sound synthesis, pairing a neural network with a harmonic-plus-noise synthesizer based on spectral modeling synthesis [48]. The network predicted the synthesizer controls, and an audio-domain loss trained both parts together end to end. The explicit synthesizer supplied domain structure and interpretable controls, while the network learned how those controls varied with

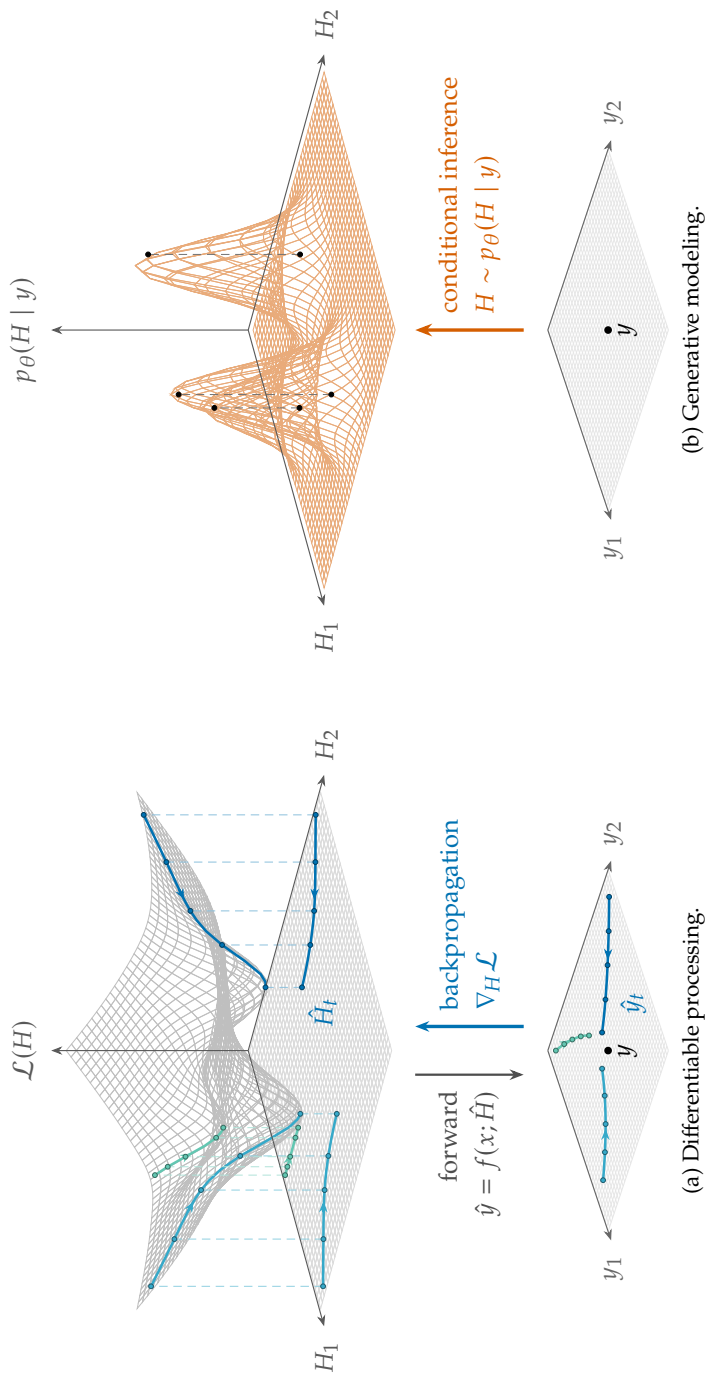


Figure 1.3: **Two key methods for audio system estimation.** (a) **Differentiable processing** optimizes an explicit system through a forward model and backpropagation, with different initializations reaching distinct local minima. (b) **Generative modeling** samples plausible systems from a potentially multimodal conditional distribution.

the input, placing the combination between fixed parametric modeling and unconstrained neural synthesis [47, 49].

Subsequent literature uses *differentiable processing* as an umbrella term for methods that make signal processing operations available to gradient-based optimization [49]. Processors composed of differentiable primitives can use automatic differentiation directly [47]. Recursive or sequential processors may instead use backpropagation through time [38], frequency-sampled nonrecursive approximations [36, 37, 6], or analytically derived backward passes [50]. Constrained parameterizations can maintain stability or other validity conditions during optimization [6, 51]. Discrete or non-smooth operations can use continuous relaxations or straight-through gradients [52, 53, 39]. When an ordinary derivative provides a poor search direction, a surrogate computation can supply the backward signal, as in delay estimation [54]. For black-box processors, gradients can be estimated numerically, including through finite differences or simultaneous perturbation stochastic approximation [55, 56, 57], or supplied by a neural proxy [57]. These routes may differentiate the original processor, an approximation, or a proxy, trading forward fidelity, backward-signal quality, and computational cost.

A usable backward signal does not by itself provide an effective search direction. The optimization landscape depends jointly on the processor, its parameterization, the metric, and the compared signals. Sinusoidal-frequency estimation under a spectral loss is a standard example, with proposed remedies including a complex-exponential surrogate for the oscillator [58] and redesigned spectral objectives [59, 60]. Expressivity, computational differentiability, and optimizability are therefore distinct properties.

For system inversion, let $\mathcal{L} = d(y, \hat{y})$ be the audio-domain loss, where $\hat{y} = f(x; H)$ is the output audio. The relevant backward path runs from this loss to the continuously parameterized variables in the system description H being estimated. Differentiability with respect to the input x alone is insufficient. For amortized estimation, a neural estimator g_θ with parameters θ maps the available

observation $\mathcal{O}$ to $H = g_\theta(\mathcal{O})$. The forward and backward computations are

$$\mathcal{O} \xrightarrow{g_\theta} H \xrightarrow{f(x; \cdot)} \hat{y} \xrightarrow{d(y, \cdot)} \mathcal{L}, \quad (1.25a)$$

$$\nabla_\theta \mathcal{L} \xleftarrow{(\partial H / \partial \theta)^\top} \nabla_H \mathcal{L} \xleftarrow{(\partial \hat{y} / \partial H)^\top} \nabla_{\hat{y}} \mathcal{L} \xleftarrow{(\partial \mathcal{L} / \partial \hat{y})^\top} 1. \quad (1.25b)$$

Direct optimization treats H itself as the optimization variable, whereas amortized estimation propagates the gradient through g_θ to θ .

For direct optimization, let p_{obs} denote the distribution of input–output observations governed by the same unknown system. A basic data-fit objective is

$$\hat{H} \in \arg \min_H \mathbb{E}_{(x, y) \sim p_{\text{obs}}} [d(y, f(x; H))]. \quad (1.26)$$

This objective is solved anew for each unknown system using the $H \rightarrow \hat{y} \rightarrow \mathcal{L}$ portion of the backward path. Figure 1.3(a) illustrates this direct formulation as local search over the description space. The objective defines the best-fitting target, but gradient descent may follow different trajectories from different initializations and converge to distinct local minima, including descriptions that produce similar output audio. It does not guarantee a globally preferred solution.

A neural estimator instead amortizes estimation across a training distribution p_{train} over observations, inputs, and outputs. Its shared parameters are learned by

$$\hat{\theta} \in \arg \min_\theta \mathbb{E}_{(\mathcal{O}, x, y) \sim p_{\text{train}}} [d(y, f(x; g_\theta(\mathcal{O})))] . \quad (1.27)$$

The estimator input $\mathcal{O}$ need not contain every variable used to evaluate the training objective. In blind estimation, for example, the estimator may receive only $\mathcal{O} = y$, while the corresponding input x is available during training to render the predicted system. Once trained, g_θ predicts H without optimizing it separately for each observation. This amortizes rather than eliminates optimization: the instance-specific search over H is replaced by training the shared parameters θ .

Both objectives supervise the description through the output audio. Direct optimization therefore requires no ground-truth H , and the neural estimator can likewise be trained from input–output examples without description labels. As written, both are point-estimation formulations: they select a description rather

than represent a distribution over alternative descriptions. Once the description space and forward map are fixed, differentiable processing searches only within the chosen proxy; it neither expands that proxy’s representational range nor resolves ambiguity between descriptions that the available observations cannot distinguish.

1.5.2 Generative Models

A *generative model* learns a sampling procedure whose outputs follow a probability distribution over a data space. In this thesis, that data space consists of audio system descriptions, such as RIR representations and processing graphs, rather than output audio signals themselves. Writing all randomness used by the procedure as Z , a sample of the system description can be viewed generally as the pushforward of a simple base distribution p_Z through a learned map T_θ :

$$H = T_\theta(Z), \quad Z \sim p_Z, \quad p_H = (T_\theta)_\# p_Z. \quad (1.28)$$

This view only states how the learned distribution is induced; it does not require either H or T_θ to be continuous. A variational autoencoder realizes T_θ as its decoder [61], a normalizing flow as a composition of invertible transformations [62, 63], and a generative adversarial network as its generator [64]. Diffusion models [65, 66, 67] and flow-matching models [68] instead obtain the sample at the end of a learned transport from base noise.

When a system description is discrete, or can be encoded as a sequence of discrete tokens $H = (h_1, \dots, h_T)$, its joint distribution can be modeled autoregressively [69, 70]:

$$p(H) = \prod_{t=1}^T p(h_t \mid h_{<t}). \quad (1.29)$$

Applying this factorization requires both a discrete representation and an order in which its tokens are generated. A sequence supplies that order directly, whereas a set or graph requires a chosen serialization or generation order. This thesis applies the construction to quantized RIR tokens and to serialized node and edge tokens of audio processing graphs. For system inverse problems, the distribution

is conditioned on the observation, yielding $p(H \mid \mathcal{O})$ and, in the autoregressive case, factors $p(h_t \mid h_{<t}, \mathcal{O})$. The models considered here learn this conditional distribution directly from paired or synthetically generated data.

Figure 1.3(b) represents this conditional distribution as potentially multimodal: its samples can occupy distinct plausible modes rather than being tied to the basin of a particular per-instance initialization. This removes per-instance optimization and can represent ambiguity through multiple samples, but its coverage is limited by the training distribution, the chosen representation of H , and the training objective.

Taken together, these perspectives show that the choice of proxy and the choice of inference procedure are conceptually distinct but interdependent in practice. The description space of H and the forward map f determine what behavior can be represented and how the resulting system can be controlled, while the requirements of inference determine which proxy choices are practically usable. Differentiable processing requires a forward computation with tractable and informative gradients, which may restrict the proxy or motivate a differentiable reformulation. Generative modeling instead requires the description to admit a learnable representation and the training data to sufficiently cover its distribution. Once a proxy is fixed, however, both approaches inherit the same representational ceiling; their differences then concern how plausible descriptions within that proxy are estimated or represented.

1.5.3 Hybrid Approaches

The two perspectives are not exclusive, since a description can be factorized and each part assigned to whichever mechanism matches its structure. Where a well-established domain structure exists, a differentiable forward model represents that part compactly and keeps its parameters inspectable; where the structure is discrete or not available in closed form, a generative model supplies it. Many combinations follow from this idea, and this thesis explores a few of them.

1.6 Contributions and Thesis Organization

This thesis organizes its contributions along two axes: the *type of target system* (LTI vs. compositional) and the *methodology* (differentiable processing vs. generative modeling). Table 1.2 summarizes their organization.

1.6.1 Chapter-by-Chapter Contributions

Differentiable Artificial Reverberation (Chapter 2) — Artificial reverberation comprises a long lineage of compact digital-filter structures that mimic reverberant responses through parameters controlling decay, coloration, and echo density [71, 72, 73]. The inverse problem here is to infer such parameters from a reference RIR or a reverberant speech signal; earlier reverberator-specific analysis and parameter-matching methods were designed for particular reverberator structures [74, 75, 76], while learning-based estimators were likewise tied to a single reverberator and trained against its parameter targets [77].

In contrast, differentiable artificial reverberation (DAR) provides a unified approach for converting several reverberator structures into differentiable counterparts and closes the loop end-to-end: a neural network predicts the parameters, the differentiable reverberator generates the corresponding impulse response, and the loss is computed on that impulse response. The main obstacle is computational, since many reverberators rely on IIR filters or feedback structures whose sequential evaluation is inefficient on GPUs; building on differentiable processing [47] and differentiable filter design [37], the chapter replaces these recursive components with FIR approximations via the frequency-sampling method, and systematically analyzes the parallelization of recursive filters in this framework. After training, the estimated parameters can be used with the original reverberators rather than their FIR training surrogates. Experiments show consistent improvements over non-end-to-end baselines in objective measures and listening tests, with the frequency-sampling approximation shown to be acceptable for the training objective.

Differentiable Acoustic Radiance Transfer (Chapter 3) — The next shift is from single source–receiver responses to acoustic field learning under geometric acous-

Table 1.2: Thesis summary and organization.

	Linear Time-Invariant <i>Impulse response</i> Room acoustics (ADR, virtual/augmented reality)	Compositional <i>Audio processing graph</i> Musical (audio effects, mixing)
Differentiable Processing	Differentiable Artificial Reverberation (Chapter 2) Differentiable Acoustic Radiance Transfer (Chapter 3)	Differentiable Audio Processing Graphs (Chapter 5) Reverse Engineering of Mixing Graphs (Chapter 6)
Generative & Hybrid	Autoregressive Modeling for RIR Estimation (Chapter 4)	Blind Estimation of Audio Processing Graphs (Chapter 7)

tics: the target is to predict acoustic responses for unseen source–receiver configurations from sparse measurements and an approximate room geometry.

The technical basis is the acoustic rendering equation [25], an acoustic analogue of the rendering equation in graphics [78]; ART solves it by discretizing time- and direction-dependent energy exchange between surface patches, with material or scattering parameters governing how incident energy is redistributed over outgoing directions. The image source method [79] and recent differentiable image source models [28] provide a different geometric acoustics model, while neural acoustic field models [27, 80] provide data-driven alternatives. Differentiable acoustic radiance transfer (DART) [81] instead retains an explicit physical forward model and makes it optimizable with flexible learnable material properties; kernel decomposition, frequency-domain computation, and sparse storage make end-to-end optimization computationally feasible. On sparse-measurement scenarios, this physically structured model generalizes better than interpolation, signal processing, and neural network baselines for energy response prediction, including under moderate geometric distortion, while remaining simple and interpretable.

Autoregressive Modeling of Neural Discrete Representations for Room Impulse Response Estimation (Chapter 4) — The RIR estimation thread then returns from spatial acoustic fields to single-response RIR estimation, focusing on the generator inside neural RIR estimators. This chapter starts from the observation that a typical estimator pairs a reference encoder with a generator that synthesizes the implied RIR, and the generator largely determines which temporal, spectral, and perceptual structures are reachable. Existing generators include filtered-noise models [82], differentiable artificial reverberators [83], and convolutional decoders [84]. This chapter instead learns a discrete time-frequency representation of RIRs and casts estimation as conditional token generation, adapting the body of work on discrete audio-token generation [85, 86, 87] to the system estimation setting [88].

With residual quantization, an autoencoder encodes each local patch in time and frequency as a sequence of discrete codebook indices, where later indices refine earlier ones; the resulting RIR is a three-axis token tensor over frequency, time, and quantization depth. Conditioned on reference audio, a transformer then

predicts these tokens autoregressively, and the trained autoencoder decoder reconstructs the RIR. Recent autoregressive modeling techniques that operate over sequence and depth [86, 89, 87] are extended to also handle the frequency axis, and their performance and computational tradeoffs are explored. On blind RIR estimation and acoustic matching, the proposed generator is generally preferred to the baselines under both objective and subjective criteria, with ablations indicating that both discretization and autoregressive modeling are important.

A hybrid extension further pairs the discrete generator with a differentiable filtered-noise decoder for late reverberation to reduce the token budget and improve inference on a central processing unit (CPU), illustrating how generative and differentiable components can be combined when each addresses a different part of the system structure.

Differentiable and Efficient Computation of Audio Processing Graphs (Chapter 5) — The compositional half begins with the forward model itself: how an audio processing graph is represented and how it is computed differentiably. Prior differentiable audio frameworks supply individual processors or a fixed processing chain [47, 90], so the routing is decided in advance and stays outside the computation. This chapter instead represents the system as a heterogeneous graph of multi-input multi-output (MIMO) differentiable processors, where the topology is part of the description. Evaluating such a graph node by node underuses parallel hardware, since the nodes are of different types and the topology may change between optimization steps; the chapter therefore renders it by grouping same-type processors for parallel execution and batching distinct graphs together. The implementation is released as an open-source PyTorch library [91].

Reverse Engineering of Music Mixing Graphs with Differentiable Processors and Iterative Pruning (Chapter 6) — With the renderer in place, the next problem is reverse engineering of music mixes: given dry source tracks and a target mix, the goal is to infer a graph of audio processors and parameters that maps the dry sources to a close match of the mix [54, 92]. The chapter extends prior reverse engineering work from linear approximations [29] and fixed processor chains [30, 93] to the compositional structure of mixing, where different tracks and subgroups may pass through different subsets of processors before being combined; many

graphs can produce similar audio, so the method imposes a structured search space rather than claiming unrestricted recovery.

The differentiable mixing console [14] is the closest mixing-domain predecessor, but it does not search for sparse graph structure. The method begins with an over-complete mixing console (every available processor placed on every track and subgroup), whose continuous parameters are optimized by gradient descent against an audio-domain matching loss. The graph is then simplified by iterative pruning [94]: candidates whose removal has negligible effect are dropped, the remaining parameters are fine-tuned, and the process repeats until a chosen sparsity–quality tradeoff is reached, with batched graph execution and dry/wet controls used as approximate importance scores to keep the cost manageable.

With suitable tolerance, the experiments show that pruning can remove roughly two-thirds of the processors while preserving the quality of the full console, and the resulting graphs support both individual mix analysis and large-scale graph-data collection.

Blind Estimation of Audio Processing Graphs (Chapter 7) — The final compositional chapter removes the dry input and addresses blind estimation of audio processing graphs from reference audio alone. Unlike Chapter 6, the dry source signal is unavailable, so the model must infer processing cues while ignoring source-dependent content.

General graph generation [95] and TokenGT [96] provide the basis for token-based graph decoding. Prior audio reverse engineering methods assume dry inputs or simpler processor chains [30, 97], whereas this chapter operates on processed audio alone. Since no large public corpus pairs processing graphs with reference audio, the model is trained on a synthetic graph-reference dataset. A convolutional reference encoder maps the audio to a latent representation, and a transformer-based graph decoder first reconstructs the connectivity and categorical node and edge types autoregressively; continuous node and edge parameters are then assigned.

Compared with prior reverse engineering work, this is an early formulation of audio processing graph estimation as a machine learning problem and a generative modeling approach to audio system inverse problems, with a systematic

investigation of autoregressive decoding strategies for graph token sequences. On singing voice effects and drum mixing tasks, exact graph recovery remains difficult when evaluated on source audio outside the training distribution, but the model recovers important local and global routing patterns including multi-band processing and sidechaining, and the rendered audio can remain perceptually similar to the reference. A differentiable-rendering extension further refines the continuous parameters against the reference audio, improving audio-domain metrics while leaving the discrete topology fixed.

1.6.2 Broader Impact

The framing of system estimation in terms of differentiable processing, generative modeling, or combinations of the two may also apply to other audio systems and, more broadly, to mechanical and other physical systems. Examples include machinery condition monitoring, where structural impulse responses replace RIRs and stiffness or damping coefficients replace surface material properties, and sensor calibration, where the estimated description is a transducer response or a microphone array geometry.

1.6.3 Publications

The following publications form the basis of this thesis:

1. S. Lee, H.-S. Choi, and K. Lee, "Differentiable Artificial Reverberation," *IEEE/ACM Transactions on Audio, Speech, and Language Processing (TASLP)*, 2022.
2. S. Lee, M. Scerbo, S. Han, M.-J. Choi, K. Lee, and E. De Sena, "Differentiable Acoustic Radiance Transfer," *IEEE Transactions on Audio, Speech, and Language Processing (TASLPRO)*, 2026.
3. S. Lee, H.-S. Choi, and K. Lee, "Yet Another Generative Model for Room Impulse Response Estimation," *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2023.

4. S. Lee, M. A. Martínez-Ramírez, W.-H. Liao, S. Uhlich, G. Fabbro, K. Lee, and Y. Mitsufuji, "GRAFX: An Open-Source Library for Audio Processing Graphs in PyTorch," *Proceedings of the International Conference on Digital Audio Effects (DAFx, demo)*, 2024.
5. S. Lee, M. A. Martínez-Ramírez, W.-H. Liao, S. Uhlich, G. Fabbro, K. Lee, and Y. Mitsufuji, "Searching for Music Mixing Graphs: A Pruning Approach," *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, 2024.
6. 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," *Journal of the Audio Engineering Society (JAES)*, 2025.
7. S. Lee, J. Park, S. Paik, and K. Lee, "Blind Estimation of Audio Processing Graph," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.

Chapter 2

Differentiable Artificial Reverberation

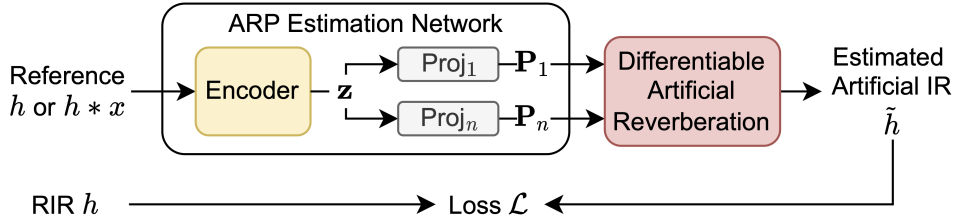
2.1 Introduction

Artificial reverberation (AR) models are efficient digital filters that approximate reverberant responses with compact and controllable parameters, such as decay, coloration, and echo density [22, 23, 24]. AR models have been applied to room acoustic enhancement [98], auditory scene generation [99, 100], post-production [77], and many more.

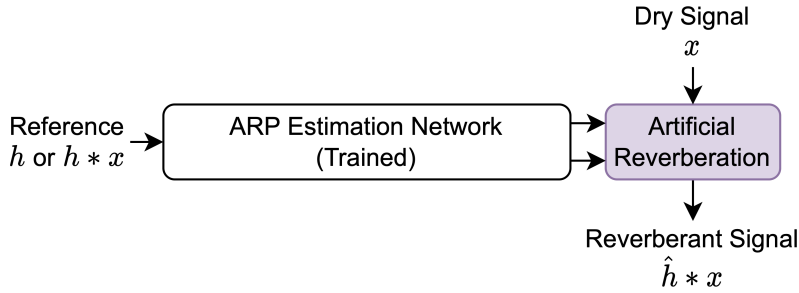
Nevertheless, estimating the AR model parameters (ARPs) that match a reference reverberation remains challenging: the forward mapping from ARPs to generated reverberation is highly nontrivial, and the inverse mapping from a reference reverberation back to ARPs is consequently difficult and often non-unique. We refer to this task as *analysis-synthesis* when the reference is a room impulse response (room IR, RIR). When the reference is indirectly provided with a reverberant signal, i.e., an RIR convolved with a dry signal, we refer to this task as *blind estimation*. We limit the scope to reverberant speech signals in this chapter.

ARP estimation is a classical problem, and various attempts have been made [74, 75, 73, 76]. However, they are often tied to a specific AR model (AR-model-dependent) or to a specific reference form (task-dependent). This inflexibility is problematic since different applications may require different AR models and different observations, such as RIRs or reverberant speech.

We address these limitations with *differentiable artificial reverberation* (DAR)



(a) Training ARP estimation network with DAR models. The network predicts ARP groups $P_1, \dots, P_n$, the DAR model renders an IR, and the loss gradients $\partial \mathcal{L} / \partial P_1, \dots, \partial \mathcal{L} / \partial P_n$ are back-propagated through the DAR model. The same reference encoder can be used for analysis-synthesis (RIR h input), blind estimation (reverberant speech $h * x$ input), or both, while only the projection layers differ per DAR model.



(b) Inference-time usage of the trained ARP estimation network. Each DAR model is used as a differentiable training surrogate for its original AR model. After training, estimated ARPs can be plugged into the AR model, which is highly efficient and real-time controllable in CPU.

Figure 2.1: **The proposed ARP estimation framework.**

models. Each DAR model takes ARPs as input and generates its corresponding AR model’s IR (see Figure 2.1a). After computing a loss between the generated IR and the reference RIR, the loss gradients can be back-propagated through the DAR model to the estimated ARPs. As a result, an ARP estimation network can be trained end-to-end with a loss on the generated IR, rather than a manually weighted parameter-matching loss. The same reference encoder can be reused across tasks and DAR models; only the lightweight projection layers differ per DAR model.

Since the AR models are exactly or close to linear time-invariant (LTI), we can in principle implement them differentiably “as is” with their difference equations [44, 45, 46, 38]. However, recurrent sample generation of their infinite impulse response (IIR) or near-LTI recursive components limits the training speed when

the model is executed on a parallel processor, such as a GPU. To address this bottleneck, we replace these recursive components with finite impulse response (FIR) approximations using the frequency-sampling method [36], following differentiable filter design work [37, 6]. This way, each DAR model becomes an FIR-based training surrogate whose IR can be generated without any recurrent step. Using this technique, we present differentiable *Filtered Velvet Noise* (FVN) [73], *Advanced Filtered Velvet Noise* (AFVN), and *Delay Network* (DN) [101, 102]. The DAR models are not identical to their originals, but the approximation errors are small enough for reliable end-to-end training; the experiments below also show that the optimization benefit outweighs the surrogate error when compared with non-end-to-end baselines using the same AR structures. After the training, we can use the estimated ARPs with the original AR models for efficient computation (see Figure 2.1b).

2.2 Related Works

2.2.1 Differentiable Digital Signal Processing

Differentiable digital signal processing (DDSP) imports structured DSP components into deep learning frameworks, allowing their parameters to be optimized through audio-domain losses while retaining useful inductive biases and interpretable controls [47]. In the case of LTI filters, both FIR and IIR filters are differentiable and can be implemented as convolutional and recurrent layers, respectively [44, 45, 46, 38]. Since recurrent IIR evaluation can bottleneck batched training, frequency-sampling has been used to approximate IIR filters with FIR filters in differentiable parametric equalizer design [36, 37, 6]. We use the same principle to implement complete AR models as differentiable FIR-based training surrogates, and we analyze the reliability of this approximation for ARP estimation.

2.2.2 ARP Estimation

Analysis-synthesis — Various methods have been proposed to systematically and automatically estimate ARPs that mimic certain characteristics of a given RIR.

Some works first extract perceptually relevant features from the RIR, then estimate the ARPs from them. For example, feedback delay network (FDN) parameters were derived from the RIR’s energy decay relief (EDR) representation which encodes the frequency-dependent reverberation decay [74]. Along with the FVN and AFVN, a parameter estimation algorithm based on linear prediction was proposed [73]. Another thread of work utilizes optimization. A genetic algorithm was proposed to find the FDN parameters that achieve the desired “fitness” to the reference RIR [75]. Since these methods assume availability of the RIR, they cannot be directly applied to the blind estimation task.

Blind Estimation — End-users (e.g., audio engineers) often interact with AR models through audio plug-ins. As a result, several attempts have been made to implement blind estimation algorithms for plug-ins. For example, a preset recommendation method based on the Gaussian mixture model was proposed [76]. DNNs were also trained for plug-in parameter estimation and preset recommendation [77].

While these methods have shown promise, there is still room for improvement. First, they generated the training data using the plug-ins to train their models. However, this procedure may be time-consuming, and relying on data generated with a single plug-in may increase the generalization error. Second, their training objectives could be suboptimal. When the task is defined as parameter regression [77], one must weigh each parameter’s perceptual importance and apply it to the loss function design, which is a highly challenging task. When the task is defined as classification [76, 77], it could suffer from scalability problems. Finally, most plug-ins provide only a few parameters, and each of them controls multiple internal ARPs simultaneously. This might limit the performance due to the reduced degree of freedom.

DAR models tackle these problems by replacing the parameter-matching objective with an end-to-end loss on the generated IR. Since the loss directly compares the estimated IR with the ground-truth RIR, we can use RIRs for training without obtaining ground-truth ARPs. We also estimate internal ARPs directly instead of relying on the smaller set of plug-in parameters.

Two-Stage Approaches — Note that both analysis-synthesis and blind estimation

tasks can be divided into two subtasks. First, we can obtain reverberation parameters, e.g., reverberation time, from a given reference (reference-to-reverberation parameters). We can directly compute the parameters from the RIR [103] for the analysis-synthesis case. For the blind estimation, various methods have been proposed [104, 105, 106, 107]. Then, we can tune the ARPs to match the desired reverberation parameter values (reverberation parameters-to-ARPs) [108, 109, 110]. This two-stage approach can be a possible alternative to our single-stage end-to-end estimation method. However, perceptually different reverberations can have the same reverberation parameters, resulting in information loss.

2.2.3 Modeling Reverberation in the Deep Learning Framework

An alternative to DAR is to generate an RIR directly using a learned model. A high-order FIR can be used directly as an RIR model [47], and neural reverberators can be controlled through conditioning methods [14, 111] or paired with a differentiable RIR generator and a parameter estimator [82]. DAR instead keeps the estimated description inside a classical AR model. This restricts the model to the chosen AR structure, but the resulting parameters are interpretable and controllable, the structure helps prevent unexpected artifacts, and the original sparse IIR/FIR implementation remains efficient for CPU rendering.

2.3 Frequency-Sampling Method for Differentiable IIR Filters

For any real IIR filter H , its frequency response $H(e^{j\omega})$ is continuous, 2π -periodic, and conjugate symmetric. We *frequency-sample* $H(e^{j\omega})$ at angular frequencies $\omega_k = 2\pi k/N$, $k = 0, \dots, \lfloor N/2 \rfloor$, to obtain its FIR approximation [36]. We denote the frequency-sampled filter with subscript N , i.e., H_N .

$$H_N[k] = H(e^{j\omega_k}). \quad (2.1)$$

Note that the frequency-sampling can also be defined on transfer functions, e.g., $H(z)$ with $z = e^{j\omega_k}$.

Since each sampling is independent and commutes with basic arithmetic operations, we can frequency-sample an IIR filter by combining frequency-sampled m -sample delays $(z^{-m})_N \in \mathbb{C}^{\lfloor N/2 \rfloor + 1}$ as follows,

$$H_N[k] = \left(\frac{\sum_{m=0}^M \beta_m z^{-m}}{\sum_{m=0}^M \alpha_m z^{-m}} \right)_N [k] = \frac{\sum_{m=0}^M \beta_m (z^{-m})_N [k]}{\sum_{m=0}^M \alpha_m (z^{-m})_N [k]}. \quad (2.2)$$

Since $(z^{-m})_N[k] = e^{-j2\pi km/N}$, using the widely used deep learning libraries, we obtain $(z^{-m})_N$ as a tensor z_m by

$$\text{angle} = 2\pi \cdot \text{arange}(N//2+1)/N, \quad (2.3a)$$

$$z_m = \exp(-1j \cdot \text{angle} \cdot m). \quad (2.3b)$$

Then, we apply equation (2.2) to $z_0, \dots, z_M$ with basic tensor operations and obtain every sample of H_N in parallel. A time-domain representation of H_N is a length- N FIR $h_N[n]$, inverse discrete Fourier transform (inverse DFT) of $H_N[k]$, i.e.,

$$h_N[n] = \frac{1}{N} \sum_{k=0}^{N-1} H_N[k] e^{j\omega_k n} (u[n] - u[n - N]) \quad (2.4)$$

where $u[n]$ is a unit step function.

Therefore, using the frequency-sampling method, we can replace any IIR filter H with an FIR $h_N[n]$ whose convolution is efficiently computed via the fast Fourier transform (FFT). In this chapter, the “differentiable IIR” filter refers to H_N , which is in fact an FIR filter.

2.3.1 Reliability of the Frequency-Sampling Method

Every IIR filter’s frequency response has “infinite bandwidth” (in the time domain) so that frequency-sampling causes *time aliasing* [4]. Specifically, the frequency-sampled IR $h_N[n]$ is the sum of length- N segments of the original IR $h[n]$.

$$h_N[n] = \sum_{m=0}^{\infty} h[mN + n] (u[n] - u[n - N]). \quad (2.5)$$

Since the energy of every stable IIR filter's IR decays over time, more frequency-sampling points N give less time aliasing, i.e., a closer approximation (see Figure 2.2). Therefore, we can use the differentiable filter H_N in place of the original H to match a reference response H_{Ref} . We elaborate this with the following analytical results.

Time Aliasing Error — If H has M distinct poles where each of them $\nu_i \in \mathbb{C}$ has multiplicity $r_i \in \mathbb{N}$, the time aliasing error asymptotically decreases as N increases as follows,

$$\|H - H_N\|_2 = \sum_{i=1}^M O(N^{r_i-1} |\nu_i|^N). \quad (2.6)$$

Response and Loss Errors — For the response-matching error itself, the reverse triangle inequality gives a direct bound in terms of the time aliasing error:

$$|\|H_{\text{Ref}} - H\|_2 - \|H_{\text{Ref}} - H_N\|_2| \leq \|H - H_N\|_2. \quad (2.7)$$

If the training objective is the squared ℓ_2 loss, the corresponding loss error is bounded by factorizing the difference of squared norms and applying the Cauchy-Schwarz inequality:

$$|\|H_{\text{Ref}} - H\|_2^2 - \|H_{\text{Ref}} - H_N\|_2^2| \leq \|H - H_N\|_2 (\|H_{\text{Ref}} - H\|_2 + \|H_{\text{Ref}} - H_N\|_2). \quad (2.8)$$

Loss Gradient Error — The loss gradient error has similar asymptotic behavior to the time aliasing error as follows,

$$\left| \frac{\partial}{\partial p} \|H_{\text{Ref}} - H\|_2^2 - \frac{\partial}{\partial p} \|H_{\text{Ref}} - H_N\|_2^2 \right| = \sum_{i=1}^M O(N^{3r_i-1} |\nu_i|^N) \quad (2.9)$$

where p is a parameter of H . See Appendix 2.C for the proofs.

2.3.2 Differentiable Dense IIR Filter with State Variable Filters

Since an arbitrary IIR filter can be expressed as serially cascaded biquads (second-order IIR filters), we can obtain a differentiable IIR filter $\tilde{H}$ as a product of

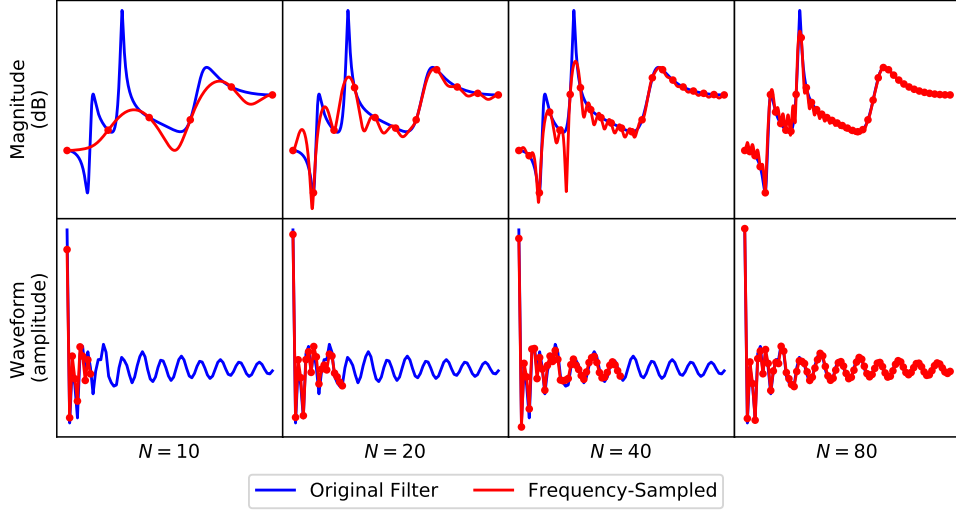


Figure 2.2: **The frequency-sampling method with a various number of sampling points N .** Blue curves represent its magnitude response $|H(e^{j\omega})|$ (top row) and IR $h[n]$ (bottom row). Red curves are the magnitude response and IR of the frequency-sampled filter H_N .

frequency-sampled biquads $(H_i)_N$.

$$\tilde{H}[k] = \prod_i (H_i)_N[k] = \prod_i \left(\frac{\sum_{m=0}^2 \beta_{i,m} z^{-m}}{\sum_{m=0}^2 \alpha_{i,m} z^{-m}} \right)_N [k]. \quad (2.10)$$

Moreover, we use the *state variable filter* (SVF) parameters f_i , R_i , m_i^{LP} , m_i^{BP} , and m_i^{HP} to express each biquad as follows,

$$\beta_{i,0} = f_i^2 m_i^{\text{LP}} + f_i m_i^{\text{BP}} + m_i^{\text{HP}}, \quad (2.11a)$$

$$\beta_{i,1} = 2f_i^2 m_i^{\text{LP}} - 2m_i^{\text{HP}}, \quad (2.11b)$$

$$\beta_{i,2} = f_i^2 m_i^{\text{LP}} - f_i m_i^{\text{BP}} + m_i^{\text{HP}}, \quad (2.11c)$$

$$\alpha_{i,0} = f_i^2 + 2R_i f_i + 1, \quad (2.11d)$$

$$\alpha_{i,1} = 2f_i^2 - 2, \quad (2.11e)$$

$$\alpha_{i,2} = f_i^2 - 2R_i f_i + 1. \quad (2.11f)$$

We prefer to use the SVF parameters rather than the biquad coefficients for the

following reasons.

- *Interpretability.* Each SVF has lowpass H_i^{LP} , bandpass H_i^{BP} , and highpass filter H_i^{HP} . They share a resonance R_i and cutoff parameter $f_i = \tan(\pi\omega_i/\omega_s)$ where ω_i is their cutoff frequency and ω_s is the sampling rate. The filter outputs are multiplied with gains m_i^{LP} , m_i^{BP} , and m_i^{HP} and then summed. This interpretability comes without any generality loss since an SVF can express any biquad [112].
- *Simple Activation Functions.* The SVF parameters have simpler stability conditions, $R_i > 0$ and $f_i > 0$, than the biquad coefficients [38], which leads to simpler activation function design. Refer to Section 2.5.3 for details.
- *Better Performance.* We found that our ARP estimation networks performed better when they estimate the SVF parameters rather than the biquad coefficients. Refer to Appendix 2.A.2 for the comparison results.

From now on, we denote each SVF-parameterized biquad with a superscript H_i^{SVF} and call it simply “SVF”.

2.3.3 Differentiable Parametric Equalizer

A low-shelving H^{LS} , peaking H^{Peak} , and high-shelving filter H^{HS} are widely used audio filters. Each of them can be obtained using the SVF as

$$H^{\text{LS}}(f, R, G) = H^{\text{SVF}}(f, R, G, 2R\sqrt{G}, 1), \quad (2.12a)$$

$$H^{\text{Peak}}(f, R, G) = H^{\text{SVF}}(f, R, 1, 2RG, 1), \quad (2.12b)$$

$$H^{\text{HS}}(f, R, G) = H^{\text{SVF}}(f, R, 1, 2R\sqrt{G}, G) \quad (2.12c)$$

where the SVF parameters are given as $H^{\text{SVF}}(f, R, m^{\text{LP}}, m^{\text{BP}}, m^{\text{HP}})$ [5]. Here, G is a new parameter that gives $20 \log_{10}(G)$ dB gain. A parametric equalizer (PEQ) is a serial composition of such filters. Following [37, 6], we use one low-shelving, one high-shelving, and $K - 2$ peaking filters.

$$H^{\text{PEQ}}(z) = H^{\text{LS}}(z)H^{\text{HS}}(z) \prod_{i=1}^{K-2} H_i^{\text{Peak}}(z). \quad (2.13)$$

Same as the IIR filter case, frequency-sampling the components and multiplying them results in a differentiable PEQ H_N^{PEQ} .

2.4 Differentiable Artificial Reverberation

In this section, we derive differentiable FVN, AFVN, and DN. We first briefly review the original AR models. Then, we obtain their DAR counterparts by frequency-sampling the IIR components, replacing them with FIRs. We also slightly modify the original models for efficient training, plausible reverberation, and better overall estimation performance.

2.4.1 Differentiable Filtered Velvet Noise

Filtered Velvet Noise — One can divide an RIR $h[n]$ into S segments (see Figure 2.3). As reverberation can be regarded as mostly stochastic, we can model each length- L_i segment $h_i[n]$ with a source noise signal $s_i[n]$ “colored with” a filter C_i . When we use *velvet noise* for each source, this source-filter model becomes *Filtered Velvet Noise* (FVN) [113, 114, 73]. The velvet noise $v_i[n]$ is sparse; in every length- T_i interval, it contains a single nonzero sample ± 1 with random sign/position such that its time-domain convolution is highly efficient. Moreover, an allpass filter U_i is introduced to smooth each source and make the output sound plausible. Finally, a deterministic (bypass) FIR $\hat{h}_0[n]$ models the direct arrival and early reflections. Then, an IR of FVN $\hat{h}[n]$ becomes

$$\hat{h}[n] = \hat{h}_0[n] + \sum_{i=1}^S \underbrace{\overbrace{v_i[n - d_i] * u_i[n]}^{s_i[n - d_i]} * c_i[n]}_{\hat{h}_i[n - d_i]} \quad (2.14)$$

where each d_i is a delay for the segment alignment ($d_1 = 0$). In practice, each allpass filter U_i is composed with Schroeder allpass filters (SAPs),

$$U_{i,j}^{\text{SAP}}(z) = \frac{1 + \gamma_{i,j} z^{-\tau_{i,j}}}{\gamma_{i,j} + z^{-\tau_{i,j}}}, \quad (2.15)$$

where $\tau_{i,j}$ is a delay line length and $\gamma_{i,j}$ is a feed-forward/back gain [71]. A low-order dense IIR filter is used as each coloration filter C_i .

Differentiable Implementation — We modify each FVN component to obtain differentiable FVN.

- *Velvet filters.* To batch the IR segments, we use the same segment lengths $L_1, \dots, L_S$ and average pulse distance values $T_1, \dots, T_S$ for every reference RIR.
- *Allpass filters.* We tune and fix the allpass filter parameters jointly with the velvet filter parameters to avoid perceptual artifacts, e.g., discontinuous and rough sound, while being computationally efficient. Since the filter is fixed, instead of the frequency-sampling method, we simply crop its IR to obtain an FIR $\tilde{u}_i[n]$.
- *Coloration filters.* We model each C_i with K serial SVFs and frequency-sample it into a length- N FIR $(c_i)_N[n]$, where $C_i(z) = \prod_{k=1}^K C_{i,k}^{\text{SVF}}(z)$, so that we can train our network to estimate its parameters.

Therefore, the IR of the differentiable FVN $\tilde{h}[n]$ becomes

$$\tilde{h}[n] = \hat{h}_0[n] + \text{overlap-add} \underbrace{\left\{ \overbrace{v_i[n] * \tilde{u}_i[n] * (c_i)_N[n]}^{\tilde{s}_i[n]} \right\}}_{\tilde{h}_i[n]}. \quad (2.16)$$

Since every component $v_i[n]$, $\tilde{u}_i[n]$, and $(c_i)_N[n]$ is an FIR, we can compute each segment $\tilde{h}_i[n] = v_i[n] * \tilde{u}_i[n] * (c_i)_N[n]$ efficiently by convolving the FIRs in the frequency domain, i.e., zero-padding followed by an FFT, multiplication, then an inverse FFT (IFFT). Finally, we overlap-add all the segments and add the deterministic FIR $\hat{h}_0[n]$ to obtain the full IR $\tilde{h}[n]$. The overlap-add can be implemented with a transposed convolution layer.

While the modifications for the differentiable FVN introduce approximation error, it is negligible in practice. We explain this with the EDR H^{EDR} , a reverse

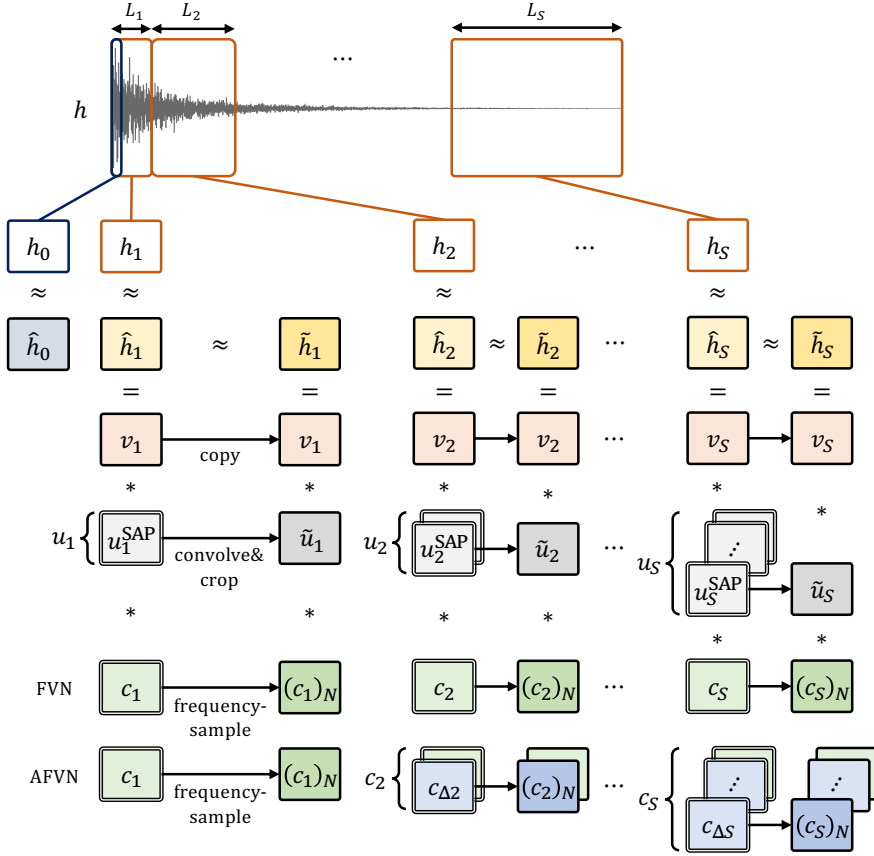
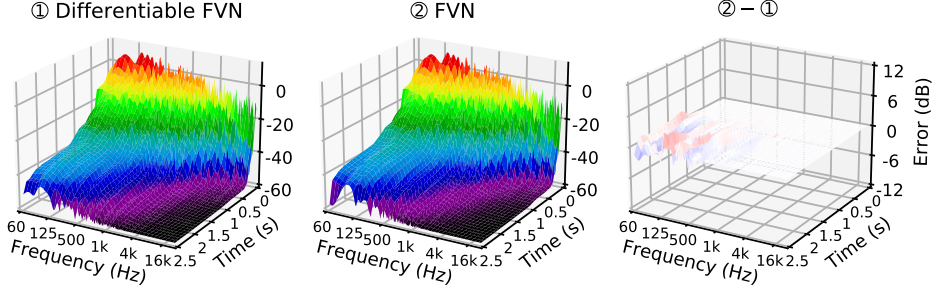
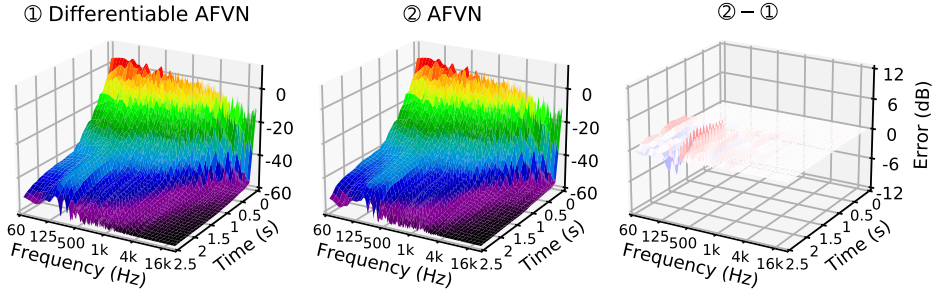


Figure 2.3: **Modified FVN, AFDN's RIR approximation, and their differentiable implementation strategy.** FVN and AFDN divide a target RIR $h[n]$ into segments $h_i[n]$ and approximate each segment with a velvet noise v_i filtered with an allpass filter U_i and a coloration filter C_i . We obtain the differentiable FVN and AFDN by converting the IIR filters U_i and C_i into FIR filters $\tilde{u}_i$ and $(c_i)_N$. Each IIR filter is emphasized with a double border box.



(a) Approximation error of differentiable FVN in EDR.



(b) Approximation error of differentiable AFVN in EDR.

Figure 2.4: EDR of IRs generated by differentiable FVN, AFVN, their original AR counterparts, and EDR errors introduced during the AR-to-DAR conversion process. Because the errors are small and mostly in the low-energy region, they are imperceptibly small (note that the z axis scale is different for the EDR plots and the EDR error plots). We obtained the FVN/AFVN parameters with the trained blind estimation networks discussed below.

cumulative sum of an IR's short-time Fourier transform (STFT) energy [74].

$$H^{\text{EDR}}[k, n] = \sum_{m=n}^{\infty} |H^{\text{STFT}}[k, m]|^2. \quad (2.17)$$

By comparing the FVN's EDR and its differentiable counterpart, we can analyze their differences in reverberation characteristics. Specifically, we obtain the EDR error E^{EDR} by calculating the difference between the two log-magnitude EDRs.

$$E^{\text{EDR}} = 10 \log_{10} \hat{H}^{\text{EDR}} - 10 \log_{10} \tilde{H}^{\text{EDR}}. \quad (2.18)$$

Figure 2.4a shows the EDRs and the EDR error. It reveals that the error is small

(at most 3dB) and mostly in the low-energy region (below -40dB). Also, there is little EDR error at the first time index, i.e., $|E^{\text{EDR}}[k, 0]| \ll 1$, indicating that the overall magnitude response remains the same during the conversion process. In conclusion, the error is hardly noticeable, and the differentiable FVN can replace the original FVN for the training (we use a loss function related to the EDR; see Section 2.5.4).

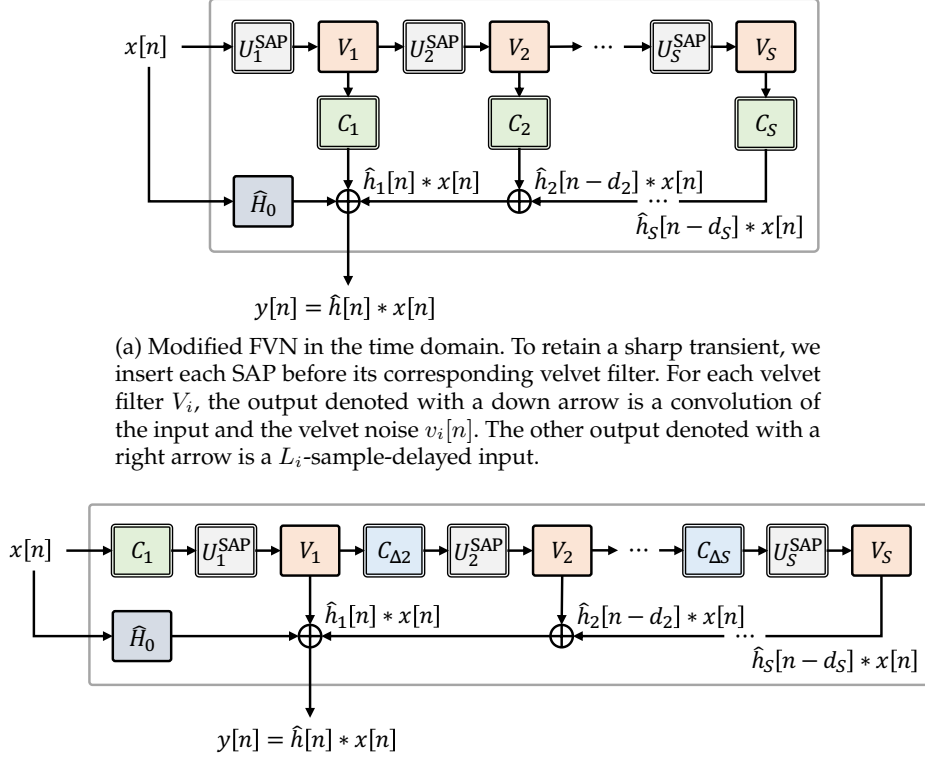
Modifications to the Original Model — FVN was initially designed for the analysis-synthesis of late reverberation; it used the direct arrival and early reflections cropped from the RIR and modeled only the late part. However, we also aim to solve the blind estimation problem where we must estimate the entire RIR. To this end, we modify the original model as follows.

- *Finer segment gain.* We divide each velvet segment $v_i[n]$ into M smaller segments $v_{i,1}[n], \dots, v_{i,M}[n]$ multiplied with gains $g_{i,1}, \dots, g_{i,M}$. Figure 2.5c shows the resulting velvet filter V_i in the time domain.
- *Cumulative allpass.* The original model smoothes every velvet noise with the same allpass filter. This is undesirable in our context since it could smear the direct arrival and early reflections. Instead, we gradually cascade the SAPs to obtain each allpass filter $U_i(z) = \prod_{j=1}^i U_j^{\text{SAP}}(z)$ so the earlier segments are less smeared than the later ones. As shown in Figure 2.5a, this modification can be implemented in the time domain by inserting each SAP U_j^{SAP} before the corresponding velvet filter V_j .
- *Shorter deterministic FIR.* With the above changes, the stochastic segments can fit the early RIR to some extent. Therefore, we shorten the early deterministic FIR $\hat{h}_0[n]$.

Refer to Appendix 2.A.1 for the evaluation of the modifications.

Model Configuration Details — Exact configurations of our FVN used for the experiments are as follows.

We generated 2.5 seconds of IR (48kHz sampling rate, total $L = 120\text{k}$ samples) with FVN. We used $S = 20$ non-uniform velvet segments whose lengths L_i are $L/40$, $L/20$, and $L/10$ for 10, 5, and 5 segments, respectively, and their average



(b) Modified AFVN in the time domain.

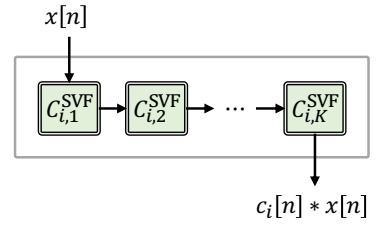
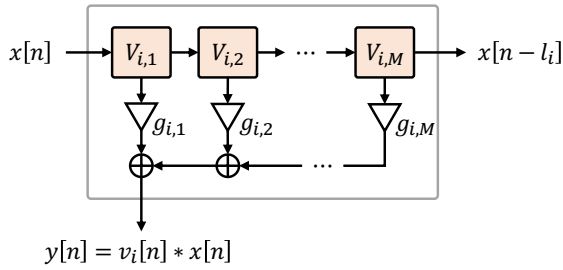


Figure 2.5: Modifications to the original FVN and AFVN.

pulse distances were set to $\mathbf{T} = [10, 20, 35, 50, 65, 90, 120, 135, 180, 220, 270, 320, 370, 420, 480, 540, 610, 680, 750, 820]$. The SAPs U_j^{SAP} were forced to have gains $\gamma_j = 0.75 + 0.01j$ and delay lengths $\tau = [23, 48, 79, 109, 113, 127, 163, 191, 229, 251, 293, 337, 397, 421, 449, 509, 541, 601, 641, 691]$. Each velvet segment had $M = 4$ sub-segment gains and was filtered with $K = 8$ SVFs. We frequency-sampled the coloration filters with $N = 4k$ points. The length of the deterministic FIR $h_0[n]$ was set to 50. We set the gain $\mathbf{g} \in \mathbb{R}^{S \times M}$, SVF parameters $\mathbf{f}, \mathbf{R}, \mathbf{m}^{\text{LP}}, \mathbf{m}^{\text{BP}}, \mathbf{m}^{\text{HP}} \in \mathbb{R}^{S \times K}$, and the bypass FIR $\mathbf{h}_0 \in \mathbb{R}^Z$ to estimation targets (total 930 ARPs). The time-domain FVN $\hat{H}$ requires 2546 floating point operations per sample (FLOPs).

2.4.2 Differentiable Advanced Filtered Velvet Noise

Since frequency-dependent decay of reverberation is gradual, one can model each coloration filter C_i as an initial coloration filter C_1 cascaded with delta-coloration filters $C_{\Delta 2}, \dots, C_{\Delta i}$.

$$C_i(z) = C_1(z) \prod_{j=2}^i C_{\Delta j}(z). \quad (2.19)$$

FVN with this modification is called *Advanced Filtered Velvet Noise* (AFVN) [73]. The delta filters' orders $\Delta K_2, \dots, \Delta K_S$ are set lower than the initial filter's K_1 for the efficiency. See Figure 2.5b for its time-domain implementation. We used $K_1 = 8$ and $K_\Delta = K_{\Delta 2} = \dots = K_{\Delta S} = 2$ in the experiments. This results in a total of $\bar{K} = K_1 + (S - 1)K_\Delta = 46$ SVFs. The other settings are the same as the FVN. This AFVN has 360 ARPs to estimate and its time-domain model $\hat{H}$ requires 1511 FLOPs. Although the orders of the coloration filters are higher than the FVN counterparts, the introduced frequency-sampling error remains approximately the same in practice (see Figure 2.4b).

2.4.3 Differentiable Delay Network

Delay Network — By interconnecting multiple delay lines in a recursive manner, one can obtain a *Delay Network* (DN) [101, 102] structure. A difference equation of the DN can be written in a general form as follows (we omit the bracket notation

for the filtering),

$$y[n] = \mathbf{C}^T \bar{\mathbf{y}}[n] + \hat{H}_0 x[n], \quad (2.20a)$$

$$\bar{\mathbf{y}}[n + \mathbf{d}] = \mathbf{A} \bar{\mathbf{y}}[n] + \mathbf{B} x[n]. \quad (2.20b)$$

That is, an input $x[n]$ is distributed and filtered (or simply scaled) with $\mathbf{B}$, then passes through $\mathbf{d}$ -sample parallel delay lines that are recursively interconnected through a mixing filter matrix $\mathbf{A}$. The delay line outputs $\bar{\mathbf{y}}[n]$ are filtered and summed with $\mathbf{C}$. We add a bypass signal filtered with $\hat{H}_0$, resulting in an output $y[n]$. The transfer function of the DN is

$$\hat{H}(z) = \mathbf{C}(z)^T (\mathbf{D}^{-1}(z) - \mathbf{A}(z))^{-1} \mathbf{B}(z) + \hat{H}_0(z). \quad (2.21)$$

Here, $\mathbf{D}(z) = \text{diag}(z^{-\mathbf{d}})$ is an $M \times M$ transfer function matrix for the delay lines, i.e., $\mathbf{D}_{ii}(z) = z^{-d_i}$. $\mathbf{B}(z)$, $\mathbf{C}(z)$, and $\mathbf{A}(z)$ are input, output, and feedback transfer function matrices of shape $M \times 1$, $1 \times M$, and $M \times M$, respectively.

Differentiable Implementation — The DN components are recursively interconnected such that one cannot divide its IR into independent segments and generate them in parallel like we did with the FVN and AFVN. Instead, we frequency-sample its entire transfer function to obtain differentiable DN, which is equivalent to a composition of individually frequency-sampled transfer function matrices.

$$\tilde{h}[n] = \text{IFFT} \left\{ \mathbf{C}_N^T (\mathbf{D}_N^{-1} - \mathbf{A}_N)^{-1} \mathbf{B}_N + (\hat{H}_0)_N \right\}. \quad (2.22)$$

Here, $\mathbf{D}_N = \text{diag}((z^{-\mathbf{d}})_N) \in \mathbb{C}^{M \times M \times \lfloor N/2+1 \rfloor}$ is a frequency-sampled version of the delay line transfer function matrix $\mathbf{D}$, i.e.,

$$(\mathbf{D}_N)_{ii} = (z^{-d_i})_N \in \mathbb{C}^{\lfloor N/2+1 \rfloor}. \quad (2.23)$$

Likewise, we can frequency-sample the transfer function matrices $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{A}$ to obtain their approximations $\mathbf{B}_N$, $\mathbf{C}_N$, and $\mathbf{A}_N$, respectively. Since each frequency-sampled transfer function matrix is a batch of $\lfloor N/2 + 1 \rfloor$ matrices, the matrix multiplications and inversions of equation (2.22) can be performed in parallel.

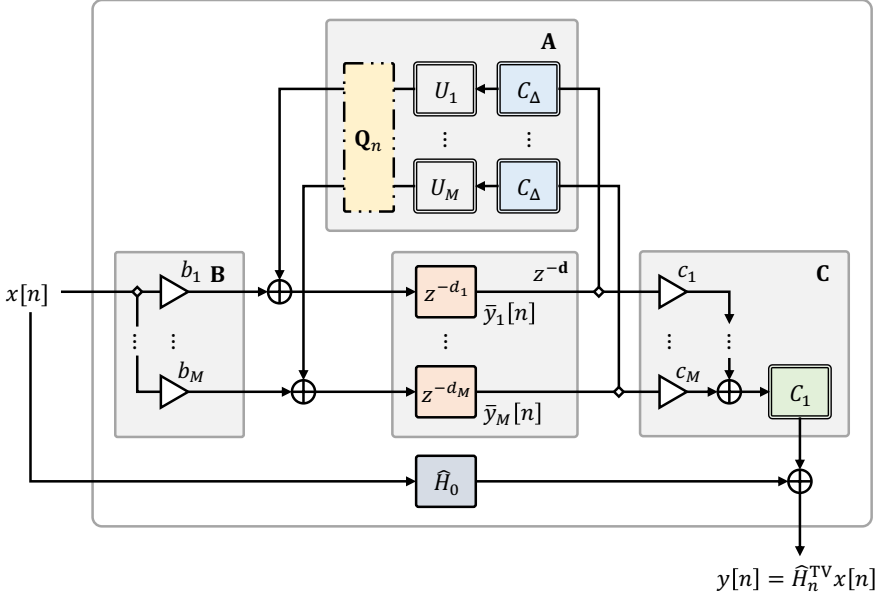


Figure 2.6: **Our DN in the time domain.** We restrict the general filter matrices to accelerate the training and obtain an efficient and controllable model. We simplify the pre-filters $\mathbf{B}$ and the post-filters $\mathbf{C}$ to have only one coloration filter C_1 and use the same C_Δ for all feedback loops. A time-varying mixing matrix $\mathbf{Q}_n$ and allpass filters $\mathbf{U}$ are introduced to use DN with small M (i.e., fast training) with minimal quality loss.

Restrictions to the General Model — The presented DN structure is fully general and able to express many variants, including the *Feedback Delay Network* (FDN) [72]. Therefore, we can also derive their differentiable versions following the proposed method. However, when following the general practice that uses many delay lines (e.g., $M = 16$) for a high-quality reverberation, its differentiable model’s frequency-sampled transfer function matrices consume too much memory, and their multiplications and inversions bottleneck the training speed. To tackle this, we reduce M and modify the components as follows to make DN plausible with low M (see Figure 2.6).

- *Pre and Post Filter Matrix.* Following most previous works [115, 116, 74], the pre filter matrix $\mathbf{B}$ is set to a (constant) gain vector $\mathbf{b}$. The post filter matrix $\mathbf{C}$ is a combination of a gain vector $\mathbf{c}$ and a common filter C_1 , i.e., $\mathbf{C} = C_1 \mathbf{c}$.

The common filter C_1 is composed of serial K_{C_1} SVFs.

- *Feedback Filter Matrix.* Most DN models combine channel-wise parallel delta-coloration (absorption) filters C_Δ and an inter-channel mixing matrix $\mathbf{Q}$ to compose the feedback filter matrix $\mathbf{A} = \mathbf{Q}C_\Delta$. Additionally, we insert channel-wise allpass filters $\mathbf{U}$ and introduce *time-variance* by modulating the mixing matrix $\mathbf{Q}$. This results in the time-varying feedback filter matrix $\mathbf{A}_n = \mathbf{Q}_n \mathbf{U} C_\Delta$.
- *Allpass Filters.* To achieve faster echo density build-up without adding more delay lines, we insert serial K_U SAPs in each feedback path. Unlike the FVN and AFVN cases, we estimate the SAP gains $\gamma \in \mathbb{R}^{M \times K_U}$ with the estimation network and frequency-sample the allpass filter matrix $\mathbf{U}$ for the differentiable model.
- *Time-varying Mixing Matrix.* We further reduce the audible “ringing” artifacts by modulating the stationary poles with the time-varying mixing matrix, which is set to $\mathbf{Q}_n = \mathbf{Q}_0 \mathbf{R}^n$ where $\mathbf{Q}_0$ is a Householder matrix and $\mathbf{R}$ is a tiny rotational matrix constructed from a random matrix [116, 117]. Since the frequency-sampling method is only applicable to LTI filters, we fix the mixing matrix to $\mathbf{Q}_0$ when obtaining the differentiable model.
- *Absorption Filters.* Unlike the conventional FDN [72], we use the same absorption filter C_Δ for every channel. Since the mixing matrix $\mathbf{Q}_n$ is always unitary, our DN becomes stable when C_Δ has magnitude response less than 1 [116]. We achieve this by using a PEQ with additional constraints (see Section 2.5.3) as C_Δ .
- *Bypass FIR.* We use a length- Z FIR $h_0[n]$ for the bypass.

We summarize the modifications. The difference equation of our time-varying DN $\hat{H}_n^{\text{TV}}$ is

$$y[n] = C_1 \mathbf{c}^T \bar{\mathbf{y}}[n] + H_0 x[n], \quad (2.24a)$$

$$\bar{\mathbf{y}}[n + \mathbf{d}] = \mathbf{Q}_n \mathbf{U} C_\Delta \bar{\mathbf{y}}[n] + \mathbf{b}x[n]. \quad (2.24b)$$

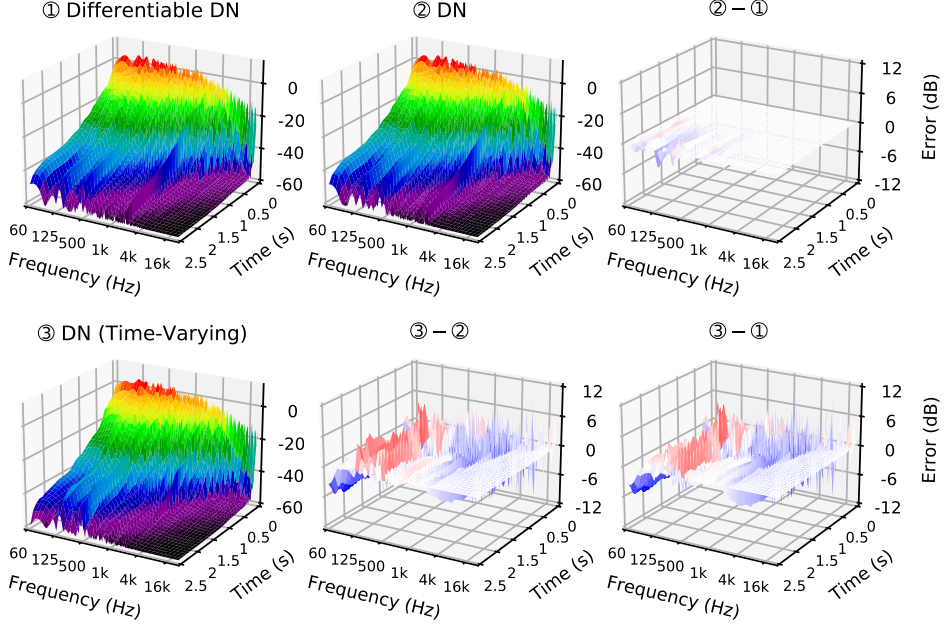


Figure 2.7: EDR of differentiable DN, its original DN with two versions (**LTI and time-varying**), and EDR errors between those three. Same as the FVN and AFVN, the frequency-sampling introduces little error, as shown in the upper right plot. However, rotating the mixing matrix introduces a considerable amount of EDR error since it modulates the pole positions. The error is mostly in the low-frequency region, where the poles are sparsely located.

The transfer function of its LTI approximation $\hat{H}(z)$ is given as equation (2.21) with the following restrictions,

$$\mathbf{B}(z) = \mathbf{b}, \quad (2.25a)$$

$$\mathbf{C}(z) = \mathbf{C}_1(z)\mathbf{c}, \quad (2.25b)$$

$$\mathbf{A}(z) = \mathbf{Q}_0 \text{diag}(\mathbf{U}_N(z) \odot \mathbf{C}_\Delta(z)). \quad (2.25c)$$

Hence, the IFFT of

$$(\mathbf{C}_1)_N \mathbf{c}^T (\mathbf{D}_N^{-1} - \mathbf{Q}_0 \text{diag}(\mathbf{U}_N \odot (\mathbf{C}_\Delta)_N))^{-1} \mathbf{b} \quad (2.26)$$

summed with the bypass FIR $\hat{h}_0[n]$ results in our differentiable DN's IR $\tilde{h}[n]$.

Figure 2.7 shows the EDR errors between the three different DN modes, i.e., differentiable DN, original LTI, and the time-varying one. The only difference between the first two is the frequency-sampling; unless the DN decays very slowly, there is little time aliasing and EDR error. However, due to the pole modulation, the EDR error increases significantly when the time-varying mixing matrix is applied. Despite this, other perceptually important factors, such as reverberation time, remain largely unchanged, as evaluated in Section 2.7.

Model Configuration Details — We used $M = 6$ delay lines with delay lengths $\mathbf{d} = [233, 311, 421, 461, 587, 613]$ in samples. We used $K_U = 4$ SAPs for each delay line. We fixed the SAP delay lengths to $\tau = [[131, 151, 337, 353], [103, 173, 331, 373], [89, 181, 307, 401], [79, 197, 281, 419], [61, 211, 257, 431], [47, 229, 251, 443]]$. Note that the SAPs introduce additional delays so that the effective delay lengths are $[1205, 1291, 1399, 1437, 1547, 1583]$ or about $[25.1, 26.9, 29.1, 29.9, 32.2, 33.0]$ in milliseconds. Our rotational matrix satisfies $\mathbf{R}^{30k} = \mathbf{I}$ so that the mixing matrix $\mathbf{Q}_n$ has a period of 0.625 second. Both post C_1 and absorption filter C_Δ have $K_{C_1} = K_{C_\Delta} = 8$ components. We used $N = 120k$ frequency-sampling points. The order of the bypass FIR is $Z = 100$. The post filter parameters $\mathbf{f}_{C_1}, \mathbf{R}_{C_1}, \mathbf{m}_{C_1}^{\text{LP}}, \mathbf{m}_{C_1}^{\text{BP}}, \mathbf{m}_{C_1}^{\text{HP}} \in \mathbb{R}^{1 \times K_{C_1}}$, feedback filter parameters $\mathbf{f}_{C_\Delta}, \mathbf{R}_{C_\Delta}, \mathbf{G}_{C_\Delta} \in \mathbb{R}^{K_{C_\Delta}}$, pre and post gain vectors $\mathbf{b}, \mathbf{c} \in \mathbb{R}^{M \times 1}$, SAP gain vector $\gamma \in \mathbb{R}^{M \times K_U}$, and the bypass FIR $\mathbf{h}_0 \in \mathbb{R}^Z$ are the estimation targets (total 200 ARPs). The resulting time-domain model $\hat{H}$ and the time-varying model $\hat{H}_n^{\text{TV}}$ consume 889 and 1285 FLOPs, respectively.

2.5 Artificial Reverberation Parameter Estimation

The details of our ARP estimation network are as follows. As shown in Figure 2.8, it transforms single channel audio input (either RIR or reverberant speech) into a shared latent $\mathbf{z}$ with a reference encoder. Then, each ARP-groupwise layer projects the latent $\mathbf{z}$ into an ARP tensor $\mathbf{P}_i$.

2.5.1 Reference Encoder

We first transform the reference RIR or reverberant speech into a log-frequency log-magnitude spectrogram. A standard convolutional neural network (CNN)–gated recurrent unit (GRU) encoder maps the spectrogram to a shared latent $\mathbf{z}$ ($\sim 7.3\text{M}$ parameters; see Figure 2.8).

2.5.2 ARP-Groupwise Projection Layers

Now we aim to transform the two-dimensional shared latent $\mathbf{z}$ into each ARP tensor $\mathbf{P}_i$ which is at most two-dimensional. To achieve this, we use two axis-by-axis linear layers. We first transpose and apply a linear layer to the latent to match the first axis shape. Then, another transposition and a linear layer match the other axis. This approach factorizes a single large layer into two smaller ones, reducing the number of parameters.

2.5.3 Nonlinear Activation Functions and Bias Initialization

Each ARP group has different stability conditions and desired distribution. To satisfy these, we attach a nonlinear activation to each projection layer and initialize the last linear layer’s bias as follows; Table 2.1 summarizes these choices for all AR models. Here, x denotes a pre-activation element.

Nonlinear Activation Functions — For the resonance $\mathbf{R}$ of the SVF, we use a scaled softplus $\zeta(x) = \log(1 + e^x)/\log(2)$ to center the initial distribution at 1 and satisfy the stability condition $\mathbf{R} > \mathbf{0}$. The cutoff parameter $\mathbf{f}$ has the same stability condition $\mathbf{f} > \mathbf{0}$, but instead of the softplus, we use $\tan(\pi\sigma(x)/2)$ where $\sigma(x) = 1/(1 + e^{-x})$ is a logistic sigmoid; with this activation, $\sigma(x) = 0$ and 1 represent cutoff frequencies of 0Hz and half of the sampling rate, respectively. The delta-coloration PEQ C_Δ inside the DN must satisfy $|C_\Delta(e^{j\omega})| < 1$, so we use $10^{-\zeta(x)}$ for the component gain $\mathbf{G}$. The shelving filters’ resonances should also satisfy $\mathbf{R} > \sqrt{2}/2$ to prevent their magnitude responses from spiking over 1, so we add $\sqrt{2}/2$ after the softplus activation. For the feed-forward/back gains γ of channel-wise allpass filters $\mathbf{U}$ inside the DN, we use sigmoid activation $\sigma(x)$.

Bias Initialization — We initialize the projection-layer bias for the cutoff parameter $\mathbf{f}$ to $\sigma^{-1}(2\omega_k/\omega_s)$, where each ω_k is the desired initial cutoff frequency of the k^{th} SVF. The frequencies are equally spaced on the logarithmic scale from $\omega_{\min} = 40\text{Hz}$ to $\omega_{\max} = 12\text{kHz}$, i.e., $\omega_k = \omega_{\min}(\omega_{\max}/\omega_{\min})^{(k-1)/(K-1)}$. For the SVFs inside the AFVN’s initial and delta-coloration filters, we randomly permute the frequency index k into a new index $\bar{k}$, preventing SVFs for the latter segments from having higher initial cutoff frequencies. Decoder biases for the SVF mixing coefficients $\mathbf{m}^{\text{LP}}$, $\mathbf{m}^{\text{BP}}$, and $\mathbf{m}^{\text{HP}}$ are initialized to 1, 2, and 1, respectively, so that each SVF’s initial magnitude response slightly deviates from 1; this prevents the coloration filters’ responses and the estimation network loss gradients from vanishing or exploding. For the PEQ gains $\mathbf{G}$ of the DN’s delta-coloration filters, we set each bias to -10 such that the initially generated IRs have long enough reverberation time.

2.5.4 Loss Function

Match Loss — We utilize a multi-scale spectral loss [47] and define a match loss as follows,

$$\mathcal{L}_{\text{Match}}(h, \tilde{h}) = \sum_i \left\| |H_i^{\text{STFT}}| - |\tilde{H}_i^{\text{STFT}}| \right\|_1. \quad (2.27)$$

i denotes STFT with different FFT sizes. We use the FFT sizes of 256, 512, 1024, 2048, and 4096, hop sizes of 25% of the respective FFT sizes, and Hann windows. Each spectrogram’s frequency axis is log-scaled like the encoder’s spectrogram.

Regularization — We additionally apply a regularization to reduce time aliasing of the estimation. As shown in equation (2.6) and (2.9), pole radii of each SVF affect the reliability of the frequency-sampled one (hence DAR model) and its parameter estimator. Since each SVF’s IR $(c_{i,k}^{\text{SVF}})_N[n]$ can be computed, we encourage reducing its pole radii by penalizing the IR’s decreasing speed. Each decreasing speed $\gamma_{i,k}$ is calculated by the ratio of the average amplitude of the first and last n_0 samples of $(c_{i,k}^{\text{SVF}})_N[n]$. We fix n_0 to $N/8$. Then, each $\gamma_{i,k}$ is weighted with a softmax function along the SVF-axis to penalize higher $\gamma_{i,k}$ more. The sum of the weighted

Table 2.1: **Target ARP tensors and nonlinear activation functions.** Bias initializations are shown for each AR model.

AR model	Component	ARP	Tensor shape	Activation	Initial bias
FVN	$s_i[n]$	$\mathbf{g}$	$S_{20} \times M_4$		
	C_i	$\mathbf{f}$	$S_{20} \times K_8$	$\tan(\pi\sigma(x)/2)$ $\zeta(x)/\zeta(0)$	$\sigma^{-1}(2\omega_k/\omega_s)$
		$\mathbf{R}$	$S_{20} \times K_8$		
		$\mathbf{m}^{\text{LP}}$	$S_{20} \times K_8$		1
		$\mathbf{m}^{\text{BP}}$	$S_{20} \times K_8$		2
		$\mathbf{m}^{\text{HP}}$	$S_{20} \times K_8$		1
	H_0	$\mathbf{h}_0$	Z_{50}		
AFVN	$s_i[n]$	$\mathbf{g}$	$S_{20} \times M_4$		
	$C_1, C_{\Delta j}$	$\mathbf{f}$	$\bar{K}_{46}$	$\tan(\pi\sigma(x)/2)$ $\zeta(x)/\zeta(0)$	$\sigma^{-1}(2\omega_{\bar{k}}/\omega_s)$
		$\mathbf{R}$	$\bar{K}_{46}$		
		$\mathbf{m}^{\text{LP}}$	$\bar{K}_{46}$		1
		$\mathbf{m}^{\text{BP}}$	$\bar{K}_{46}$		2
		$\mathbf{m}^{\text{HP}}$	$\bar{K}_{46}$		1
	H_0	$\mathbf{h}_0$	Z_{50}		
DN	$\mathbf{B}$	$\mathbf{b}$	M_6		
	$\mathbf{C}$	$\mathbf{c}$	M_6	$\tan(\pi\sigma(x)/2)$ $\zeta(x)/\zeta(0)$	$\sigma^{-1}(2\omega_k/\omega_s)$
		$\mathbf{f}_{C_1}$	$K_{C_1 8}$		
		$\mathbf{R}_{C_1}$	$K_{C_1 8}$		
		$\mathbf{m}_{C_1}^{\text{LP}}$	$K_{C_1 8}$		1
		$\mathbf{m}_{C_1}^{\text{BP}}$	$K_{C_1 8}$		2
		$\mathbf{m}_{C_1}^{\text{HP}}$	$K_{C_1 8}$		1
	$\mathbf{A}$	$\mathbf{f}_{C_\Delta}$	$K_{C_\Delta 8}$	$\tan(\pi\sigma(x)/2)$ $\zeta(x) + \sqrt{2}/2$ $10^{-\zeta(x)}$ $\sigma(x)$	$\sigma^{-1}(2\omega_k/\omega_s)$
		$\mathbf{R}_{C_\Delta}$	$K_{C_\Delta 8}$		
		$\mathbf{G}_{C_\Delta}$	$K_{C_\Delta 8}$		
		γ	$M_6 \times K_{\mathbf{u}4}$		-10
	H_0	$\mathbf{h}_0$	Z_{100}		

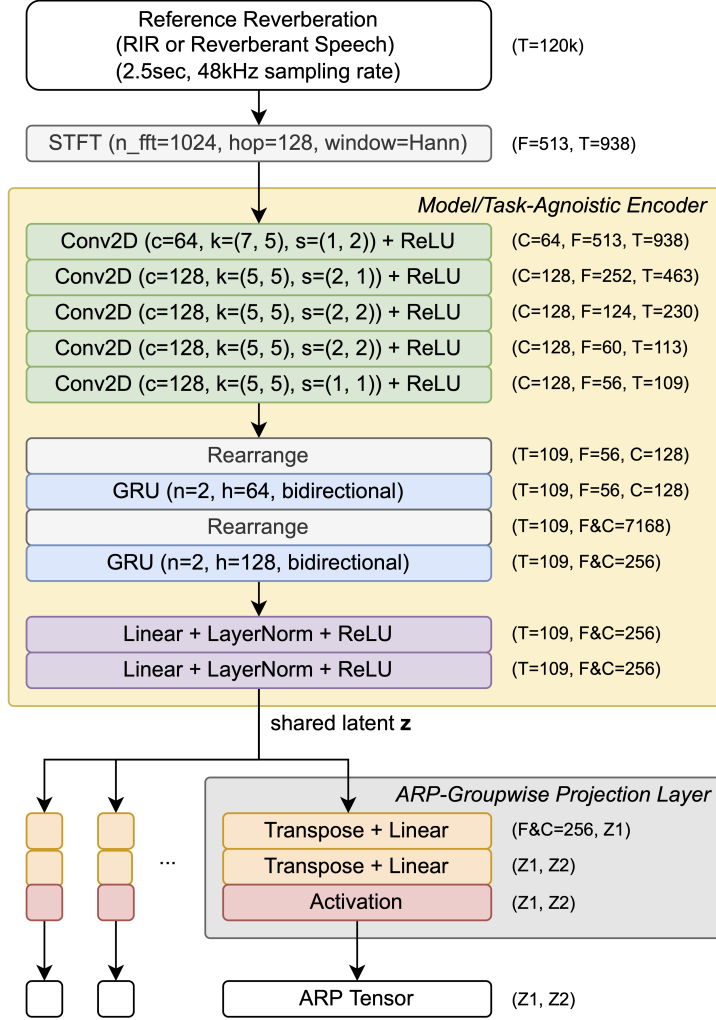


Figure 2.8: **Architecture of the ARP estimation network.** For each two-dimensional convolutional (Conv2D) layer, c , k , and s denote number of output channel, kernel size, and stride, respectively. For each gated recurrent unit (GRU) layer, n and h denote number of layers and hidden features. Shape and size of the intermediate output tensors are provided in the parentheses, where F , T , and C denote frequency, time, and channel axis, respectively. $Z1$, and $Z2$ denote each ARP tensor’s first and second axis. We omitted the batch axis.

decreasing speed values results in the regularization loss $\mathcal{L}_{\text{Reg}}$ as follows,

$$\gamma_{i,k} = \frac{\sum_{n=N-n_0}^N |(c_{i,k}^{\text{SVF}})_N[n]|}{\sum_{n=0}^{n_0} |(c_{i,k}^{\text{SVF}})_N[n]|}, \quad (2.28a)$$

$$\mathcal{L}_{\text{Reg}} = \sum_{i=1}^S \frac{\sum_{k=1}^K \gamma_{i,k} e^{\gamma_{i,k}}}{\sum_{j=1}^K e^{\gamma_{i,j}}}. \quad (2.28b)$$

We omit the regularization loss for the DN networks since, with DN, the time aliasing is inevitable to match the reference with reverberation time longer than the number of frequency-sampling points N . Hence, our full loss function $\mathcal{L}$ is

$$\mathcal{L}(h, \tilde{h}) = \mathcal{L}_{\text{Match}}(h, \tilde{h}) + \beta \mathcal{L}_{\text{Reg}}(\mathbf{c}_N^{\text{SVF}}) \quad (2.29)$$

where $\beta = 1$ for the FVN and AFVN and $\beta = 0$ for the DN.

2.6 Experimental Setup

2.6.1 Data

Room Impulse Response — We collected 1835 real-world RIR measurements from various datasets including OpenAIR [34] and ACE Challenge [35]. The number of RIRs was insufficient for training, so we used them only for the validation and test (836 and 999 RIRs, respectively), ensuring that each set consists of RIRs from different rooms.

For training, we synthesized 200k RIRs with shoebox room simulations using the image source method implemented in `pyroomacoustics` [118, 119]. To reflect various acoustic environments, we randomized simulation parameters for every RIR synthesis, including room size, each wall’s frequency-dependent absorption coefficient, and source/microphone positions. We tuned the randomization scheme to match the training dataset to the validation set in terms of reverberation parameter statistics.

We pre-processed each RIR by removing the pre-onset part, applying a small direct-sound gain augmentation to the training set, and normalizing its energy. The onset was detected by extracting a local energy envelope of the RIR and finding its

maximum point [120]. Following established RIR augmentation practice [121], we multiplied a random gain sampled from $\mathcal{U}(-12\text{dB}, 3\text{dB})$ to the first 5ms of each training RIR. This slightly reduces the average direct-to-reverberant ratio (DRR) of the training set and matches the validation set; during evaluation and testing, we omitted this procedure. Finally, each RIR was normalized to have unit energy.

Reverberant Speech — We used VCTK [122] for dry speech. We split it into the train (61808), validation (21608), and test (4912) sets so that each set is composed of the speech recordings from different speakers. We sampled an RIR and a dry speech sample from their respective datasets and convolved them to generate reverberant speech. We random-cropped 2.5-second segment from it for the input.

2.6.2 Training

We trained each network with every proposed DAR model for the three tasks: analysis-synthesis, blind estimation, and both tasks. For the both-performing ones, we fed either an RIR or reverberant speech with equal probability.

We set the initial learning rate to 10^{-4} for FVN and AFVN and 10^{-5} for DN. We used the Adam optimizer. For DN, gradients were clipped to ± 10 for stable training. After 250k steps, we performed the learning rate decay. We multiplied the learning rate by $10^{-0.2}$ and $10^{-0.1}$ every 50k steps for the analysis-synthesis networks and the others, respectively. We finished the training if the validation loss did not improve after 50k steps, which took no more than 500k steps for the analysis-synthesis and 1M steps for the others.

2.6.3 Evaluation Metrics

We evaluated our networks with the match loss $\mathcal{L}_{\text{Match}}$ and EDR distance, defined as an average of the absolute EDR error. Furthermore, we evaluated reverberation parameter differences, i.e., reverberation time T_{30} , DRR, and clarity C_{50} difference, denoted as ΔT_{30} , ΔDRR , and ΔC_{50} , respectively [103, 123]. We calculated both full-band differences and average of differences measured at octave bands of center frequencies of 125, 250, 500, 1k, 2k, 4k and 8kHz. Then, we obtained their respective median values.

We can interpret the reverberation parameter differences by comparing them with their respective just noticeable differences [100]. Reported just noticeable differences of T_{30} from previous works vary from 5% [124] up to about 25% [125]. For C_{50} , 1dB [126] and 1.1dB [127] were reported. For DRR it differs for various range, e.g., 6dB at -10 dB, 2dB at 0dB, 3dB at 10dB, and 8dB at 20dB [128].

2.6.4 Subjective Listening Test

Following prior studies [77, 82], we conducted a modified Multiple Stimuli with Hidden Reference and Anchor (MUSHRA) test [129]. We asked subjects to score the reverberation similarity between a given reference reverberant speech $h * x$ and the following signals (h and x denote a test RIR and dry speech signal, respectively).

- A hidden reference $h * x$ (exactly the same audio).
- A lower anchor $h' * x$ obtained by averaging the test set RIRs [82].
- Another anchor $h'' * x$ obtained by random-sampling an RIR from the test set [77].
- Estimations to evaluate $\hat{H}_1(x), \dots, \hat{H}_n(x)$. We used the original AR models $\hat{H}_1, \dots, \hat{H}_n$, not the differentiable ones $\tilde{H}_1, \dots, \tilde{H}_n$, to obtain the reverberant speech.

The similarity score ranged from 0 to 100, where we provided text descriptions for 5 equally divided ranges (0–20: totally different, 20–40: considerably different, 40–60: slightly different, 60–80: noticeable, but not much, and 80–100: imperceptible). While we asked the participants to score both analysis-synthesis and blind estimation results, we divided them into two separate pages. As a result, a total of 10 stimuli (3 proposed models, 4 baselines described in Section 2.6.5, 2 anchors, and 1 hidden reference) were scored for each task. We conducted the test using the webMUSHRA API [130]. A total of 24 sets were scored for each task, where the first 3 sets were used for training. A total of 12 people participated where 7 people were audio engineers and the others were instrument players. We discarded 1 subject’s results since they scored the hidden reference lower than 90 for more than 15% of the entire sets.

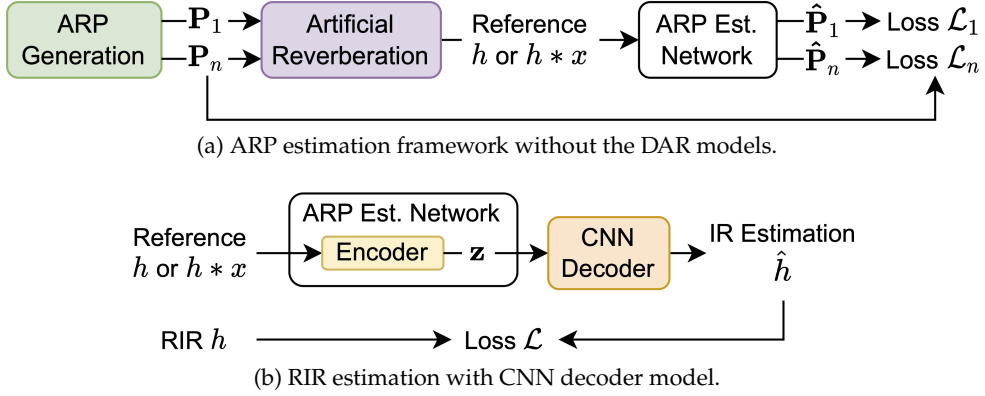


Figure 2.9: **Baseline methods for the experiments.**

2.6.5 Baselines

We compared our framework with the following baselines. Refer to Appendix 2.B for further details.

Parameter-matching Networks — Assuming that the DAR models are not available, similar to the previous non-end-to-end approach [77], we generated the training data (reference-and-ARPs pair) with the DAR models and trained the same networks with an ARP-matching loss (see Figure 2.9a). Note that we used the DAR models simply for the convenience of on-the-fly data generation on the GPU, but the same procedure can be performed with the AR models.

CNN Decoder — We trained the same estimation network but instead of the DAR model we attached a decoder composed of one-dimensional transposed convolutional layers and temporal convolutional networks (TCNs). With this decoder, the entire network resembles an autoencoder (see Figure 2.9b). From now, we denote this model as CNN. We trained two CNNs, one for each of the two tasks.

Note that we did not compare our methods with possible two-stage solutions [103, 104, 105, 106, 107, 108, 109, 110] (see Section 2.2.2) since each DAR model’s architecture became different from its original.

2.7 Evaluation Results

2.7.1 Comparison with the Baseline Models

Advantage of the End-to-end Learning — Figure 2.10 compares the proposed end-to-end approach with the baselines on objective metrics and subjective scores. Here, `AS` and `BE` denote analysis-synthesis and blind estimation, respectively. `Full` and `Freq` denote the full-band reverberation parameter difference and the average of octave-band reverberation parameter differences, respectively. The joint-task networks’ performance is denoted as small dots. By a large margin, the end-to-end model outperformed the non-end-to-end ARP-matching baselines. The baselines struggled to match the reverberation decay, reporting noticeably large reverberation time differences $\Delta T_{30} \geq 20\%$. A two-sided t-test for each AR/DAR model and task showed that the performance difference was statistically significant (all $p < 10^{-5}$). Subjective listening test results also agreed with the objective metrics; median scores of the end-to-end approaches were from 70 to 80, showing close matches to the reference, while the parameter matching models scored less than 60. Figure 2.11 further visualizes each estimation result with EDR and EDR-error plots. This shows that the end-to-end learning enabled by the DAR models improves ARP estimation relative to the parameter-matching scheme.

Advantage of the DSP Prior — The CNN baselines isolate the effect of removing the structured AR prior. Since the CNN baselines have more trainable parameters and complex architecture, they reach lower losses than the DAR-equipped networks. However, they produce audible artifacts. The AR models avoid such artifacts and achieve higher perceptual quality, reflected in higher subjective listening scores than the CNN baselines. Again, these subjective score differences were statistically significant (all $p < 10^{-5}$).

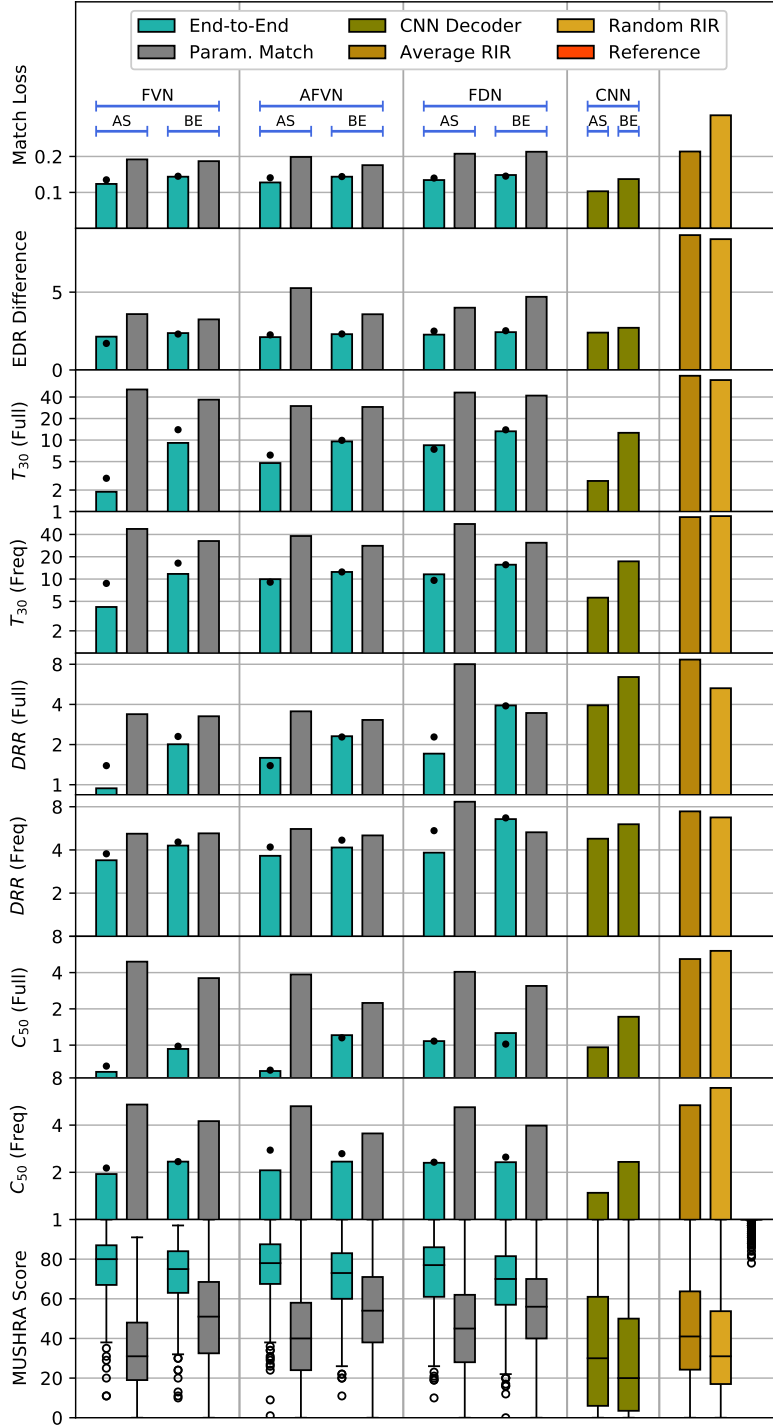


Figure 2.10: ARP estimation results on various tasks with the AR/DAR models.

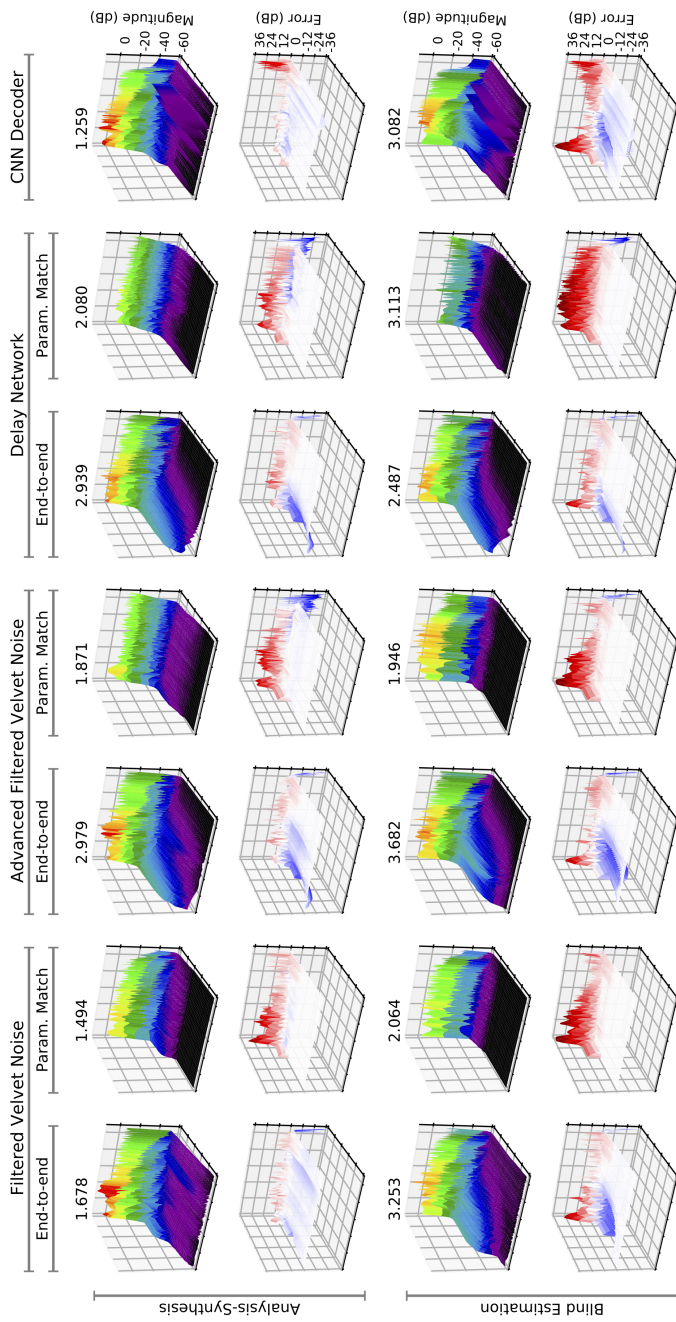


Figure 2.11: **EDR plots of a reference RIR (in short T_{30} range) and its estimations.** A total of twelve networks' estimations and their errors are visualized where each network is trained for one of the three AR models (FVN, AFVN, and DN), one of the two tasks (analysis-synthesis and blind estimation), and one of the two training approaches (end-to-end learning with the proposed DAR models and the baseline non-end-to-end parameter-matching learning). The number provided above each plot is the EDR distance of the estimation.

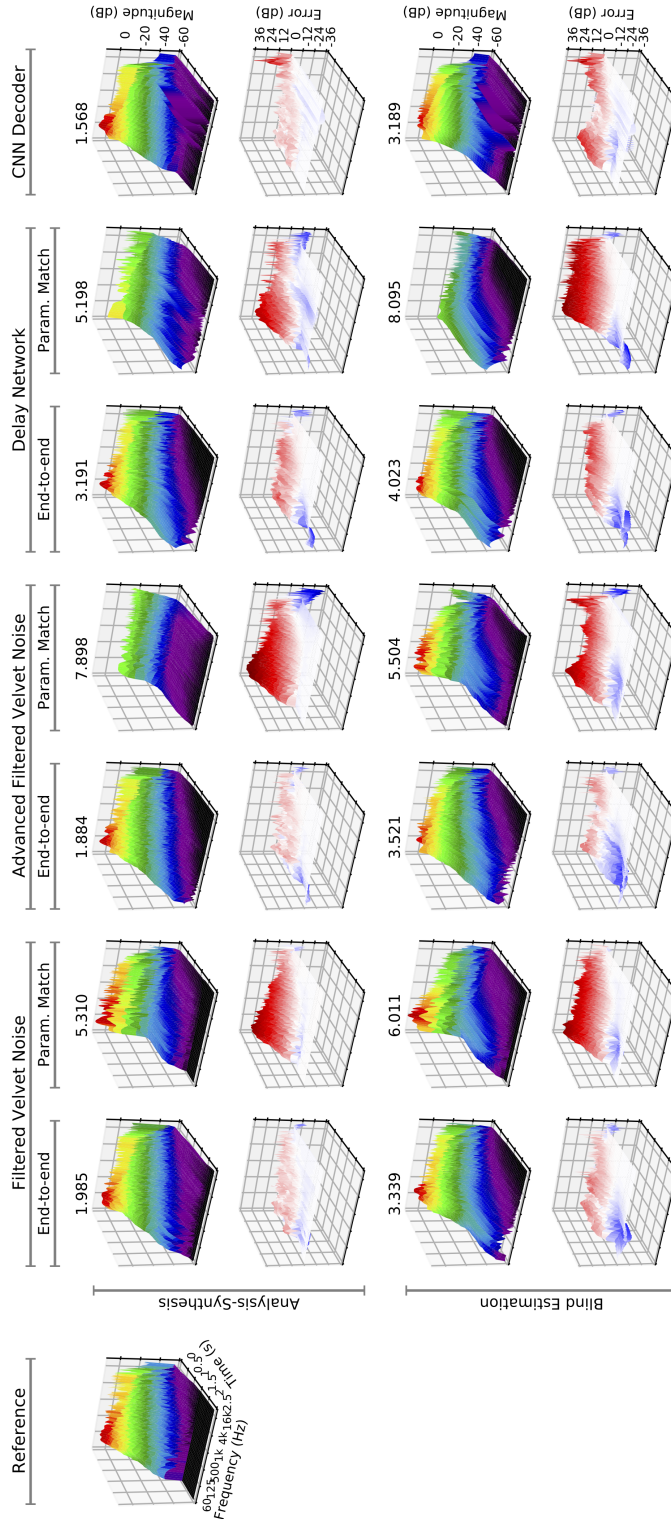


Figure 2.12: EDR plots of a reference RIR (in mid T_{30} range) and its estimations.

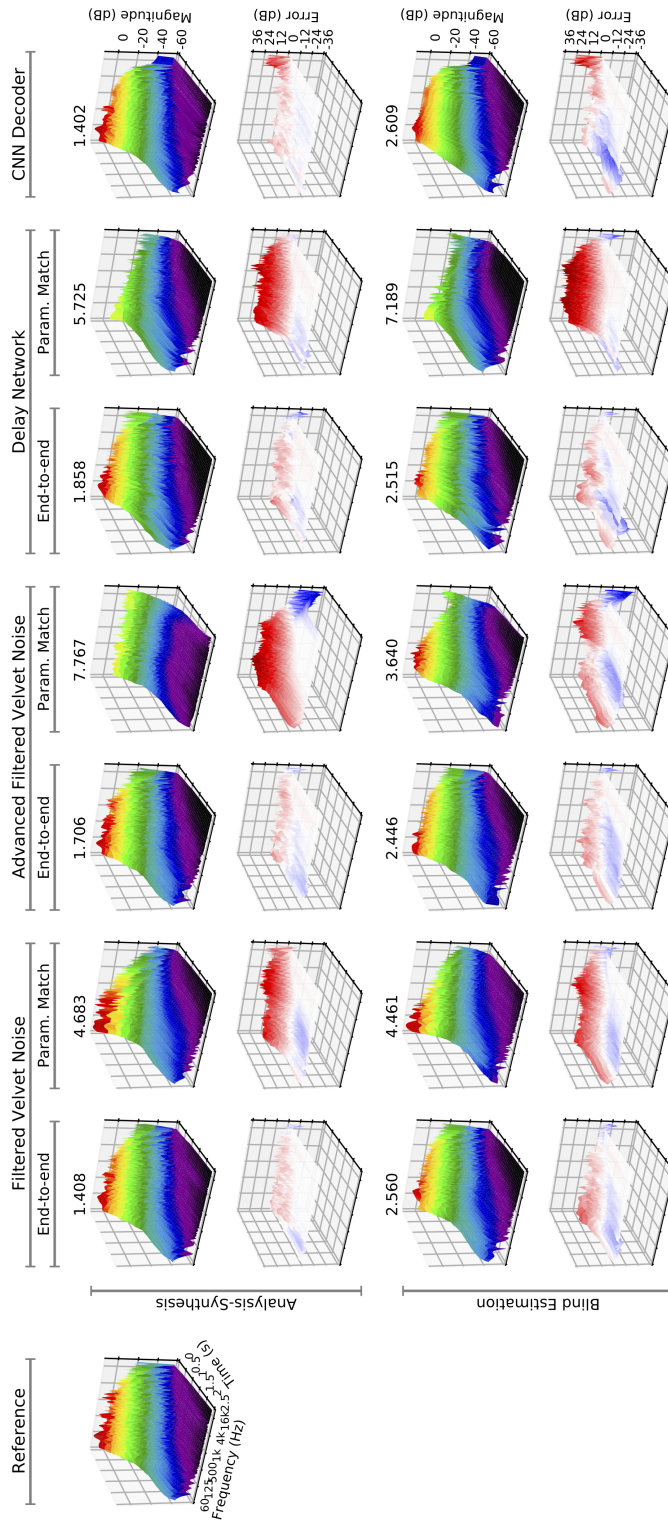


Figure 2.13: EDR plots of a reference RIR (in long T_{30} range) and its estimations.

2.7.2 Reliability of the DAR Models

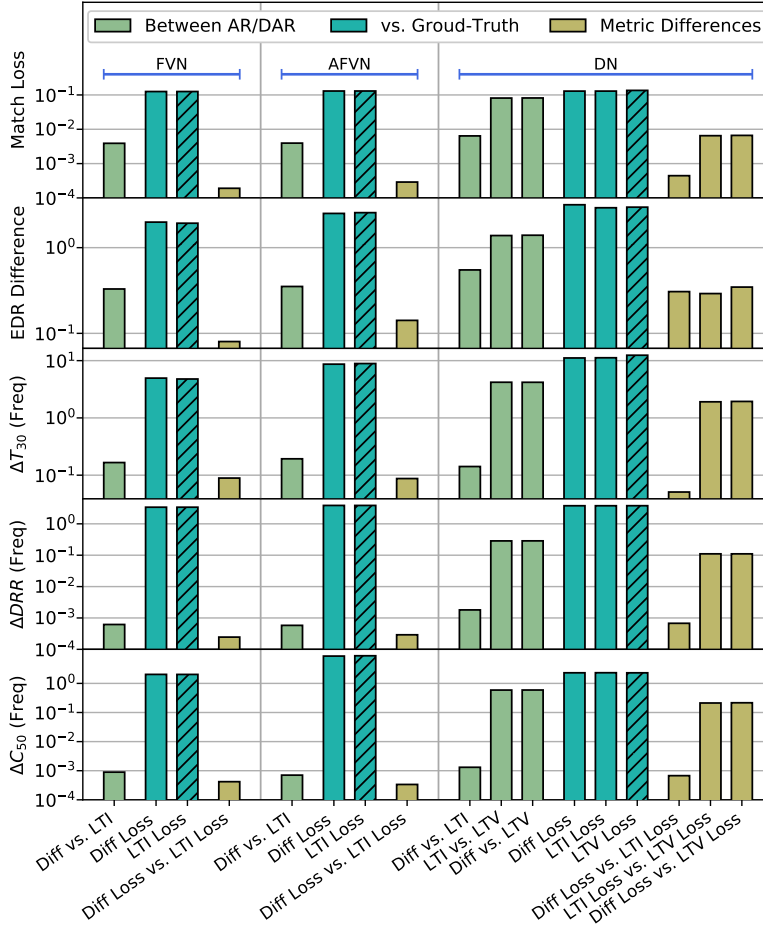


Figure 2.14: **Reliability of the DAR models.** *Diff*, *LTI*, and *LTV* denote the differentiable, original LTI, and the time-varying model, respectively. The hatched results are the same values of Figure 2.10 (analysis-synthesis).

The previous sections argued that each DAR model remains close to its original AR model, making it reliable as a training surrogate. Here, we empirically validate this argument. Figure 2.14 summarizes the objective metrics calculated with the DAR and AR models. We used the trained analysis-synthesis network for this evaluation. First, metrics calculated between the AR model IR and the DAR model IR were orders of magnitude smaller than the metrics calculated with the

ground truth. DN was the only exception, showing more deviation than the other AR models due to the time variance. Nevertheless, the reverberation parameter differences between the DN IR and differentiable model IR were less than the just noticeable difference values (for example, $\Delta T_{30} < 3\%$). Second, as expected with equation (2.8), each metric difference was smaller than the corresponding AR-to-DAR conversion error. As a result, overall evaluation results were very similar regardless of the used model. In short, the DAR models are reliable for training in practice.

2.7.3 Performance Difference Between Target Tasks

As expected, the analysis-synthesis networks performed better in most metrics than the blind estimation networks since the reference reverberation is directly given, i.e., the former have less inference burden than the latter. Likewise, the joint-task networks (shown as dots in Figure 2.10) show slightly degraded results than their single-task counterparts. Yet, the reverberation parameter difference only slightly increased (less than the just noticeable differences), indicating that the shared reference encoder can support both reference forms in this setting.

2.7.4 AR Model Efficiency and Estimation Performance

Each AR model approximates reference reverberation differently, and consequently, its computational efficiency and its estimation networks' performance vary.

Filtered Velvet Noise — FVN filters each source segment independently (see Figure 2.15). Such flexibility enables its parameter estimators to outperform those of the more constrained AR models. However, at the same time, it also has the largest number of ARPs (930), which makes it cumbersome to control manually. It could also be computationally expensive to use multiple FVN instances in real-time (2546 FLOPs for each).

Advanced Filtered Velvet Noise — AFVN instead optimizes each coloration filter by decomposing it into the initial coloration and delta-coloration filters, making the model more controllable and efficient (360 ARPs and 1511 FLOPs) than the FVN.

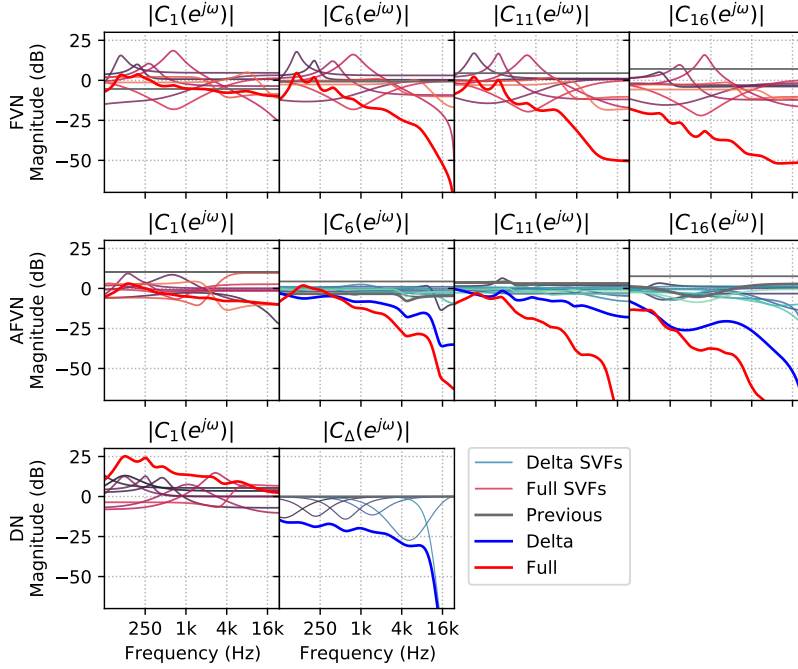


Figure 2.15: **Magnitude responses of the coloration filters of the AR models.** We used the proposed analysis-synthesis networks to obtain the filter parameters for this plot. Each AR model has a different approach to model the frequency-dependent characteristic of given reverberation.

Such simplification costs estimation performance since the underlying assumption of the optimization that coloration of real-world RIRs changes gradually does not necessarily hold. In addition, the AFVN networks could be more challenging to optimize than the FVN’s since each segment’s coloration depends on the previous ones’. Indeed, we empirically observed that their training losses decreased slower than the FVN’s.

Delay Network — In DN, we simplified the filter structure even further by using a single absorption filter for each feedback loop. Similar to the AFVN case, DN trades its estimators’ performance for controllability and efficiency (200 ARPs and 1285 FLOPs). Again, the estimation performance degradation comes from the DN’s restricted expressivity (forced exponential decay) and training difficulty.

2.8 Conclusion

We proposed DAR models, combining DSP-based AR with end-to-end ARP estimation. Among numerous pre-existing AR models, we implemented FVN, AFVN, and DN differentially by replacing IIR or near-LTI recursive components with FIR approximations via frequency-sampling; the same methodology applies broadly to near-LTI recursive filters whose transfer functions can be frequency-sampled. Using the DAR models, we trained ARP estimation networks end-to-end. The evaluation results showed that our networks captured the target reverberation accurately in both analysis-synthesis and blind estimation tasks. We showed, in particular, that end-to-end training significantly improves the estimation performance at the tolerable cost of the approximation errors caused by the frequency-sampling. Additionally, the structural priors of the AR models avoid perceptual artifacts that an unstructured RIR generator might produce, demonstrating the value of incorporating domain knowledge into the learning pipeline.

Here, we outline the remaining challenges and future work. First, we used the shoe-box simulation to obtain a large amount of training data, which is slightly different from the real-world RIRs. Notably, EDR errors of the end-to-end models showed similar patterns regardless of the equipped AR/DAR model (see Figure 2.11). This suggests that the data characteristics discrepancy might be the cause of the performance degradation. Therefore, collecting more real-world RIRs and using powerful augmentation methods could replace the simulation-based data and improve the performance. Second, we only used the RIRs and the reverberant speech signals as references. However, other possible applications with different reference types exist (for example, automatic mixing of musical signals). Finally, the choice of AR structure itself remains open. Our comparison showed that a more compact parameterization bounds the achievable estimation performance by its limited expressive power, so other AR structures and estimator designs may yield better performance-efficiency tradeoffs, especially when the AR parameterization is compact, controllable, and suitable for differentiable training.

Table 2.2: **Ablation on the AR/DAR model configurations.** Match losses are reported as $\mathcal{L}_{\text{Match}} (\times 10^{-1})$.

Model	$\mathcal{L}_{\text{Match}}$	ΔT_{30} (%)		ΔDRR (dB)		ΔC_{50} (dB)	
		Full	Freq	Full	Freq	Full	Freq
<i>Filtered Velvet Noise</i>							
Full FVN	1.236	1.89	4.20	0.94	3.39	0.60	1.95
Fixed pulse distance	1.257	2.74	6.14	1.56	4.21	0.48	2.25
Uniform segmentation	1.255	2.56	5.43	1.48	3.68	0.89	2.29
Without bypass FIR	1.250	2.23	10.71	3.17	11.94	0.81	9.88
No fine gains	1.261	2.53	4.42	1.51	3.65	0.82	2.21
<i>Advanced Filtered Velvet Noise</i>							
Full AFVN	1.277	4.77	9.96	1.59	3.64	0.61	2.06
No initial filters	1.446	6.75	12.60	1.46	8.43	1.76	3.64
No delta filters	1.324	7.12	32.16	2.99	5.01	0.92	2.62
No initial/delta filters	1.695	8.91	28.21	2.15	6.62	2.64	5.89
<i>Delay Network</i>							
Full DN	1.334	8.47	11.59	1.71	3.83	1.08	2.30
No initial filters	1.446	10.47	20.75	3.69	4.55	2.31	3.03
No delta filters	1.351	7.83	31.22	2.22	4.71	0.94	2.38
No initial/delta filters	1.729	11.62	31.20	4.00	6.68	2.03	4.54
No time-varying mixing	1.307	5.93	12.48	1.81	3.93	1.08	2.39
No SAPs	1.535	21.10	10.76	1.03	3.44	0.88	2.28
No SAPs/time-varying	1.251	7.43	10.79	1.00	3.39	0.70	2.09

Appendix 2.A Ablations/Comparisons of the AR/DAR Models

2.A.1 AR/DAR Model Ablations

We conducted ablation studies to verify that the modifications to the original AR models improve the estimation performance. We evaluated the modified models with the analysis-synthesis task since their performance differences remained the same regardless of the task in our initial experiments.

Table 2.2 summarizes the results. FVN, AFVN, and DN denote the proposed models. For the FVN, the ablated rows remove the non-uniform pulse distances ($T_i = 10$), the non-uniform segmentation ($\Delta l_i = L/S$), the bypass FIR ($h_0[n]$), and the finer segment gains ($M = 1$), respectively. Their results show that the proposed modifications contribute to better performance. We omitted the same

Table 2.3: **Differentiable FVN with various coloration filters and parameterization approaches.** Match losses are reported as $\mathcal{L}_{\text{Match}} (\times 10^{-1})$.

Filter	$\mathcal{L}_{\text{Match}}$	$\Delta T_{30} (\%)$		$\Delta \text{DRR} (\text{dB})$		$\Delta C_{50} (\text{dB})$	
		Full	Freq	Full	Freq	Full	Freq
Serial SVFs	1.236	1.89	4.20	0.94	3.39	0.60	1.95
Parallel SVFs	1.235	1.95	4.52	0.96	3.64	0.53	1.86
Parametric EQ	1.254	2.50	5.47	0.97	3.61	0.62	1.89
Serial Biquads	1.322	3.83	9.44	1.55	4.61	0.59	2.38
Parallel Biquads	1.337	4.88	11.77	1.57	4.01	0.62	2.19
FIR	1.370	2.72	10.10	1.80	3.73	0.51	2.16
Zero-Phase FIR	1.384	3.54	11.84	1.64	4.31	0.62	2.65

ablations for the AFVN. Instead, we discarded its initial filter C_1 , delta filter $C_{\Delta j}$, or both. We also evaluated the DN without the post filter C_1 or the absorption filter C_{Δ} . The results reveal the importance of modelling the frequency-dependent nature of reverberation.

For the DN, the last three rows remove the time-varying mixing matrix $\mathbf{Q}_n$, the allpass filters $\mathbf{U}$, and both, respectively. While removing both reports best evaluation metrics, its low echo density produced unrealistic sound. Using both $\mathbf{U}$ and $\mathbf{Q}_n$ improved plausibility of reverberation with an acceptable performance degradation. This illustrates that the objective metrics do not fully capture the perceptual plausibility of the reverberation.

2.A.2 Comparisons on Different Coloration Filters

We compared different IIR/FIR filters and parameterization approaches. We evaluated them with the analysis-synthesis task using FVN. Table 2.3 summarizes the results. The zero-phase FIR follows [47]. The order of every filter was set to 40. The activation functions and bias initialization for each filter’s decoder are as follows. The parallel SVFs have the same setting as the serial ones. For the biquad decoders, we used activations from [6], and their biases are initialized to match those of the SVFs. For the parametric EQ, unlike DN, we used 10^x for the activation of the gain $\mathbf{G}$. Both FIRs have no activation and custom bias initialization.

In spite of having identical expressive power, the SVFs performed better than the biquads by large margins. The parametric EQ performed slightly worse than

Table 2.4: **Effects of frequency-sampling resolution and regularization on the match loss.** Values are reported as $\mathcal{L}_{\text{Match}} (\times 10^{-1})$.

Model	Frequency-Sampling Resolution							
	$\times 1$ (Full)		$\times 0.5$		$\times 0.25$		$\times 0.125$	
	DAR	AR	DAR	AR	DAR	AR	DAR	AR
FVN $_{\beta=1}$	1.236	1.236	1.259	1.259	1.252	1.249	1.574	1.570
FVN $_{\beta=0}$	1.231	1.237	1.283	1.282	1.264	1.263	1.280	1.773
DN	1.277	1.307	—	1.328	—	1.361	—	1.393

the serial SVFs due to its restricted degree of freedom. Both FIRs also perform worse than the SVFs and the parametric EQ because they cannot change their frequency responses radically as other IIR filters could.

2.A.3 Effects of the Frequency-Sampling Resolution

Table 2.4 demonstrates the effects of the frequency-sampling resolution and the regularization loss $\mathcal{L}_{\text{Reg}}$. From the table, $\times x$ denotes the relative sampling resolution compared to the default configuration, e.g., $\times 0.5$ denotes $N = 2k$ for the FVN and $N = 60k$ for the DN. For each resolution and regularization setup, we report match loss with the DAR and AR models. Results of the differentiable DNs with lower resolutions are omitted since they are calculated with shorter IRs.

As expected, higher sampling resolution led to smaller loss difference. Also, introducing the regularization term ($\beta = 1$) reduced the difference further. As a result, the final differentiable FVN model (with full resolution and regularization) showed little loss difference. Moreover, higher resolution led to better performance, which might be because increased resolution helped each network to find better ARPs.

Appendix 2.B Details on the Baseline Methods

2.B.1 ARP Match Models

Training — We trained the ARP match baseline models as follows. First, we randomized ARPs $\mathbf{P}_1, \dots, \mathbf{P}_n$ and generated an IR using the DAR model. Then,

each baseline network estimated ARPs $\hat{\mathbf{P}}_1, \dots, \hat{\mathbf{P}}_n$ from the given IR. We evaluated the estimation and trained the baseline using an ARP-match loss $\mathcal{L}_{\text{ARP}}$ defined as follows,

$$\mathcal{L}_{\text{ARP}}(\mathbf{P}_i, \hat{\mathbf{P}}_i) = \sum_i \alpha_i \left\| f_i(\mathbf{P}_i) - f_i(\hat{\mathbf{P}}_i) \right\|_1. \quad (2.30)$$

Here, the index i denotes a different ARP tensor, e.g., $\mathbf{P}_0 = \mathbf{g}$ and $\mathbf{P}_1 = \mathbf{h}_0$. $f_i(\cdot)$ and α_i are an elementwise function and a constant, respectively. We set $\alpha_i = 10$ for all DAR models' bypass FIRs and the DN's absorption filter parameters. We used $f_i(x) = \log_{10}(x)$ for the FVN and AFVN segment gains. We chose $f_i(x) = x$ and $\alpha_i = 1$ otherwise.

Network Architecture — We trained an almost identical network to the proposed network for each model/task. The only difference is that we added an activation $10^{-\zeta(x)/20}$ to the $\mathbf{g}$ decoders of FVN and AFVN to improve their performance.

Data Generation — We tuned each DAR model's ARP randomization scheme to match the synthesized IRs' reverberation parameter statistics to the validation set. For the FVN baselines, we first sampled a reverberation time $T_{30} \sim \mathcal{U}_{\log}[50\text{ms}, 8\text{s})$ then generated the segment gains $\mathbf{g}$ that match the T_{30} . Here, $\mathcal{U}_{\log}[\cdot, \cdot)$ denotes a uniform distribution in log scale. In this procedure, we also compensated the average pulse distance by dividing each gain g_i by $\sqrt{T_i}$. For the initial coloration filter C_1 , we first generated PEQ parameters as $\omega_i \sim \mathcal{U}_{\log}[40\text{Hz}, 16\text{kHz})$, $R_i \sim \mathcal{U}_{\log}[0.2, 5)$, and $G_i \sim \mathcal{U}[-18\text{dB}, 18\text{dB})$ (we sorted the cutoff frequencies after the sampling) and derived their SVF parameters. After that, we perturbed each parameter's value slightly. This ARP generation method was motivated by the observation that the differentiable SVFs act like a relaxed PEQ (see Figure 2.15). We gradually changed the parameters of C_1 to obtain $C_2, \dots, C_K$, modeling the frequency-dependent decay. Finally, we set $h_0[n]$ as uniform noise with a gain sampled from $\mathcal{U}[-24\text{dB}, 0\text{dB})$. We followed similar procedures for the AFVN baselines. One difference is that the first delta filter $C_{\Delta 2}$ is randomized and the delta filters afterward are slight deviations of the first one. For the DN baselines, we sampled each absorption PEQ's gain with $G_i \sim \mathcal{U}[-2.8\text{dB}, 0\text{dB})$.

2.B.2 CNN Decoder Models

Regarding the first and second axis of the shared latent $\mathbf{z}$ as a time and channel axis, respectively, our decoder upsamples $\mathbf{z}$ with seven one-dimensional transposed convolution layers. Configurations of these layers are as follows. Channels: 128, 128, 64, 64, 64, 64, then 64. Kernel sizes: 5, 5, 5, 5, 5, 5, then 3. Strides: 3, 3, 3, 3, 3, 3, then 2. Furthermore, after each upsampling, we inserted a small temporal convolutional network with configurations as follows. Channels: 128, 128, 64, 64, 64, 64, and 64. The number of layers: 3, 3, 2, 2, 2, 2, and 2. Kernel sizes: all 7. Finally, we added 1×1 convolution as the last layer to mix all channels. We used the ReLU activation between the layers. This decoder has about 2.5M parameters.

Appendix 2.C Details on the Frequency-Sampling Method

2.C.1 Proof of Equation (2.6)

With equation (2.5), the time aliasing error can be written as

$$\|H - H_N\|_2^2 = \sum_{n=0}^{N-1} \left| \sum_{m=1}^{\infty} h[mN + n] \right|^2 + \sum_{n=N}^{\infty} |h[n]|^2. \quad (2.31)$$

With the triangle inequality and $\|X\|_2^2 \leq \|X\|_1^2$, we can upper-bound $\|H - H_N\|_2^2$ with $2S_N^2$ where $S_N = \sum_{n=N}^{\infty} |h[n]|$ is an absolute sum of the tail of the original IR $h[n]$ ($n \geq N$). Next, we expand $h[n]$ with partial fraction expansion [131, 3]. For $H(z)$ with M distinct poles where each of them $\nu_i \in \mathbb{C}$ has multiplicity $r_i \in \mathbb{N}$, we can express $h[n]$ as a sum of an FIR $h^{\text{FIR}}[n]$ and IIRs $h_{i,k}^{\text{IIR}}[n]$ where following holds,

$$H(z) = H^{\text{FIR}}(z) + \sum_{i=1}^M \sum_{k=1}^{r_i} \underbrace{\frac{\zeta_{i,k}}{(1 - \nu_i z^{-1})^k}}_{H_{i,k}^{\text{IIR}}(z)}. \quad (\zeta_{i,k} \in \mathbb{C}) \quad (2.32)$$

Then, applying the triangle inequality to the summation gives an upper bound U_N , an absolute sum of the IIRs' tails.

$$S_N \leq \sum_{i=1}^M \sum_{k=1}^{r_i} \sum_{n=N}^{\infty} |h_{i,k}^{\text{IIR}}[n]| = U_N. \quad (2.33)$$

From above equation, each IIR, first sum, and second sum show $O(n^{k-1}|\nu_i|^n)$, $O(N^{k-1}|\nu_i|^N)$, and $O(N^{r_i-1}|\nu_i|^N)$ asymptotic behavior, respectively [3]. We conclude the proof:

$$\|H - H_N\|_2 \leq \sqrt{2}S_N \leq \sqrt{2}U_N = \sum_{i=1}^M O(N^{r_i-1}|\nu_i|^N). \quad (2.34)$$

2.C.2 Proof of Equation (2.9)

We denote $H(e^{j\omega})$ and $\partial X/\partial p$ with H and X' . X'_N denotes $(X_N)' = (X')_N$. We upper-bound the gradient difference $\Delta\mathcal{G}$ using integral inequalities as

$$\Delta\mathcal{G} \leq \underbrace{\int_0^{2\pi} |H_{\text{Ref}}(H' - H'_N)| \frac{d\omega}{\pi}}_{\leq 2\|H_{\text{Ref}}\|_2 \|H' - H'_N\|_2} + \underbrace{\int_0^{2\pi} |HH' - H_N H'_N| \frac{d\omega}{\pi}}_{\leq 2\|HH' - H_N H'_N\|_2} \quad (2.35)$$

where the under-braced ones are the Cauchy–Schwarz inequalities. Considering the first integral's upper bound, $\|H_{\text{Ref}}\|_2$ is a finite constant. In $\|H' - H'_N\|_2$, H' is another stable LTI filter if p is well-defined. If ν_i is a function of p , its multiplicity r_i is doubled in H' . In the worst case, p controls every pole of H and $\|H' - H'_N\|_2 = \sum_{i=1}^M O(N^{2r_i-1}|\nu_i|^N)$ holds. Similarly, the upper bound of the second integral consists of an l_2 distance between HH' and $H_N H'_N$. Since HH' has $2r_i$ or $3r_i$ multiplicity for each pole ν_i , $\|HH' - H_N H'_N\|_2 = \sum_{i=1}^M O(N^{3r_i-1}|\nu_i|^N)$ holds at the worst case, which concludes the proof.

Chapter 3

Differentiable Acoustic Radiance Transfer

3.1 Introduction

Acoustic field learning [27] asks how to predict acoustic measurements at novel source–receiver configurations from sparse observations and a possibly approximate room geometry. In the trajectory of this thesis, this moves from estimating a reverberation model for an individual response (Chapter 2) to modeling the response field of a space. The estimated system description is therefore no longer a single response or a compact reverberator, but a room-dependent acoustic field constrained by geometry, source and receiver configurations, and material-related properties. Unlike sound field estimation [132, 133, 134, 135, 136, 137, 138], which typically reconstructs a pressure field within a local receiver region, acoustic field learning asks the model to generalize over source–receiver configurations, though recent sound field work has begun to relax this distinction [139].

Acoustic field learning has been explored using deep neural networks, considering different estimation targets: most commonly room impulse responses (RIRs) [27, 80, 140, 141, 142], but also propagated speech [143, 144, 145] or acoustic parameters (e.g., reverberation time) [146]. These methods have reported promising results on both synthetic [147] and real-world benchmarks [148]. However, they are typically trained on large-scale measurements from a single environment. For example, the RIR predictors [27, 80, 140, 141, 142] have primarily been demonstrated as efficient codecs, compressing dense collections of RIRs to reduce storage

usage. Yet acoustic measurements are costly and time-consuming to acquire, and improving model performance under data-scarce conditions is desirable for practical deployment.

Several directions have been pursued to improve such data efficiency. One approach is to formulate the task as few-shot learning: instead of training an independent neural network for each scene, a single model can be trained on multiple scenes to maximize data utilization and promote generalization [140, 146, 149]. A different approach is to leverage auxiliary geometry-related information; extra visual input can be utilized to estimate room geometry [144, 145, 150, 151, 152, 149] or the room surface mesh can be directly used [153]. Finally, some incorporate inductive biases by designing (or training) models to encode properties of sound propagation, such as the inverse square law [142], acoustic reciprocity [27, 141, 154], and propagation time delay [142]. This brings the models closer to acoustic simulators. However, it remains unclear whether these models combine such laws with room geometry information in a physically consistent manner.

Following the differentiable signal processing perspective of Section 1.5, this chapter instead keeps an explicit acoustic forward model and optimizes its physically meaningful variables. Differentiable room acoustics models range from wave-based methods [155], which are accurate but expensive, to geometric acoustics methods [28, 156, 157, 158, 159], which solve the acoustic rendering equation (ARE) [25] by treating sound propagation as rays. However, most prior differentiable geometric acoustics models fix the material behavior and optimize only reflection or absorption coefficients. Several recent hybrid approaches add neural components to compensate for simulator inaccuracies [160, 161]; here, the complementary goal is to make the geometric acoustics model itself more expressive and learnable.

In this context, *Acoustic Radiance Transfer* (ART) [25, 162, 163, 164, 165, 166, 167] appears to be a particularly attractive geometric acoustics model. ART computes time- and direction-dependent energy exchange between surface patches through discretization, transforming the ARE into a finite linear system. This formulation enables the use of arbitrary material properties while retaining high runtime efficiency. Nevertheless, ART has not yet been considered within a differentiable

signal processing framework.

To fill this gap, we introduce differentiable acoustic radiance transfer (DART), a differentiable implementation of ART. The primary challenge of realizing DART is the high computational cost; ART achieves its noted efficiency only at runtime when its materials remain fixed, a condition that does not hold during optimization. We overcome this challenge by introducing multiple techniques, including kernel decomposition, frequency-domain computation, and sparse storage/operations. As a result, we significantly reduce memory and computational cost while retaining flexible material parameterization.

We evaluate DART on acoustic field learning with sparse observations. In this chapter, the predicted acoustic measurement is the acoustic energy response, or *echogram*, of a novel source–receiver pair, rather than the full RIR waveform. While this task might appear simpler, our investigation reveals that it is still highly challenging in many practical scenarios, such as sparse measurements (from 4 to 48), multi-room scenes with diverse acoustic characteristics, and evaluation on unseen source/receiver regions. Existing baselines, such as the differentiable image source method [28] and neural network-based approaches [80, 142], struggle in these settings, often failing to beat simple nearest-neighbor or linear interpolation. On the other hand, due to its grounded physical formulation, DART consistently shows strong performance.

The subsequent analyses compare material parameterizations, test robustness to geometric distortion, validate design choices through ablation studies, and measure computational and memory cost. An open-source implementation of DART accompanies this chapter; to our knowledge, it is among the first open-source ART implementations beyond simplified versions [168, 169].

3.2 Background

3.2.1 Acoustic Rendering Equation

Definition — Accounting for the non-negligible speed of sound, Siltanen et al. [25] extended the rendering equation for light transport [78] to the acoustic rendering equation. Sound is regarded as a collection of “rays,” each carrying an acoustic

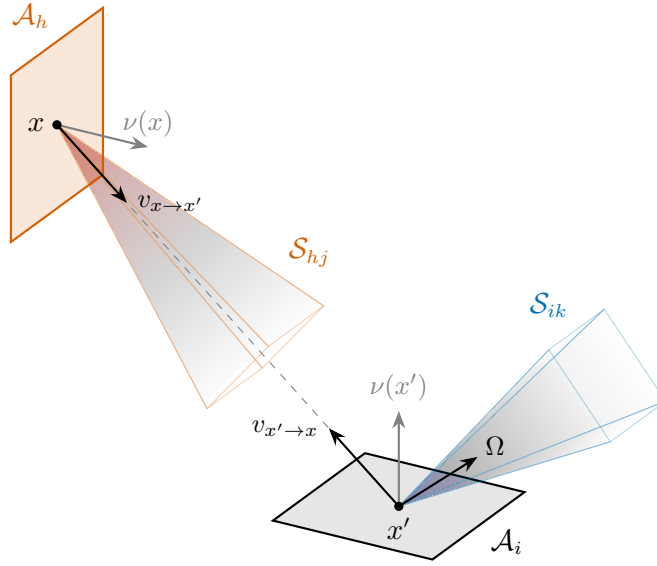


Figure 3.1: **The acoustic rendering equation (ARE) and acoustic radiance transfer (ART).** For a room boundary $\mathcal{A}$, ARE states that a radiance from a surface point $x' \in \mathcal{A}$ with outgoing direction Ω equals all other radiances heading to the point x' properly attenuated and delayed with a reflection kernel R . ART discretizes the ARE into surface patches (two of which, $\mathcal{A}_h$ and $\mathcal{A}_i$, are depicted) and direction bins ($\mathcal{S}_{hj}$ and $\mathcal{S}_{ik}$, visualized as cones). Arrows depict the direction vectors used in the equations. Patches are shown as rectangles for clarity; in practice, triangle meshes are used.

radiance $L(x', \Omega, t) \in \mathbb{R}^+$, defined as the time-varying energy flux per unit projected area per unit solid angle. It is a function of surface point $x' \in \mathcal{A}$, emitting direction $\Omega \in \mathcal{S}$, and continuous time $t \in \mathbb{R}$, where $\mathcal{A}$ and $\mathcal{S}$ denote a room surface geometry and the set of all unit-length direction vectors, respectively. See Figure 3.1 for an illustration. The ARE reads

$$L(x', \Omega, t) = L^{(0)}(x', \Omega, t) + \iint_{\mathcal{A}} R(x, x', \Omega, t) * L(x, v_{x \rightarrow x'}, t) d\mathcal{A}(x). \quad (3.1)$$

That is, the radiance is the sum of an initial term $L^{(0)}$ and the aggregation of other radiances convolved ($*$) with a reflection kernel R . Here, $v_{x \rightarrow x'} = (x' - x) / \|x' - x\|$ denotes a unit-length direction vector from a surface point x toward x' . Formal definitions of $L^{(0)}$ and R will be introduced later. The use of radiance for room acoustics is valid when the characteristic dimensions of the room geometry are

orders of magnitude larger than the longest wavelength of interest [26].

Neumann Series — The ARE has a so-called Neumann series solution [78], where the radiance L can be obtained as an infinite sum of k -th order terms,

$$L(x', \Omega, t) = L^{(0)}(x', \Omega, t) + \sum_{k=1}^{\infty} L^{(k)}(x', \Omega, t), \quad (3.2)$$

where each k -th order term $L^{(k)}$ (for $k \geq 1$) is obtained by recursive reflections of the initial radiance $L^{(0)}$.

$$L^{(k+1)}(x', \Omega, t) = \iint_{\mathcal{A}} R(x, x', \Omega, t) * L^{(k)}(x, v_{x \rightarrow x'}, t) d\mathcal{A}(x). \quad (3.3)$$

In practice, the series is truncated at a finite order, chosen large enough that the remaining energy is perceptually negligible.

The reflection kernel is typically decomposed into four factors as

$$R(x, x', \Omega, t) = V(x, x') G(x, x') D(x, x', t) \rho(v_{x' \rightarrow x}, \Omega; x'). \quad (3.4)$$

The first term, $V(x, x')$, is a visibility function that equals 1 if x and x' are in an unobstructed line of sight, 0 otherwise. The second term is known as a patch-to-patch geometry, given as

$$G(x, x') = \frac{|v_{x' \rightarrow x} \cdot \nu(x')| |v_{x \rightarrow x'} \cdot \nu(x)|}{\|x - x'\|^2} \quad (3.5)$$

where $\nu(x)$ denotes the unit-length normal vector of the surface $\mathcal{A}$ at $x \in \mathcal{A}$ and each dot product computes the cosine of the angle between the two vectors. Its numerator is the product of two absolute cosines, corresponding to Lambert's cosine law for the source and reflecting patches. Its denominator corresponds to the inverse square law. The third term models propagation delay, given as

$$D(x, x', t) = \delta\left(t - \frac{\|x - x'\|}{c}\right) \quad (3.6)$$

where $\delta(\cdot)$ and c denote the Dirac delta and the speed of sound, respectively. We omit the attenuation due to air absorption for simplicity. These three terms,

visibility, patch-to-patch geometry, and delay, depend only on the room geometry $\mathcal{A}$.

The last term $\rho(v_{x' \rightarrow x}, \Omega; x')$ is the bidirectional reflectance distribution function (BRDF) [170], which describes how incident energy arriving from direction $v_{x' \rightarrow x}$ is reflected into an outgoing direction Ω at the surface point x' . In this chapter, we generalize this to the bidirectional scattering distribution function (BSDF) [171], which, to our knowledge, has not previously been explored in ART. While the BRDF is defined over a single hemisphere, modeling either exterior or interior reflection, the BSDF extends both input and output domains to the full sphere $\mathcal{S}$. This enables the modeling of transmission, which is common in complex multi-room geometries. We nonetheless retain the term “reflection kernel” for consistency with the existing literature.

Injection — The injection process computes the initial radiance $L^{(0)}$, which represents the result of the emitted sound undergoing its first reflection (or transmission) at the room surface. For a source s emitting a unit impulse $\delta(t)$, the initial radiance is given by

$$L^{(0)}(x, v_{x \rightarrow x'}, t) = \frac{\Gamma_s(v_{x_s \rightarrow x}, o_s)}{4\pi} \frac{|v_{x \rightarrow x_s} \cdot \nu(x)|}{\|x - x_s\|^2} \times V(x_s, x) D(x_s, x, t) \rho(v_{x \rightarrow x_s}, v_{x \rightarrow x'}; x) \quad (3.7)$$

where x_s , o_s , and Γ_s denote source position, orientation, and directivity, respectively.

Detection — A receiver r with position x_r , orientation o_r , and directivity Γ_r receives the power from the radiances as

$$E_L(t) = \iint_{\mathcal{A}} L(x, v_{x \rightarrow x_r}, t) * \Gamma_r(v_{x_r \rightarrow x}, o_r) \frac{|v_{x \rightarrow x_r} \cdot \nu(x)|}{\|x - x_r\|^2} \times V(x, x_r) D(x, x_r, t) d\mathcal{A}(x). \quad (3.8)$$

Finally, with the direct arrival from the source to the receiver,

$$E_D(t) = \frac{\Gamma_s(v_{x_s \rightarrow x_r}, o_s) \Gamma_r(v_{x_r \rightarrow x_s}, o_r)}{4\pi \|x_r - x_s\|^2} V(x_s, x_r) D(x_s, x_r, t), \quad (3.9)$$

we obtain a time-varying response called the *echogram*

$$E(t) = E_D(t) + E_L(t), \quad (3.10)$$

which describes energy transfer from the source s to the receiver r under the room geometry $\mathcal{A}$. Each echogram can be used to synthesize an RIR with the so-called auralization process, e.g., noise shaping [172].

3.2.2 Acoustic Radiance Transfer

Discretization — We first discretize the geometry into N_{pat} patches. The set of all incoming and outgoing directions (the unit sphere $\mathcal{S}$) is also partitioned into N_{dir} bins independently for each patch $\mathcal{A}_i$.

$$\mathcal{A} = \mathcal{A}_1 \cup \dots \cup \mathcal{A}_{N_{\text{pat}}}, \quad \mathcal{S} = \mathcal{S}_{i1} \cup \dots \cup \mathcal{S}_{iN_{\text{dir}}}. \quad (3.11)$$

Then, in ART [25], a discrete radiance is defined as an average of the continuous radiance over the discretized patch $\mathcal{A}_i$ and direction bin $\mathcal{S}_{ik}$. The continuous time t is also discretized with sampling period Δt , yielding a discrete index n .

$$\hat{L}_{ik}[n] = \iint_{\mathcal{S}_{ik}} \iint_{\mathcal{A}_i} L(x', \Omega, n\Delta t) \frac{d\mathcal{A}(x')}{|\mathcal{A}_i|} \frac{d\Omega}{|\mathcal{S}_{ik}|}. \quad (3.12)$$

Here, $|\mathcal{A}_i|$ and $|\mathcal{S}_{ik}|$ denote the area of the surface patch $\mathcal{A}_i$ and the solid angle of the direction bin $\mathcal{S}_{ik}$, respectively.

Assume that the continuous radiance is approximately constant within each discretized patch and direction bin:

$$L(x', \Omega, n\Delta t) \approx \hat{L}_{ik}[n], \quad x' \in \mathcal{A}_i, \Omega \in \mathcal{S}_{ik}. \quad (3.13)$$

Then, for a discretized reflection kernel defined as

$$\hat{R}_{hj,ik}[n] = \iint_{\mathcal{S}_{ik}} \iint_{\mathcal{A}_i} \iint_{\mathcal{S}_{hj}} \iint_{\mathcal{A}_h} \tilde{R}(x, \Theta, x', \Omega, n\Delta t) d\mathcal{A}(x) d\Theta \frac{d\mathcal{A}(x')}{|\mathcal{A}_i|} \frac{d\Omega}{|\mathcal{S}_{ik}|} \quad (3.14)$$

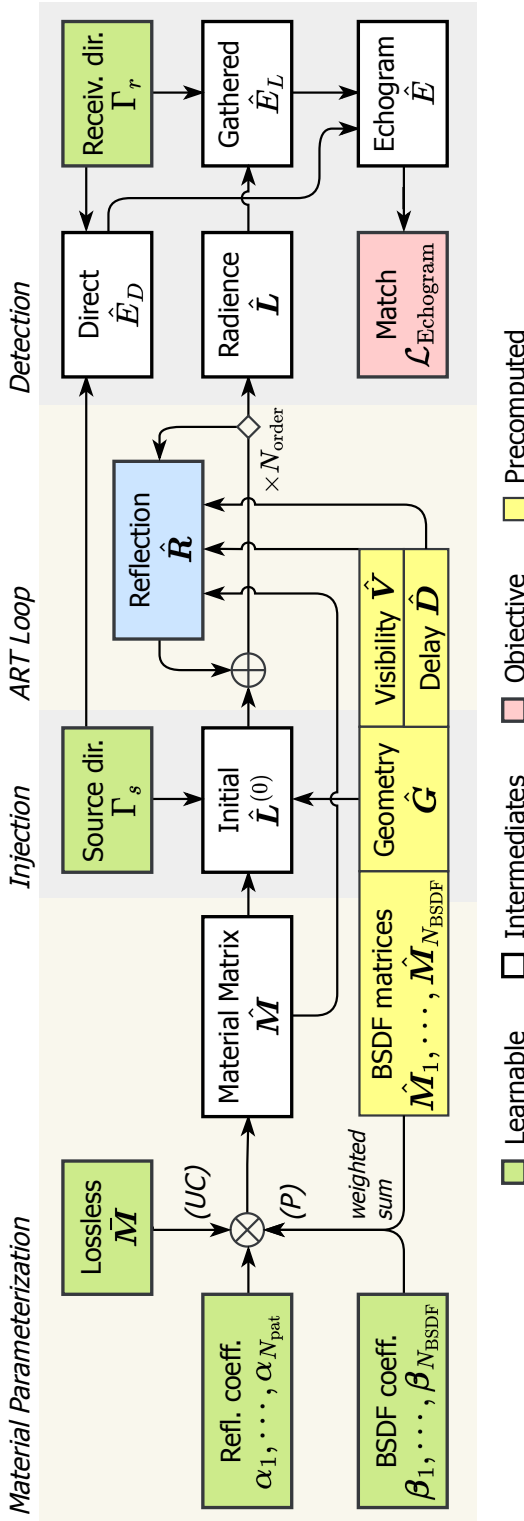


Figure 3.2: **Overview of DART.** An echogram is computed in three stages: injection, iterative reflections ("ART loop"), and detection. The reflection, highlighted in a blue box, is decomposed into three components: delay, mean visibility, and material matrix. The first two can be precomputed, reducing the optimization cost. The third component, the material matrix, depends on material parameters and is therefore computed for every optimization step. The material matrix is also used for the injection stage. The material matrix has unconstrained (UC) and parametric (P) variants. Learnable source or receiver directivities can optionally be introduced. All signal flows are differentiable, so that the echogram prediction loss can be backpropagated to all learnable components (shown as green boxes) end-to-end.

where we used the auxiliary reflection kernel given as

$$\tilde{R}(x, \Theta, x', \Omega, t) = R(x, x', \Omega, t) \cdot \delta(v_{x \rightarrow x'} - \Theta), \quad (3.15)$$

we obtain the discrete version of the ARE.

$$\hat{L}_{ik}[n] \approx \hat{L}_{ik}^{(0)}[n] + \sum_{h=1}^{N_{\text{pat}}} \sum_{j=1}^{N_{\text{dir}}} \hat{R}_{hj,ik}[n] \circledast \hat{L}_{hj}[n]. \quad (3.16)$$

Here, $\circledast$ denotes the discrete-time convolution. Throughout this chapter, hatted symbols denote discretized quantities. Injection and detection can also be discretized accordingly, as we show later. The discrete ARE, injection, and detection constitute the ART system, which converges to its continuous counterpart as the discretization becomes finer.

Delay Network Interpretation — ART can be interpreted as belonging to the family of delay networks, filter structures that recursively interconnect multiple parallel delay lines. We derive the explicit relationship in Appendix 3.A. Also belonging to this family are widely used artificial reverberation algorithms, such as the feedback delay network (FDN) [101, 102] and scattering delay network (SDN) [99]. However, ART differs in several aspects, including that it models energy rather than pressure and operates at a much larger scale. As an example, consider a room geometry mesh with $N_{\text{pat}} = 400$ surface patches and $N_{\text{dir}} = 256$ direction bins. The total number of parallel delay lines is given by the number of radiances, reaching roughly $N_{\text{rad}} \approx 100\text{k}$, orders of magnitude more than the tens of delay lines typical of FDNs and SDNs.

Frequency-domain ART — Rather than solving the time-domain ARE (3.16) directly, one can solve its frequency-domain counterpart to accelerate the radiance computation on a GPU. Consider a stack of all discrete radiances in the z -domain; we denote such a collection with a boldface letter:

$$\hat{\mathbf{L}}(z) = [\hat{L}_{11}(z), \hat{L}_{12}(z), \dots, \hat{L}_{N_{\text{pat}}N_{\text{dir}}}(z)]^T. \quad (3.17)$$

Similarly, let $\hat{\mathbf{R}}(z)$ denote the matrix collecting all discretized reflection kernel

elements. In this notation, the frequency-domain counterpart of the discrete ARE (3.16) reads

$$\hat{\mathbf{L}}(z) \approx \hat{\mathbf{L}}^{(0)}(z) + \hat{\mathbf{R}}(z)\hat{\mathbf{L}}(z), \quad (3.18)$$

and the corresponding Neumann series (3.2 and 3.3) can also be written as [25]

$$\hat{\mathbf{L}}(z) \approx \hat{\mathbf{L}}^{(0)}(z) + \sum_{k=1}^{\infty} \hat{\mathbf{L}}^{(k)}(z), \quad (3.19a)$$

$$\hat{\mathbf{L}}^{(k)}(z) = \hat{\mathbf{R}}(z)\hat{\mathbf{L}}^{(k-1)}(z). \quad (3.19b)$$

Now, to obtain a length- T echogram, we first solve the above z -domain ARE (3.18) at the uniform samples on the upper half of the unit circle, i.e.,

$$z = z_{T,f} = \exp \frac{2\pi j f}{T}, \quad f = 0, \dots, \lfloor T/2 \rfloor, \quad (3.20)$$

where $j = \sqrt{-1}$. Each solution can be obtained either by (i) direct inversion, i.e., explicitly forming the inverse matrix as in most prior differentiable delay networks [83, 173, 174], (ii) direct solve, i.e., solving the linear system using factorization methods [167], or (iii) computing the Neumann series (3.19) truncated to a finite order N_{order} . All three benefit from GPU parallelism, since each frequency bin has an independent linear system.

For DART, we choose the truncated Neumann series, since direct inversion is too costly at this scale. The series approach also allows us to leverage the sparsity of the reflection kernel, significantly reducing the computation and memory costs (see the following section). Note that explicit time-domain recursion [175, 176] is also possible, but requires $T \gg N_{\text{order}}$ steps, each involving convolutions rather than elementwise multiplications.

After solving the frequency-domain systems, we apply the inverse Fast Fourier Transform (IFFT) to obtain time-domain signals that approximate the first T samples of the discrete radiances $\hat{\mathbf{L}}[n]$. The source of error is time aliasing, where the tails $n \geq T$ of the original signals are folded back into the range $0 \leq n < T$ [177]. Therefore, unless additional techniques are applied, a sufficient number of samples T is required to compute the echogram $\hat{E}[n]$ accurately.

3.3 Methods

We aim to optimize the material properties of the surface patches. However, under the current ART formulation, every optimization step updates the material properties, requiring the discrete reflection kernel (3.14) to be recomputed, which is prohibitively expensive. Also, convolution with the kernel (3.16 or 3.18) is costly and memory-intensive. With the example from Section 3.2 ($N_{\text{rad}} \approx 100\text{k}$), storing the dense reflection kernel alone would require roughly $N_{\text{rad}}^2 \approx 40\text{GB}$ of GPU memory per discrete-time index n in single floating point precision. To address this challenge, we introduce several techniques that together minimize the optimization cost and make DART feasible. See Figure 3.2 for an overview.

3.3.1 Kernel Decomposition

Our first key ingredient is a kernel decomposition that decouples the effects of the fixed room geometry from the learnable material properties, allowing us to precompute the former before optimization.

Delay Decomposition — First, following prior ART methods [25, 163, 164], we separate the propagation delay as

$$\hat{R}_{hj,ik}[n] \approx \hat{D}_{hj}[n] \cdot \hat{S}_{hj,ik}. \quad (3.21)$$

$\hat{D}_{hj}[n]$ denotes a discrete delay signal for the radiance $\hat{L}_{hj}[n]$ and $\hat{S}_{hj,ik}$ is a time-independent, purely scalar-valued matrix. Factoring out the time-dependent propagation delay significantly lowers the computational cost, reducing the number of convolutions from N_{rad}^2 to N_{rad} .

Each discrete delay is realized with a linear fractional delay [178] to ensure nonnegativity of energy quantities, given as

$$\hat{D}_{hj}[n] = a_{hj}\delta[n - \lfloor \hat{d}_{hj} \rfloor] + (1 - a_{hj})\delta[n - \lceil \hat{d}_{hj} \rceil], \quad (3.22)$$

where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ denote floor and ceil operations, respectively. Its linear weight

and delay length are given as

$$a_{hj} = \lceil \hat{d}_{hj} \rceil - \hat{d}_{hj}, \quad (3.23a)$$

$$\hat{d}_{hj} = \frac{\iint_{\mathcal{S}_{hj}} \iint_{\mathcal{A}_h} V_{\mathcal{A}}(x, \Theta) \|x - x'(x, \Theta)\| d\mathcal{A}(x) d\Theta}{c\Delta t \iint_{\mathcal{S}_{hj}} \iint_{\mathcal{A}_h} V_{\mathcal{A}}(x, \Theta) d\mathcal{A}(x) d\Theta}. \quad (3.23b)$$

Here, $V_{\mathcal{A}}(x, \Theta)$ is an indicator function that equals 1 if the ray emitted from x in direction Θ hits any point on the geometry $\mathcal{A}$. $x'(x, \Theta) \in \mathcal{A}$ denotes a surface point where the ray first hits. Hence, the delay length $\hat{d}_{hj}$ is the average travel delay of the valid rays emitted from patch $\mathcal{A}_h$ within direction bin $\mathcal{S}_{hj}$ that hit any point on the geometry $\mathcal{A}$.

Visibility-Material Decomposition — After the above decomposition, the remaining scalar-valued matrix is given as

$$\begin{aligned} \hat{S}_{hj,ik} = & \iint_{\mathcal{S}_{ik}} \iint_{\mathcal{A}_i} \iint_{\mathcal{S}_{hj}} \iint_{\mathcal{A}_h} V(x, x') G(x, x') \\ & \rho(v_{x' \rightarrow x}, \Omega; x') \delta(v_{x \rightarrow x'} - \Theta) d\mathcal{A}(x) d\Theta \frac{d\mathcal{A}(x')}{|\mathcal{A}_i|} \frac{d\Omega}{|\mathcal{S}_{ik}|}, \end{aligned} \quad (3.24)$$

which still requires the nested integration of multiple terms, including the BSDF. To reduce complexity further, we introduce an additional decomposition step (see Figure 3.3 for illustration):

$$\hat{S}_{hj,ik} \approx \sum_{l=1}^{N_{\text{dir}}} \hat{V}_{hj,il} \hat{M}_{il,ik}. \quad (3.25)$$

The first term is the *mean visibility matrix*, which quantifies the fraction of the propagated source radiance $\hat{D}_{hj}[n] * \hat{L}_{hj}[n]$ visible from the receiving patch $\mathcal{A}_i$ with direction bin $\mathcal{S}_{il}$. It is defined as

$$\hat{V}_{hj,il} = \iint_{\mathcal{S}_{il}} \iint_{\mathcal{A}_i} V_{hj}(x', \Phi) \frac{d\mathcal{A}(x')}{|\mathcal{A}_i|} \frac{d\Phi}{|\mathcal{S}_{il}|}, \quad (3.26)$$

where $V_{hj}(x', \Phi)$ is another indicator function that equals 1 if the ray emitted from x' in direction Φ will hit patch $\mathcal{A}_h$ with an incident direction falling within $\mathcal{S}_{hj}$. By aggregating all delayed source radiances attenuated by the mean visibility matrix,

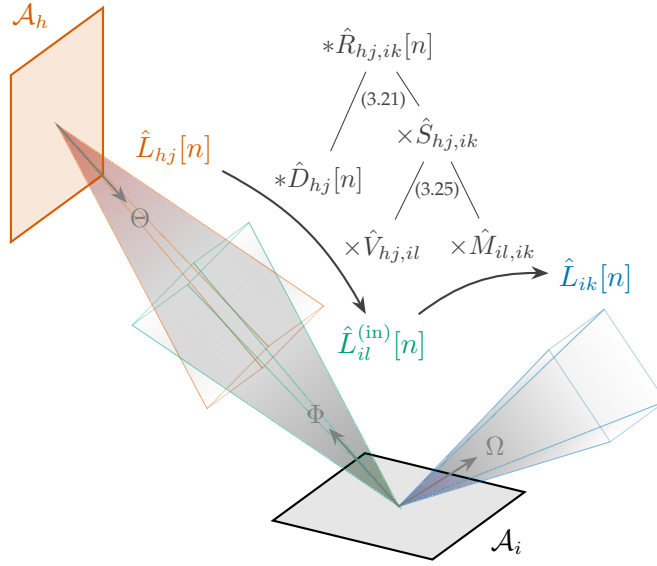


Figure 3.3: **Decomposition of the reflection kernel.** The tree shows the two-level factorization into the propagation delay $\hat{D}_{hj}[n]$, mean visibility matrix $\hat{V}_{hj,il}$, and material matrix $\hat{M}_{il,ik}$. The first step (3.21) separates the propagation delay from the reflection kernel, so that it is applied to each source radiance $\hat{L}_{hj}[n]$. The second step (3.25) factorizes the remaining matrix $\hat{S}_{hj,ik}$ into two matrices; each delayed radiance is first attenuated by the mean visibility matrix to yield the incident radiance $\hat{L}_{il}^{(\text{in})}[n]$, and then multiplied by the material matrix to yield the outgoing radiance $\hat{L}_{ik}[n]$. The direction bins of source, incident, and outgoing radiance are visualized in shades of orange, green, and blue, respectively, and arrows $\Theta \in \mathcal{S}_{hj}$, $\Phi \in \mathcal{S}_{il}$, $\Omega \in \mathcal{S}_{ik}$ mark the corresponding directions as used in the text.

we obtain an incident radiance $\hat{L}_{il}^{(\text{in})}[n]$ for each reflecting or transmitting patch $\mathcal{A}_i$ and incident direction $\mathcal{S}_{il}$.

The second term, the material matrix, models the effect of the BSDF and converts the incident radiance $\hat{L}_{il}^{(\text{in})}[n]$ into contributions to the outgoing radiance $\hat{L}_{ik}[n]$. It is defined as

$$\hat{M}_{il,ik} = \iint_{\mathcal{S}_{ik}} \iint_{\mathcal{S}_{il}} \rho_i(\Phi, \Omega) |\Phi \cdot \nu_i| d\Phi \frac{d\Omega}{|\mathcal{S}_{ik}|}. \quad (3.27)$$

As in the ARE discretization (3.13), both terms rely on a locally constant assumption: the BSDF ρ_i is uniform within each patch $\mathcal{A}_i$, and the mean visibility is constant within each directional bin. The same assumptions underlie the injection approximation introduced later (3.40); all such errors vanish as the discretization

becomes finer. Refer to Appendix 3.B.1 for the derivation.

The discrete delays $\hat{D}[n]$ and the mean visibility matrix $\hat{V}$ can be computed once and stored prior to optimization, as they only depend on the fixed room geometry.

3.3.2 Material Parameterization

The material matrix $\hat{M}$ still requires numerically integrating the BSDFs during optimization. We explore two strategies that sidestep this cost, corresponding to two variants of DART.

Unconstrained (UC) Variant — We can bypass the integration and directly learn the matrix entries, factorized into a reflection coefficient $\alpha_i \in [0, 1)$ per patch $\mathcal{A}_i$ and a lossless (energy-preserving) [165] matrix $\bar{M}$.

$$\hat{M}_{il,ik}^{(\text{UC})} = \alpha_i \bar{M}_{il,ik}, \quad \sum_{l=1}^{N_{\text{dir}}} \bar{M}_{il,ik} = 1. \quad (3.28)$$

The unconstrained approach is particularly useful when only approximate meshes are available, as is common in practice. Geometric errors in such meshes, whether missing or misplaced fine details, produce rich diffraction and scattering patterns that can be approximated with complex BSDFs on the simplified geometry [179, 180]. The maximal flexibility of the material matrix allows the unconstrained DART to learn such details.

Parametric (P) Variant — Alternatively, we can model each patch’s BSDF ρ_i as a convex combination of predefined N_{BSDF} BSDFs, each denoted as $\rho_{i,m}$, as follows,

$$\rho_i(\Phi, \Omega) = \alpha_i \sum_m \beta_{i,m} \rho_{i,m}(\Phi, \Omega), \quad \sum_{m=1}^{N_{\text{BSDF}}} \beta_{i,m} = 1, \quad (3.29)$$

where $\beta_{i,m} \in [0, 1)$ denotes the combination weights. Then, the corresponding material matrix also becomes a convex combination of the BSDF matrices.

$$\hat{M}_{il,ik}^{(\text{P})} = \alpha_i \sum_{m=1}^{N_{\text{BSDF}}} \beta_{i,m} \hat{M}_{il,ik,m}. \quad (3.30)$$

Each BSDF matrix follows the definition of the material matrix (3.27) and is given as

$$\hat{M}_{il,ik,m} = \iint_{\mathcal{S}_{ik}} \iint_{\mathcal{S}_{il}} \rho_{i,m}(\Phi, \Omega) |\Phi \cdot \nu_i| d\Phi \frac{d\Omega}{|\mathcal{S}_{ik}|}, \quad (3.31)$$

and it can be precomputed since each component BSDF $\rho_{i,m}$ is fixed. We let both reflection coefficients α_i and combination weights $\beta_{i,m}$ be learnable. As component BSDFs, we use the ideal specular and diffuse reflections [25]. For multi-room scenes, we also include specular and diffuse transmissions. Their definitions are given as

$$\rho_i^{(\text{DiffuseReflection})}(\Phi, \Omega) = \frac{1}{\pi} \mathbb{1}_{\text{sgn}(\Phi \cdot \nu_i) = \text{sgn}(\Omega \cdot \nu_i)}, \quad (3.32a)$$

$$\rho_i^{(\text{DiffuseTransmission})}(\Phi, \Omega) = \frac{1}{\pi} \mathbb{1}_{\text{sgn}(\Phi \cdot \nu_i) \neq \text{sgn}(\Omega \cdot \nu_i)}, \quad (3.32b)$$

$$\rho_i^{(\text{SpecularReflection})}(\Phi, \Omega) = \frac{\delta(\Phi + \Omega - 2\nu_i)}{|\Phi \cdot \nu_i|}, \quad (3.32c)$$

$$\rho_i^{(\text{SpecularTransmission})}(\Phi, \Omega) = \frac{\delta(\Phi + \Omega)}{|\Phi \cdot \nu_i|}. \quad (3.32d)$$

$\text{sgn}(\cdot)$ denotes the sign function and $\mathbb{1}$ is an indicator function, which equals 1 when the condition holds, 0 otherwise. This parametric DART is consistent with standard material choices in room acoustics modeling, providing more compact and interpretable parameters ($N_{\text{BSDF}} + 1$ parameters per patch, versus $N_{\text{dir}}^2 + 1$ for unconstrained DART).

3.3.3 Radiance Computation

Aliasing Suppression — For optimization, it is unnecessary to compute the full-length echogram because, after sufficient reflections, the sound field becomes diffuse, and the remaining response can be well approximated by an exponential decay. However, using a short echogram length T in the frequency domain leads to coarse sampling and introduces the aforementioned time aliasing. To mitigate this, we employ the recently proposed technique [181], sampling *outside* the unit circle:

$$z = z_{T,\gamma,f} = \gamma^{-1/T} \exp \frac{2\pi j f}{T}, \quad f = 0, \dots, \lfloor T/2 \rfloor. \quad (3.33)$$

Here, $\gamma \in (0, 1)$ denotes a constant circle radius that controls the amount of aliasing suppression. By applying this sampling scheme to the truncated series (3.19) with the decomposed kernel (3.21 and 3.25), we can in practice compute the radiance by

$$\hat{\mathbf{L}}[n] \approx \gamma^{-n/T} \cdot \text{IFFT} \left[\sum_{k=0}^{N_{\text{order}}} \hat{\mathbf{L}}^{(k)}(z_{T,\gamma,f}) \right], \quad (3.34)$$

where the k -th term is given as

$$\hat{\mathbf{L}}^{(k)}(z_{T,\gamma,f}) = \hat{\mathbf{M}}\hat{\mathbf{V}} \left[\hat{\mathbf{D}}(z_{T,\gamma,f}) \odot \hat{\mathbf{L}}^{(k-1)}(z_{T,\gamma,f}) \right] \quad (3.35)$$

where $\odot$ denotes elementwise multiplication. The frequency-domain initial radiances and delays can be obtained by

$$\hat{\mathbf{L}}^{(0)}(z_{T,\gamma,f}) = \text{FFT} \left[\gamma^{n/T} \hat{\mathbf{L}}^{(0)}[n] \right], \quad (3.36a)$$

$$\hat{\mathbf{D}}(z_{T,\gamma,f}) = \text{FFT} \left[\gamma^{n/T} \hat{\mathbf{D}}[n] \right]. \quad (3.36b)$$

Sampling outside the unit circle damps the problematic tails $n \geq T$ with an exponentially decreasing window $\gamma^{n/T}$, thereby reducing the aliasing by more than a factor of γ . We leave the derivation of this error bound in Appendix 3.C.

Leveraging Sparsity — We implement the core computations using sparse tensors and operations. For instance, prior to optimization, we identify discrete radiances that never send or receive energy and therefore never contribute to the echogram. We can remove corresponding columns and rows from the tensors, which reduces the number of discrete radiances to $\bar{N}_{\text{rad}} < N_{\text{rad}}$ and yields more compact tensors (denoted with a subscript r). Furthermore, even after the reduction, the mean visibility and material matrices are highly sparse; their numbers of nonzeros are denoted as $\bar{N}_{\text{vis}}, \bar{N}_{\text{mat}} \ll \bar{N}_{\text{rad}}^2$, respectively. Hence, we compute the Neumann series terms (3.35) with sparse matrix multiplications spmm as

$$\hat{\mathbf{L}}_r^{(k)} = \text{spmm}(\hat{\mathbf{M}}_r, \text{spmm}(\hat{\mathbf{V}}_r, \hat{\mathbf{D}}_r \odot \hat{\mathbf{L}}_r^{(k-1)})), \quad (3.37)$$

where the tensor shapes are given as

$$\hat{\mathbf{L}}_{\mathbf{r}}^{(k)}, \hat{\mathbf{L}}_{\mathbf{r}}^{(k-1)}, \hat{\mathbf{D}}_{\mathbf{r}} \in \mathbb{C}^{\bar{N}_{\text{rad}} \times \lfloor T/2+1 \rfloor}, \quad \hat{\mathbf{M}}_{\mathbf{r}}, \hat{\mathbf{V}}_{\mathbf{r}} \in \mathbb{R}^{\bar{N}_{\text{rad}} \times \bar{N}_{\text{rad}}}. \quad (3.38)$$

We use two consecutive sparse matrix multiplications because their product would be substantially denser than either factor.

Complexity Analysis — We applied factorization and sparse operations to the single dense reflection kernel. As a result, the computational and memory complexities of a single reflection step in the frequency domain are improved, respectively, to:

$$\mathcal{O}(N_{\text{rad}}^2 T) \rightarrow \mathcal{O}(\bar{N}_{\text{rad}} T) + \mathcal{O}(\bar{N}_{\text{vis}} T) + \mathcal{O}(\bar{N}_{\text{mat}} T), \quad (3.39a)$$

$$\mathcal{O}(N_{\text{rad}}^2 T) \rightarrow \mathcal{O}(\bar{N}_{\text{rad}} T) + \mathcal{O}(\bar{N}_{\text{vis}}) + \mathcal{O}(\bar{N}_{\text{mat}}), \quad (3.39b)$$

where the three terms correspond to the complexities of applying delays, mean visibility, and material matrices, respectively. The actual sparsity ratios depend on the room geometry and discretization resolution; however, in practice, a significant fraction of discrete radiances can be pruned, and both matrices are highly sparse relative to $\bar{N}_{\text{rad}}^2$, with densities on the order of 10^{-4} to 10^{-3} . The visibility matrix is typically the sparser of the two by roughly an order of magnitude, since most patch-direction pairs lack line-of-sight connectivity.

3.3.4 Injection and Detection

Injection — We obtain the discrete initial radiance $\hat{L}_{ik}^{(0)}[n]$ by discretizing its continuous counterpart (3.7). Using analogous locally constant approximations (refer to Appendix 3.B.2 for the derivation), we can decompose it as

$$\hat{L}_{ik}^{(0)}[n] \approx \sum_{l=1}^{N_{\text{dir}}} \hat{M}_{il,ik} \frac{\hat{P}_{il}^{(\text{in})}[n]}{\hat{G}_{il}} \quad (3.40)$$

where $\hat{P}_{il}^{(\text{in})}[n]$ is the incident power from the source arriving at patch $\mathcal{A}_i$ from direction bin $\mathcal{S}_{il}$, defined as

$$\hat{P}_{il}^{(\text{in})}[n] = \hat{D}_{s,il}[n] \iint_{\mathcal{S}} \Gamma_s(\Theta, o_s) V_{il}(x_s, \Theta) \frac{d\Theta}{4\pi}. \quad (3.41)$$

Here, $\hat{D}_{s,il}[n]$ is a discrete delay signal encoding the average propagation delay from the source s to patch $\mathcal{A}_i$ in direction bin $\mathcal{S}_{il}$, and the integral accumulates the directivity-weighted visibility over all ray directions. In practice, the incident power is computed via Monte Carlo, by shooting rays from the source and aggregating their contributions into the corresponding patch-direction bins. The dividing term $\hat{G}_{il}$ is a precomputable constant, called the integrated geometry, which converts the incident power into incident radiance.

$$\hat{G}_{il} = |\mathcal{A}_i| \iint_{\mathcal{S}_{il}} |\Phi \cdot \nu_i| d\Phi. \quad (3.42)$$

The material matrix $\hat{M}_{il,ik}$ then scatters the incident radiance into the initial (outgoing) radiance.

We apply the material matrix rather than an explicit BSDF evaluation for the first reflection, departing from prior ART methods. In those methods, the BSDF is sampled directly via ray tracing during injection; during optimization, this would require maintaining the BSDF explicitly and recomputing the material matrix (3.27) from it at every step to keep the system consistent. By using the material matrix directly for injection, we avoid this recomputation. This introduces a minor approximation but also reduces variance from ray tracing stochasticity, making DART less sensitive to the number of injection rays.

Detection — Similarly, discretizing the continuous detection (3.8) yields

$$\hat{E}_L[n] \approx \sum_{i=1}^{N_{\text{pat}}} \sum_{k=1}^{N_{\text{dir}}} \hat{W}_{r,ik}[n] \otimes \hat{L}_{ik}[n] \quad (3.43)$$

where $\hat{W}_{r,ik}[n]$ is the detection weight for patch $\mathcal{A}_i$ and direction bin $\mathcal{S}_{ik}$, defined as

$$\hat{W}_{r,ik}[n] = \hat{D}_{r,ik}[n] \iint_{\mathcal{S}} \Gamma_r(\Theta, o_r) V_{ik}(x_r, \Theta) d\Theta. \quad (3.44)$$

As in the injection, $\hat{D}_{r,ik}[n]$ encodes the propagation delay from patch $\mathcal{A}_i$ to the receiver r , and the integral accumulates the directivity-weighted visibility. The detection weight is convolved with the radiance $\hat{L}_{ik}[n]$ and summed over all patch-direction pairs to yield the detected power $\hat{E}_L[n]$. In practice, we use the sparse radiance discussed in Section 3.3.3 for both injection and detection, which we omit for brevity. Finally, after computing the direct arrival $\hat{E}_D[n]$, we obtain the echogram $\hat{E}[n]$, which is multiplied by an auxiliary learnable gain $g \in \mathbb{R}^+$ to match the scale of the target echograms.

3.3.5 Learnable Directivity

In practical scenarios, the source directivity Γ_s (or receiver directivity Γ_r) may be unknown. We employ a learnable, axially symmetric directivity module inspired by DiffRIR [28]. We introduce K learnable parameters $\kappa_1, \dots, \kappa_K$ at equispaced angles ϕ_k from 0 to π . For a ray with outgoing direction Θ , let θ be its angle from the principal axis o . The directivity is a softmax-weighted sum of κ_k , with weights determined by the angular distance $|\theta - \phi_k|$ and a temperature $\lambda > 0$:

$$\Gamma(\Theta, o) = \sum_{k=1}^K \frac{\exp(-\lambda|\theta - \phi_k|)}{\sum_{l=1}^K \exp(-\lambda|\theta - \phi_l|)} \kappa_k. \quad (3.45)$$

3.4 Experimental Setup

3.4.1 Benchmarks

We evaluate DART with two real-world datasets. First, we use the Hearing Anything Anywhere (HAA) dataset [28]. The HAA dataset comprises four scenes: Classroom, Hallway, Dampened, and Complex. Each scene features several hundred RIR measurements with varying receiver positions, all from a single fixed source. We follow the same split as initially proposed: 12 RIRs for training, half of the remaining RIRs for validation, and the remaining ones for testing. The HAA dataset also provides rough room geometry meshes, which are used for DART and other baselines; Figure 3.4 shows them.

While the HAA dataset serves as an excellent benchmark, it also has some

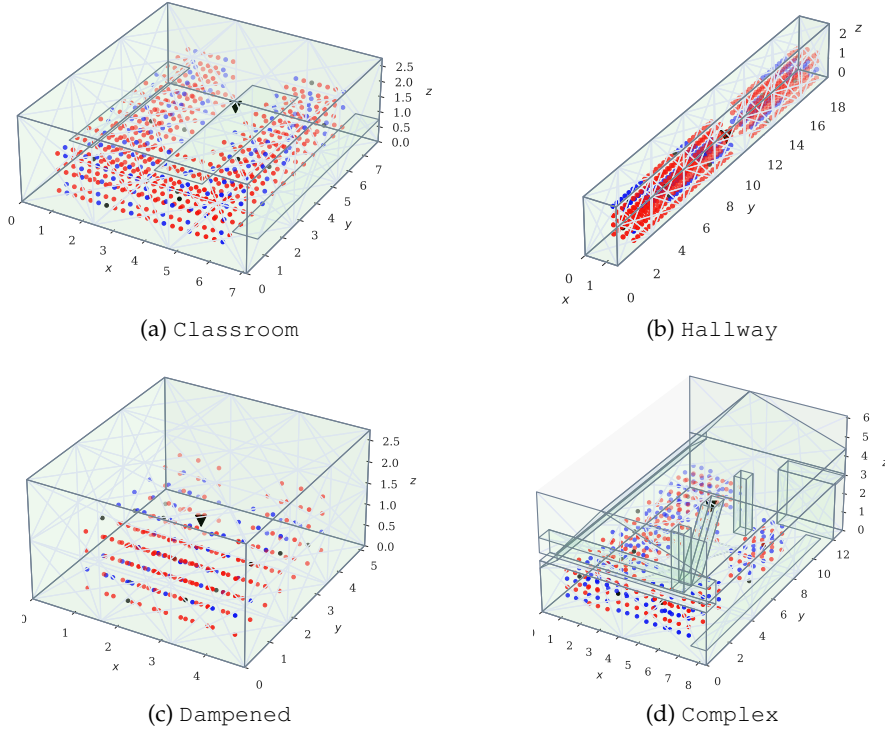


Figure 3.4: **Room geometries of the Hearing Anything Anywhere (HAA) dataset.** Triangles indicate source positions and small dots indicate receiver positions. Black, blue, and red colors denote train, validation, and test data, respectively.

weaknesses: each scene consists of only one room, and is measured only with a single fixed source position and orientation. This motivates us to complement it with a more challenging Coupled Room (CR) dataset [182]. The CR dataset also comprises four scenes: *Office* $\rightarrow$ *Anechoic*, *Office* $\rightarrow$ *Stairwell*, *MeetingRoom* $\rightarrow$ *Hallway*, and *Office* $\rightarrow$ *Kitchen*. Each scene, denoted in the form *RoomA* $\rightarrow$ *RoomB*, has two rooms, *RoomA* and *RoomB*. 101 receivers are positioned around the transition area from *RoomA* to *RoomB*, and four sources are considered with different positions and orientations, two in each room. See Figure 3.5 for an example, and Figure 3.6 for qualitative results on this split. Since the CR dataset provides a floor plan and the ceiling height for each scene, we construct each room geometry mesh ourselves. Similar to the HAA dataset, these meshes are coarser than the actual room geometries, omitting interior objects and

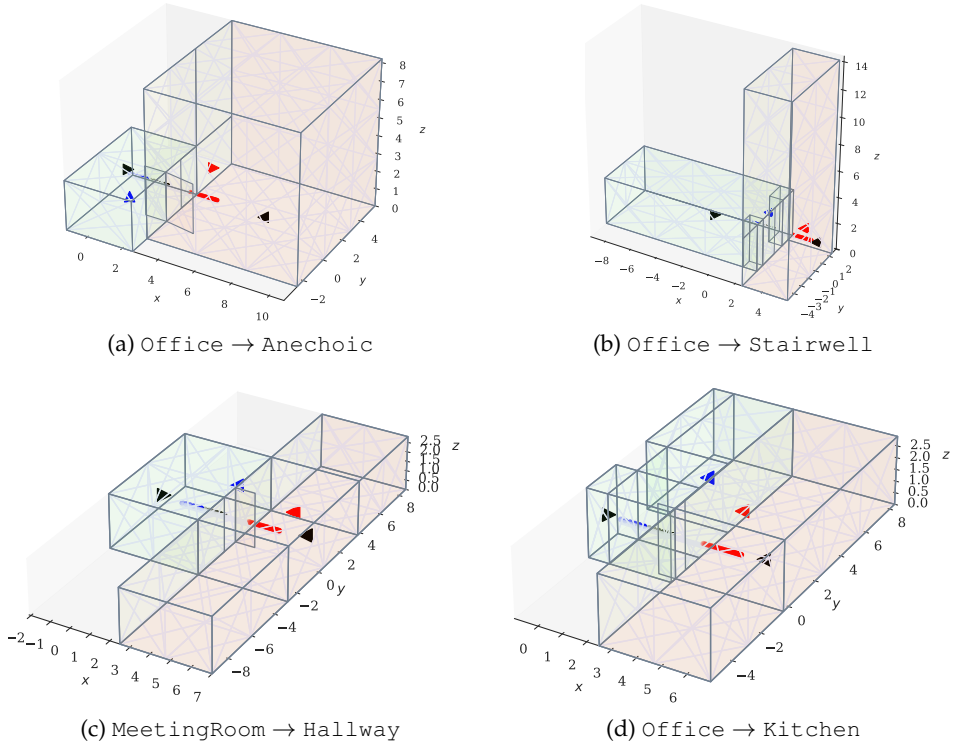


Figure 3.5: **Room geometries of the Coupled Room (CR) dataset.** The coupled room split is visualized, with RoomA shaded in green and RoomB shaded in red. Triangles indicate source positions and small dots indicate receiver positions. Black, blue, and red colors denote train, validation, and test data, respectively.

fine architectural detail.

For the CR dataset, we consider two data splits to analyze the models' behavior under different scenarios. In the first one, called *random* split, we sample three receiver positions for each source (hence, a total of 12 samples) and use them for training. We sample another 12 non-overlapping source–receiver pairs for validation in the same way, and the remaining samples form a test set. The other one, called *unseen* split, tests the models under completely unseen source/receiver pairs. As shown in Figure 3.5, the training uses two sources, each with six receiver positions (again, a total of 12 samples). The validation uses one remaining source in RoomA with six different receivers. For the testing, we use the remaining source in RoomB with 32 different receivers. Hence, this split tests whether the

models can generalize and accurately estimate sound propagation, beyond merely interpolating the observations.

All RIRs are resampled to a 16kHz sampling rate. The sampling rate of the discrete radiance is set to 1kHz, as it has been shown to be sufficiently close to the continuous-time ART [165]. We obtain each echogram by squaring the corresponding RIR and summing blocks of 16 consecutive samples. We consider RIRs or echograms of 0.32s length, i.e., $T = 320$.

3.4.2 Metrics

We compare the predicted echograms and their corresponding ground truths with the sample-level l_1 distance (L1). We also consider three reverberation parameter distances: reverberation time (T60, in %), early decay time (EDT, in seconds), and clarity (C50, in dB). Each model is validated at 25 regular intervals during its optimization, and then the best checkpoint is selected for the final test.

3.4.3 Implementation Details

Discretization — For surface discretization, we subdivide the room geometry into small triangle patches. Specifically, we subdivide the raw parallelogram mesh into approximately equal-area parallelograms, ensuring no edge exceeds a specified maximum length, and then split each into four triangles along its diagonals. The resulting patch count ranges from 120 (Hallway, HAA) to 260 (Office $\rightarrow$ Anechoic, CR). We let each subpatch have its own independent material parameters. Each patch’s direction sphere $\mathcal{S}$ is partitioned into a grid of $N_{\text{azi}} = 12$ azimuths and $N_{\text{ele}} = 12$ elevations, i.e., $N_{\text{dir}} = 144$, with equal solid angles. For further analysis of how these surface and directional resolutions affect optimization, see Appendix 3.E.

Radiance Computation — We set the truncated Neumann series order N_{order} for each scene based on its geometric scale. We divide the total propagation length ($\approx 110\text{m}$ for a 0.32s echogram) by the shortest room dimension to upper-bound the number of repeated reflections, which sets the truncation order. The resulting values range from 25 for the Complex scene to 72 for the Hallway scene. We

empirically observed that this choice was sufficient to model the dominant energy contributions within the echogram duration; increasing the order beyond these values produced only negligible changes in the predicted echograms. For the time aliasing suppression, we use $\gamma = 10^{-2}$, which reduces the aliasing error by more than 20dB. We found this value to be sufficient, and using a smaller γ amplified numerical noise due to the exponentially increasing window (3.34), consistent with prior observations [181].

Ray Tracing Monte Carlo — Several ART components are computed via Monte Carlo ray tracing. For the mean visibility matrix, we sample 10×10 points on each patch and shoot 4096 rays from each point, totaling $N_{\text{precompute}} = 100 \times 4096 \times N_{\text{pat}}$ rays; the propagation delays are obtained from the same traces. For injection and detection, we shoot 10k rays each, using stratified sampling on the patch and direction grid.

Directivity — As both datasets employ directional speakers and omnidirectional microphones, we learn only the source directivity Γ_s , with scale $\lambda = 8$ and $K = 128$ directions, and omit the receiver directivity Γ_r .

Core Libraries — We use `PyTorch` [183] for the implementation. For efficient sparse computation and backpropagation, we rely on `pytorch_sparse`¹ and `pytorch_scatter`². `NVIDIA Warp` [184] is used for efficient ray tracing.

3.4.4 Optimization

Learning Objective — As a main optimization objective, we use an echogram loss defined as

$$\mathcal{L}_{\text{Echogram}} = \mathcal{L}_{\text{NMSE}} + \mathcal{L}_{\text{EDC}}. \quad (3.46)$$

¹github.com/rustyls/pytorch_sparse

²github.com/rustyls/pytorch_scatter

$\mathcal{L}_{\text{NMSE}}$ is a normalized mean square error (NMSE) between the predicted echogram $\hat{E}^{\text{pred}}[n]$ and its ground truth $\hat{E}^{\text{true}}[n]$.

$$\mathcal{L}_{\text{NMSE}} = \frac{\left\| \hat{E}^{\text{pred}}[n] - \hat{E}^{\text{true}}[n] \right\|^2}{\left\| \hat{E}^{\text{true}}[n] \right\|^2}. \quad (3.47)$$

$\mathcal{L}_{\text{EDC}}$ is a normalized mean absolute error of the energy decay curve (EDC), an inverse cumulative sum of an echogram [74].

$$\mathcal{L}_{\text{EDC}} = \frac{\left\| \text{EDC}(\hat{E}^{\text{pred}})[n] - \text{EDC}(\hat{E}^{\text{true}})[n] \right\|_1}{\left\| \text{EDC}(\hat{E}^{\text{true}})[n] \right\|_1}. \quad (3.48)$$

The NMSE loss $\mathcal{L}_{\text{NMSE}}$ focuses on the high-energy region, i.e., direct arrival and early reflections, while the EDC loss $\mathcal{L}_{\text{EDC}}$ focuses on the reverberation decay.

Parameterization and Initialization — All learnable parameters are stored as unconstrained tensors and mapped to valid ranges via nonlinear activations (e.g., sigmoid for the reflection coefficients α). We initialize the reflection coefficients α to 0.5, source directivity Γ_s as omnidirectional, and the global gain g to 1. The combination coefficients β of parametric DART are initialized to have a 0.95 : 0.05 reflection-to-transmission ratio and 0.8 : 0.2 diffuse-to-specular ratio, and the lossless matrix $\bar{M}$ of unconstrained DART is initialized to what the parametric model’s initial combination coefficients β would yield. With these settings, both variants initially predict identical echograms for the same source–receiver pair.

Optimization Setup — We use the AdamW optimizer [185] with a batch size of 1. We employ a base learning rate of 10^{-2} for the reflection coefficients α , source directivity Γ_s , and global gain g , and one fourth of the base learning rate for the lossless matrix $\bar{M}$ and combination coefficients β , so that fine scattering patterns are learned after coarser parameters. The parameter learning rates are annealed to zero using a cosine scheduler. We optimize DART for 25k steps, which takes no more than 2.5 hours with a single NVIDIA RTX4090 GPU.

Table 3.1: **Main benchmark results.** Lower is better for all metrics. For each benchmark setup and metric, the best result is highlighted in bold and blue, and the second-best result is highlighted in pink.

Methods	HAA					CR (random)					CR (unseen)				
	L1	T60	EDT	C50		L1	T60	EDT	C50		L1	T60	EDT	C50	
Nearest neighbor	0.895	7.53	0.220	1.60		0.891	5.01	0.179	1.50		—	—	—	—	
Linear interpolation	0.892	7.32	0.206	1.57		0.983	4.85	0.184	1.57		—	—	—	—	
DiffRIR [28]	0.745	8.13	0.160	1.65		0.772	14.06	0.217	1.86		5.930	42.88	0.349	4.46	
INRAS [80]	0.989	10.75	0.331	2.78		1.838	18.34	0.328	3.03		6.726	61.22	1.934	4.75	
w/ echogram	0.822	8.60	0.232	1.81		2.574	15.34	0.428	3.17		1.757	22.17	0.542	2.93	
AVR [142]	0.972	9.18	0.287	2.09		1.049	9.32	0.329	2.53		2.985	25.46	1.030	9.84	
w/ echogram	0.948	8.48	0.259	2.18		1.236	8.37	0.277	2.86		3.509	15.35	0.647	5.03	
DART (UC)	0.642	7.00	0.115	1.12		0.510	5.07	0.080	0.72		0.831	19.90	0.073	0.69	
DART (P)	0.638	15.95	0.097	1.84		0.577	5.82	0.083	0.77		0.900	24.51	0.125	1.82	

3.4.5 Baselines

Low Anchors — Following prior works [80, 142, 28], we use nearest neighbor and linear interpolation as the simplest baselines.

DiffRIR — DiffRIR is a differentiable image source method (ISM) introduced with the HAA dataset [28]. It models each RIR as an early specular component and a late reverberation residual. The early component is computed via the ISM, where the energy of each image source path is determined by surface reflection coefficients, source directivity, and air absorption. The late reverberation is a learnable residual signal shared across the scene. We adopt the official code with minor hyperparameter adjustments, e.g., sampling rate, to match our benchmark setup.

INRAS — Inspired by ART, INRAS [80] employs three neural network encoders corresponding to injection, reflections, and detection. These encoders take “bounce points” sampled from the room geometry, or their relative positions with respect to the source and receiver, as input. Then, a neural decoder aggregates the latent features from the encoders and predicts the RIR. Starting from the official implementation, we adapt it to our benchmark setup: bounce points are taken from the (subdivided) mesh patch centers, source orientation is provided to the injection encoder, and the stereo “listener module” is removed for mono output. We train INRAS with a batch size of 4, a learning rate of 5×10^{-5} , and 125k optimization steps.

AVR — Acoustic volume rendering (AVR) [142] extends differentiable volume rendering in vision [186] to acoustics by accounting for the nonnegligible speed of sound. An RIR is modeled as the aggregation of contributions from all incoming directions at the receiver. For each direction, a ray is cast into the scene and contributions are accumulated along the ray based on a neural-network-predicted volumetric density and source signal, accounting for propagation delay, distance-based attenuation, and transmittance. Unlike the other baselines, AVR does not take explicit room geometry as input. Since AVR tends to overfit on sparse measurements, we train it for 2.5k steps with a 2×10^{-4} learning rate.

Echogram Variants — The neural baselines, INRAS and AVR, were originally pro-

posed for RIR prediction, which could be more challenging than echogram prediction. Therefore, for a fair comparison, we also train echogram variants, modified to predict energy quantities, e.g., using softplus activation for non-negativity, and trained with the same echogram loss $\mathcal{L}_{\text{Echogram}}$ as DART. We denote these variants with “w/ echogram” or “(E).” We do not include an echogram variant of DiffRIR, as it did not yield improved results in preliminary experiments. We keep the same hyperparameters for the echogram variants, except for AVR (E), which is trained for 50k steps, longer than the RIR-predicting AVR, as it exhibits less overfitting.

3.5 Results

3.5.1 Main Results

Table 3.1 summarizes the average evaluation results for each benchmark.

Both DART variants perform well across all three benchmarks, with a wider margin over the baselines on the more challenging CR benchmarks. The unconstrained (UC) variant generally achieves the strongest results, while the parametric (P) variant remains competitive despite its constrained material model. For reverberation time, the results are more mixed: the unconstrained variant does not always lead but remains close, while the parametric variant occasionally struggles. DART’s advantage is largest in the remaining metrics, which reflect acoustic characteristics that vary considerably with source and receiver position and orientation.

The neural baselines, INRAS and AVR, struggle to surpass even simple nearest-neighbor and linear interpolation. While this contrasts with prior results on larger-scale benchmarks [147, 148] where orders of magnitude more data were available, it is consistent with more recent findings under sparse conditions [28, 149], suggesting that these models lack sufficient inductive bias to produce physically plausible predictions from limited data. The echogram variants generally show slightly better results than their RIR counterparts on the HAA and CR (unseen) benchmarks, while the comparison is mixed on CR (random).

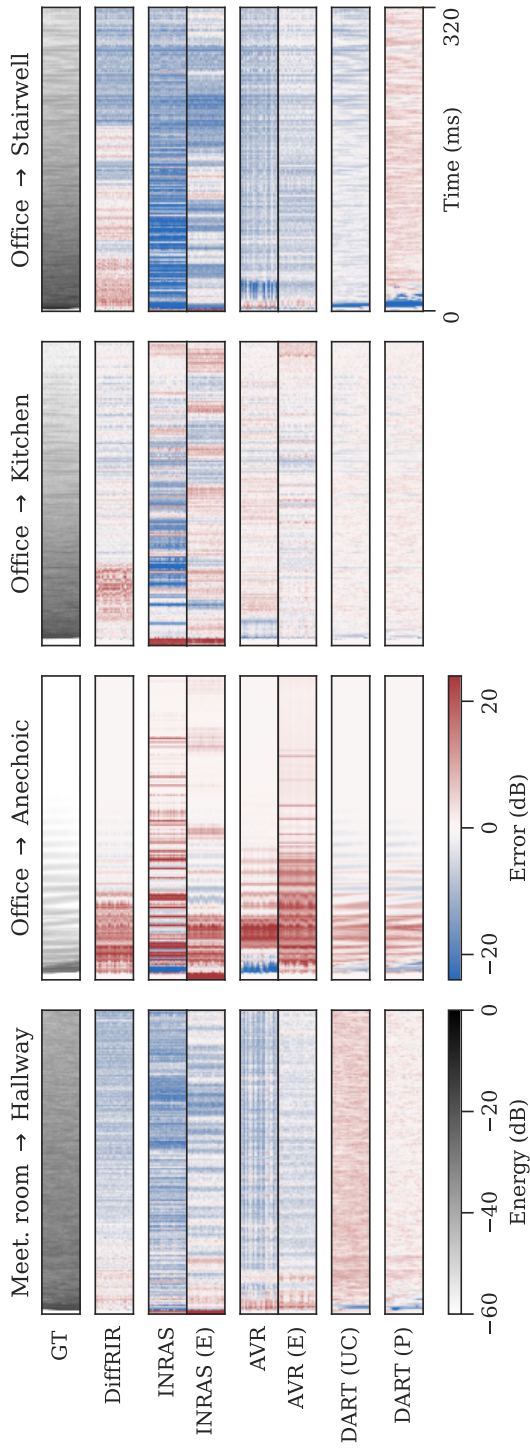


Figure 3.6: **Echogram residuals on the CR dataset, unseen split.** The top row shows all ground-truth test echograms, in decibels $10 \log_{10} \hat{E}^{\text{true}}[n]$, stacked vertically. For each model, the signed residuals of its predictions, $\mathcal{E} = 10 \log_{10} \hat{E}^{\text{pred}}[n] - 10 \log_{10} \hat{E}^{\text{true}}[n]$, are shown. All residual plots share the same diverging colormap with fixed limits of $\pm 24\text{dB}$, centered at zero. White corresponds to a perfect match (zero error), while red and blue indicate over- and underestimation of the echogram, respectively.

3.5.2 Qualitative Analysis

The CR (unseen) setup is the most challenging among the three benchmarks, as it requires models to generalize to entirely new source and receiver regions. We examine its test results in detail. In Figure 3.6, for each scene, the test echograms (in decibels) are vertically stacked and visualized as a matrix, where each echogram corresponds to a row. Then, signed residuals of the model predictions are shown, each defined as

$$\mathcal{E} = 10 \log_{10} \hat{E}^{\text{pred}}[n] - 10 \log_{10} \hat{E}^{\text{true}}[n]. \quad (3.49)$$

It reveals that the neural baselines, regardless of the optimization target (RIR or echogram), fail to accurately reconstruct both early and late parts of the echogram. For example, INRAS, despite its architecture being inspired by ART and having access to geometry and source/receiver positions, was unable to estimate direct arrivals, possibly because the corresponding delay lengths were unseen during training. Among the baselines, DiffRIR performs best, owing to the physical grounding of its ISM backbone. However, its early reflection model captures only specular paths, missing scattered energy, while its single shared residual for late reverberation cannot adapt to the varying reverberation decays across different source–receiver pairs. In contrast, DART, regardless of the parameterization method, consistently achieves lower errors than any baseline.

We observe that the baselines especially struggle at the `Office` $\rightarrow$ `Anechoic` and `Office` $\rightarrow$ `Stairwell` scene. These scenes comprise two coupled rooms with drastically different acoustic properties, e.g., `Anechoic` having much lower reverberation time than `Office`. The baselines, trained on measurements from receivers only in `Office`, did not capture this and overestimated the decay. DART, by contrast, recovered markedly different reflection coefficients for the two rooms and, as a result, retained its accuracy for those challenging scenes. The optimized per-patch reflection coefficients $\alpha_1, \dots, \alpha_{N_{\text{pat}}}$ of the parametric variant are visualized in Figure 3.7, where the two rooms of these scenes take on clearly different values. For the other two scenes where all the baselines performed relatively well, DART predicted nearly homogeneous reflection coefficients across the two rooms. Notably, DART recovered this difference without any receiver inside `RoomB`, e.g., `Anechoic` in `Office` $\rightarrow$ `Anechoic`, during optimization.

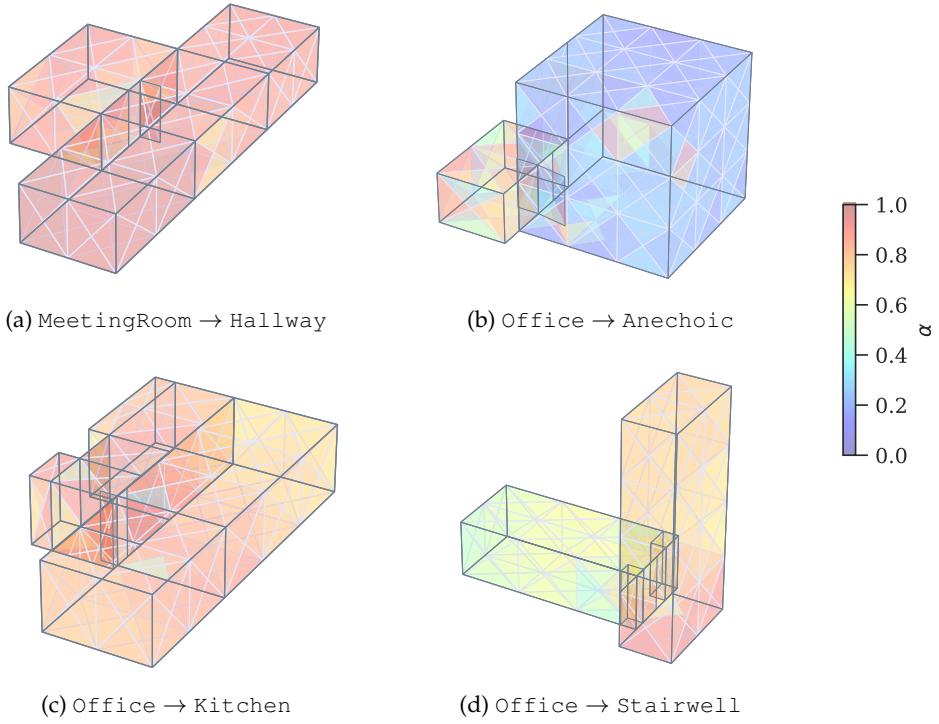


Figure 3.7: **Optimized reflection coefficients of parametric DART, CR unseen split.** Each mesh shows the optimized per-patch reflection coefficients $\alpha_1, \dots, \alpha_{N_{\text{pat}}}$ for one Coupled Room (CR) scene. Higher reflection coefficients (more reverberant) are shown in red and lower reflection coefficients (more absorptive, anechoic) in blue.

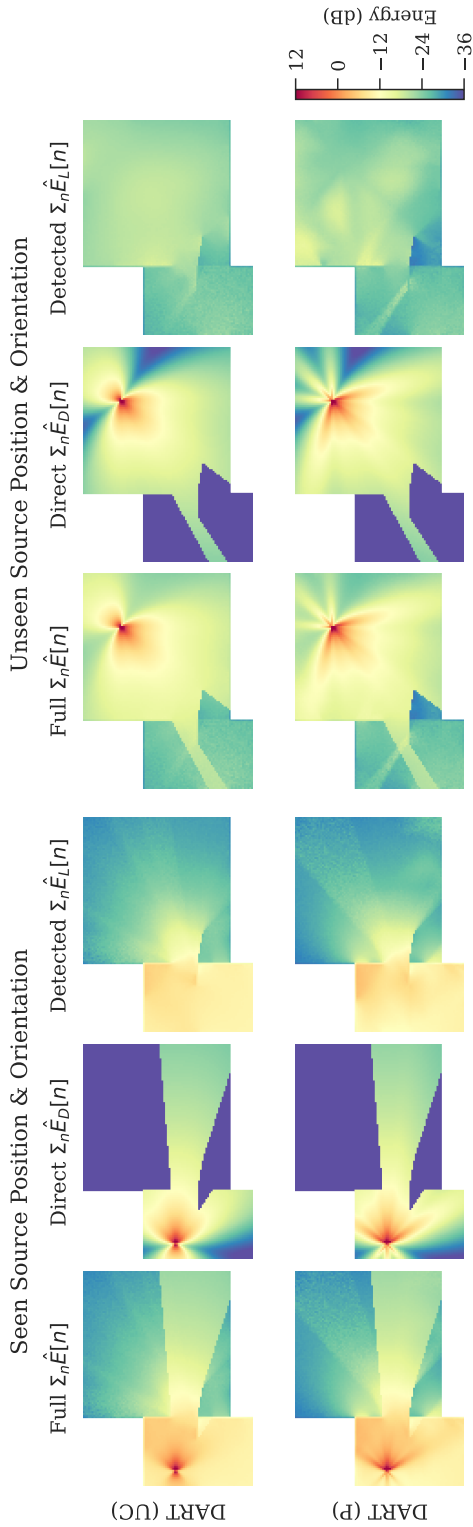


Figure 3.8: **Spatial energy distribution predicted by DART.** Results are shown for two different source configurations. The first (left three columns) corresponds to a source located in `Office` and oriented to the right, which is observed during optimization. The second (right three columns) corresponds to an unseen source located in `Anechoic` and oriented toward the lower left. We also visualize the direct-arrival and detected energy distributions separately; their sum yields the full energy distribution.

3.5.3 Material Parameterization Comparison

The evaluation results indicate that the unconstrained DART generally outperforms the parametric DART. To further investigate the effects of different material parameterizations, we visualize the spatial energy distributions predicted by these two variants. We choose the `Office` $\rightarrow$ `Anechoic` scene from the CR dataset (random split), and the full energy is computed as a sum of the predicted echogram $\hat{E}[n]$ over time n . We also visualize the sum of direct arrival $\hat{E}_D[n]$ and detected energy components $\hat{E}_L[n]$ separately. Two source configurations are evaluated: one seen source in `Office` and one unseen source in `Anechoic`. Receivers are positioned on a regular grid with 0.1m spacing at 1.5m height.

Figure 3.8 shows the results. Regardless of the parameterization, both DART variants respect the surrounding room geometry and predict valid energy distribution fields. However, we observe several differences. First, the source directivity of the parametric DART deviates slightly from typical radiation patterns. Second, in the unseen configuration, parametric DART exhibits irregular patterns in the scattered field produced by the radiances. We suspect that this is caused by the model trying to match the small number of measurements in the local receiver region (the small transition area between `Office` and `Anechoic`) with limited degrees of freedom. In contrast, the unconstrained DART produces smoother source radiation and energy fields. It also has fewer visible discontinuities caused by the patch and direction discretizations.

We also visualize the learned scattering patterns of representative surface patches of the `MeetingRoom` $\rightarrow$ `Hallway` scene. Because each lossless material matrix encodes scattering responses for all incident directions, we sample a single row, corresponding to a single incident direction, from the matrix for each patch. Figure 3.9 shows the results. The unconstrained DART converges to a wide range of scattering patterns, such as broadened or anisotropic lobes, whereas the parametric DART is limited to combinations of predefined BSDFs. It produces noticeably different, possibly less realistic, reflection-to-transmission ratios in some patches, likely a consequence of fitting the observations under its constrained parameterization.

3.6 Further Analysis

We conduct additional experiments to further analyze DART. All eight scenes from the HAA and CR datasets (random split) were used for evaluation, and we report the average results. We retain all baselines except the RIR-predicting versions of the neural models, as they did not improve over their echogram counterparts in the main benchmarks.

3.6.1 Effect of Measurement Quantity

We assess the effect of measurement quantity by optimizing the models with extremely sparse to moderate numbers of measurements, ranging from 4 to 48.

The top row of Figure 3.10 shows the results. Most models exhibit an expected trend: more data yields improvement across all metrics. The unconstrained DART shows consistently strong performance across the entire range. The parametric DART, by contrast, shows mixed results: while competitive in some metrics under data-scarce conditions, it underperforms in others, e.g., reverberation time and clarity, with this gap widening as more measurements become available.

3.6.2 Sensitivity to Geometry Distortion

DART depends on the room geometry, which in practice is known only approximately. The benchmarks above already use coarse meshes that omit fine architectural detail, yet DART, especially the unconstrained variant, achieved strong results. This result is consistent with the prior work that showed geometry can be substantially simplified while preserving key acoustic parameters [187]. To further stress-test this robustness, following prior works [28, 145], we optimized DART and other baselines on synthetically distorted meshes, created by perturbing each vertex by uniform noise $u \sim \mathcal{U}[-s, s]$, varying the scale s from 0.125m to 1m. The echogram measurements are kept unchanged.

The bottom row of Figure 3.10 shows the results. The unconstrained DART retains its performance well up to 0.25m distortion, likely because the low radiance sampling rate (1kHz $\approx$ 0.34m spatial resolution) makes sub-resolution perturbations negligible and the flexible material matrix can adapt to modest geometric

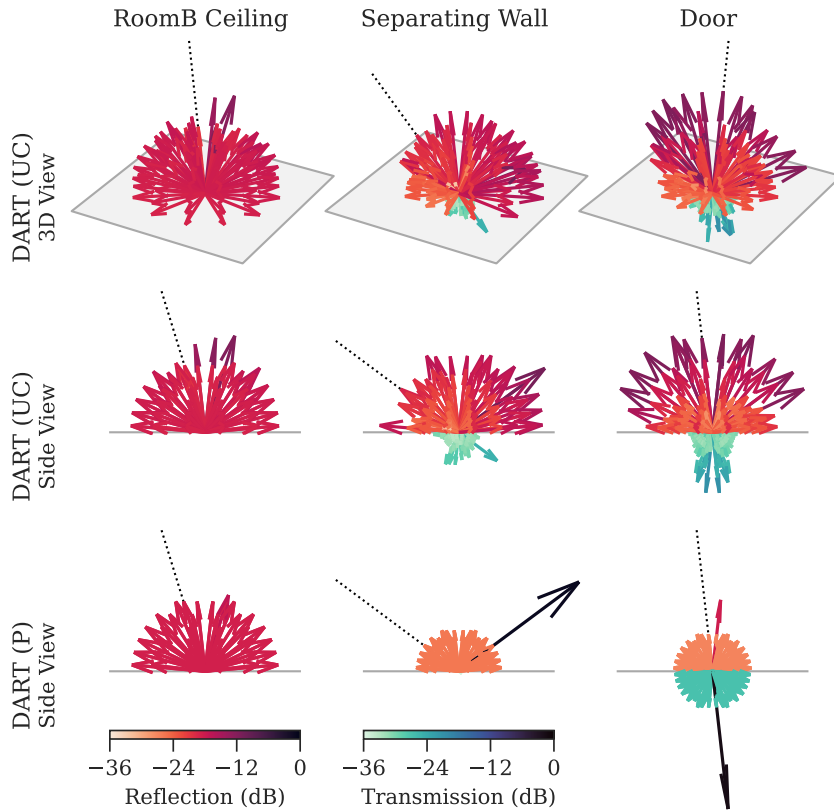


Figure 3.9: Example learned scattering patterns of DART. Three representative surface patches from the `MeetingRoom→Hallway` scene are shown, each corresponding to one column. Each black dotted line indicates the incident direction. The colors and lengths of the arrows in the reflected and transmitted patterns are shown on a decibel scale. Surface patches are visualized as rectangles for clarity, although they are triangular and vary in shape and size in the actual geometry. The materials of the unconstrained DART are shown in a three-dimensional view (top row) and a side view (middle row), while the materials of the parametric DART are shown only in a side view (bottom row) for brevity.

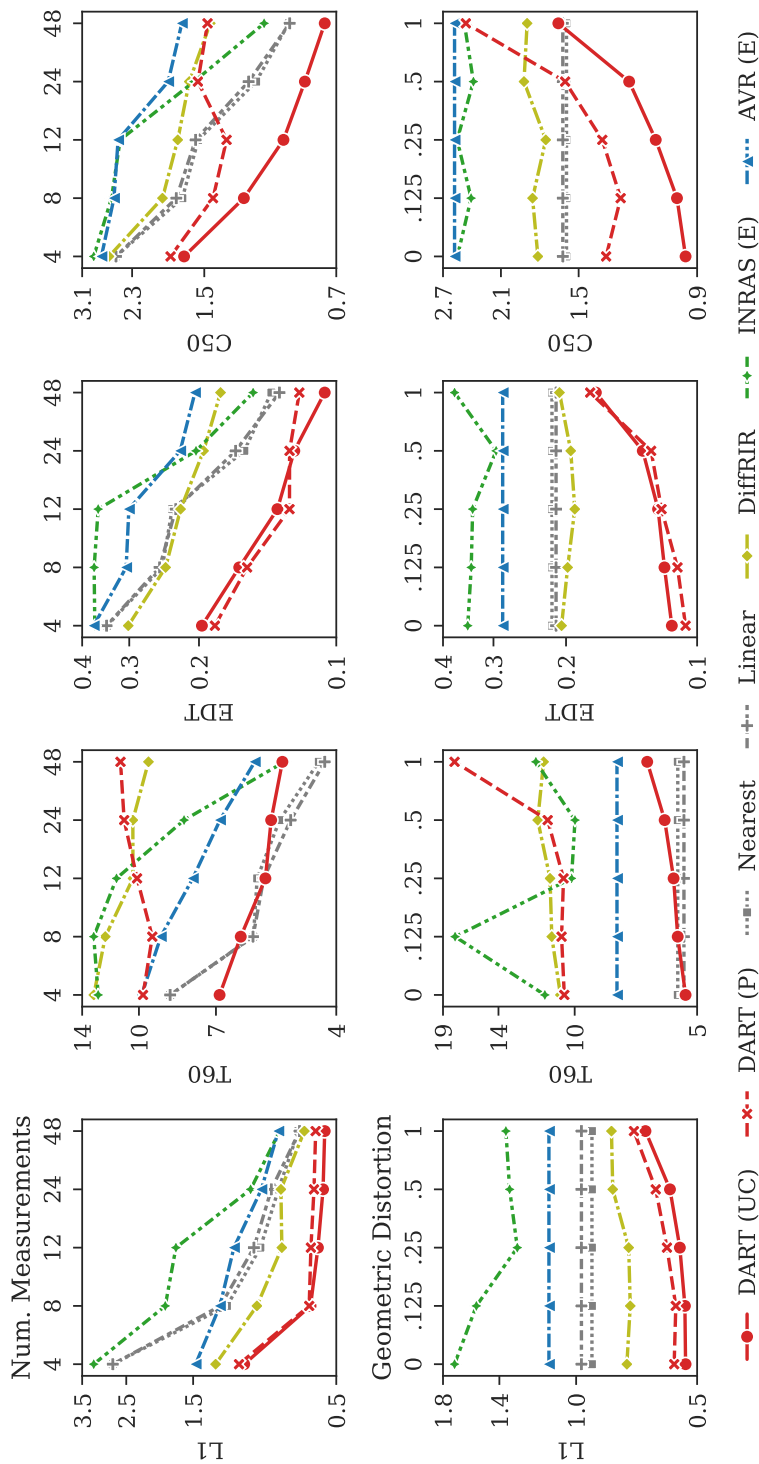


Figure 3.10: Test results with different numbers of measurements (top row) and geometric distortion (bottom row).

changes. At the largest distortion ($s = 1\text{m}$), however, the gap to even the simplest baselines shrinks, as such perturbations effectively erase critical geometric features (e.g., room dimensions, aperture sizes). The parametric DART is more sensitive to severe distortion due to its limited flexibility. DiffRIR exhibits a similar trend but with less degradation, potentially due to its learnable residual signal. INRAS (E), despite taking geometry as input, does not show a clear trend, suggesting that its ART-inspired architecture might not utilize the geometry in the same way as ART itself. The remaining baselines do not rely on geometry and are therefore unaffected.

3.6.3 Ablation Study

We evaluate the various design choices and hyperparameters of DART; Table 3.3 reports the results, along with the parameter count, memory usage (in GB), and iteration time (in ms) for each variant during training and inference (measured with a single RTX4090 GPU). In addition to the base 12-measurement setup, we include a 48-measurement setup to investigate the effects of data availability on each configuration.

First, we verify that allowing each subpatch to have its own independent material parameters is advantageous, as the shared variant gave degraded results in most metrics, especially in T60 and C50. This gap was particularly pronounced for the parametric DART, which has orders of magnitude fewer parameters than the unconstrained one, so further restriction was especially detrimental.

Next, we varied the discretization resolution for the unconstrained variant. For the patch resolution, we roughly halved or doubled the number of patches by adjusting the mesh subdivision parameters. For the directional resolution, we explored 8×8 and 16×16 grids, compared to the base 12×12 . With 12 measurements, resolution has no clear effect on performance. However, with 48 measurements, finer patch and directional resolutions consistently improve the results, suggesting that the optimal discretization depends on data availability. That said, increasing the resolution also comes with increased computational cost, and the gains from finer discretization may not always justify this cost; the most lightweight variant, with both coarser patch and directional resolutions,

still achieves competitive results with significantly fewer parameters and faster runtime. The base setting offers a good balance between computational cost and performance.

We also examined special cases of the parametric DART: optimizing the reflection coefficients under fixed scattering properties (only specular or only diffuse reflection), as in prior works [149, 156, 159, 158, 157]. Both restrictions degraded the results, confirming the importance of flexible material properties.

Finally, we validated the two nontrivial approximations that we introduced for DART: the visibility-material decomposition (3.25) and the use of the material matrix for injection (3.40). For the parametric variant, whose predefined component BSDFs are known, each can be disabled: the decomposition by precomputing the full matrix (3.24) per component, and the material matrix injection by reverting to explicit single-bounce ray tracing. For the unconstrained variant, disabling either would require explicit BSDF parameterization and numerical integration, significantly increasing computational cost and system complexity. We therefore test on the parametric variant only.

The last two rows of Table 3.3 report the results. Removing the decomposition shows no clear performance difference but incurs a significant slowdown, as the full matrix is much denser than its decomposed factors. Using explicit injection slightly degrades most metrics, likely due to increased stochasticity from ray tracing. Overall, these results confirm that both approximations are reasonable, while the decomposition additionally provides a clear computational advantage.

3.6.4 Computational Cost

Table 3.2 lists the asymptotic work complexity and measured execution time of each operation in the unconstrained DART. Operations marked with the approximation symbol $\approx$ involve ray tracing, whose exact complexity depends on the scene geometry and the implementation; the $\log N_{\text{pat}}$ factor reflects acceleration-structure traversal for ray-triangle intersection.

The precomputation of fixed components such as the mean visibility matrix and propagation delays is performed once per scene before optimization. The visibility-material decomposition (3.25) reduces the four nested integrals (or three, if we

Table 3.2: **Asymptotic work complexity and execution speed (in ms) of each operation in DART.**

Operation	Work Complexity	Speed
Precomputation	$\approx \mathcal{O}(N_{\text{precompute}} N_{\text{pat}} \log N_{\text{pat}})$	406.8
Injection	$\approx \mathcal{O}(N_{\text{ray}} \log N_{\text{pat}} + \bar{N}_{\text{mat}} T)$	5.2
Delay	$\mathcal{O}(N_{\text{order}} \bar{N}_{\text{rad}} T)$	2.6
Visibility	$\mathcal{O}(N_{\text{order}} \bar{N}_{\text{vis}} T)$	12.4
Material	$\mathcal{O}(N_{\text{order}} \bar{N}_{\text{mat}} T)$	33.4
Accumulation	$\mathcal{O}(N_{\text{order}} \bar{N}_{\text{rad}} T)$	2.6
FFT & IFFT	$\mathcal{O}(\bar{N}_{\text{rad}} T \log T)$	0.2
Detection	$\approx \mathcal{O}(N_{\text{ray}} \log N_{\text{pat}} + \bar{N}_{\text{rad}} T \log T)$	5.8
Total (sync)	—	62.2
Total (wall clock)	—	56.6

collapse the Dirac delta) of the full scalar-valued matrix (3.24) to at most two surface integrals for the mean visibility matrix (3.26). Precomputing these decomposed factors is therefore substantially cheaper than precomputing the full matrix, but would still dominate the overall training time unless performed beforehand.

The main iterative radiance calculation is the most costly part. Each of the N_{order} reflection steps is sequential, as it depends on the previous step’s output. Within each step, the batched sparse matrix multiplications are performed in parallel. The sparsity ratios, averaged across all tested scenes, are

$$\bar{N}_{\text{rad}} \approx 5.8 \times 10^{-1} N_{\text{rad}}, \quad (3.50a)$$

$$\bar{N}_{\text{vis}} \approx 8.9 \times 10^{-4} \bar{N}_{\text{rad}}^2 = 3.0 \times 10^{-4} N_{\text{rad}}^2, \quad (3.50b)$$

$$\bar{N}_{\text{mat}} \approx 5.6 \times 10^{-3} \bar{N}_{\text{rad}}^2 = 1.9 \times 10^{-3} N_{\text{rad}}^2. \quad (3.50c)$$

Applying the material matrix is the most costly step, as this matrix is roughly six times denser than the mean visibility matrix. The delay and accumulation steps are relatively lightweight, as they involve only elementwise multiplications and additions on the frequency-domain radiances, respectively.

Injection and detection each involve one set of ray-triangle intersection tests, parallelized over N_{ray} rays. The detection and direct-arrival tests are fused into a single pass. Finally, the $T \log T$ terms correspond to the FFT and IFFT required for the frequency-domain radiance computation and the frequency-domain convolu-

tion for detection, respectively.

The per-operation times reported in Table 3.2 are measured with GPU synchronization barriers. The actual wall-clock time is 56.6ms, slightly shorter than the sum of the per-operation times, as the GPU pipelines consecutive operations.

Inference time is scene-dependent, mainly driven by the truncation order and number of radiances: 87ms for `Complex`, 34ms for `Dampened`. Overall, the inference time of DART is higher than that of the other baselines, approximately 5ms except for AVR at 11.5ms. Still, this gap can be narrowed by reducing the discretization resolution, which drops inference time to 13.4ms with only a modest performance tradeoff (see Table 3.3). We can further apply ART acceleration techniques after optimization, e.g., precomputing and compressing the transport matrix [188, 165]. When evaluating multiple receivers for the same source, DART can reuse the computed radiances and perform only separate detections, reducing per-query cost.

3.7 Conclusion

This chapter introduced DART. By decomposing the reflection kernel into mean visibility and material matrices, computing radiance via a frequency-domain truncated Neumann series, and exploiting sparsity, we achieved tractable end-to-end optimization on a single GPU. Experiments on real measurements showed that DART performs favorably against existing baselines, especially in various challenging yet practical scenarios.

We presented two variants: unconstrained DART, which learns free-form material matrices, and parametric DART, which constrains them to convex combinations of predefined BSDFs. The unconstrained variant generally achieved stronger results than the parametric variant owing to its greater flexibility and ability to absorb geometric distortions, yet this gap may narrow with richer component BSDFs [189].

DART can be extended in multiple ways. Examples include adding diffraction modeling to ART [190]. We can also combine other geometric acoustics models, e.g., the image source method, to mitigate blurred specular components caused

Table 3.3: **Ablation study of DART.** Parameter count, memory usage (in GB), and wall-clock time (in ms) are shown. Costs are reported for both training and inference, with the latter in parentheses.

Variants	Model Size & Runtime			12 Measurements				48 Measurements			
	Parameters	Memory	Speed	L1	T60	EDT	C50	L1	T60	EDT	C50
DART (UC)	4204555	3.32 (0.74)	143.4 (56.6)	0.576	6.04	0.097	0.92	0.547	5.58	0.071	0.72
Shared parameters	4204369	3.35 (0.74)	143.3 (56.1)	0.572	9.84	0.091	1.23	0.550	8.61	0.073	0.96
Coarser patch	2545595	2.34 (0.48)	91.3 (35.9)	0.576	6.07	0.101	0.94	0.551	5.92	0.066	0.77
Coarser direction	830795	1.66 (0.29)	44.1 (17.8)	0.579	6.44	0.094	0.92	0.558	5.91	0.070	0.80
Coarser pat. & dir.	503035	1.26 (0.19)	34.2 (13.4)	0.579	6.55	0.097	0.95	0.561	6.05	0.069	0.84
Finer patch	7885373	5.74 (1.38)	293.3 (121.6)	0.571	7.22	0.101	0.98	0.541	5.53	0.071	0.70
Finer direction	13287755	5.72 (1.56)	409.3 (160.6)	0.577	6.04	0.098	0.91	0.543	5.49	0.070	0.71
DART (P)	971	3.17 (0.67)	141.1 (55.8)	0.608	10.89	0.090	1.30	0.586	11.75	0.084	1.46
Shared parameters	198	3.17 (0.67)	141.3 (55.7)	0.624	15.23	0.110	2.44	0.583	13.35	0.100	1.47
Specular refl. only	534	3.14 (0.63)	141.1 (55.8)	0.678	10.81	0.113	1.66	0.636	9.85	0.106	1.41
Diffusive refl. only	534	3.14 (0.63)	140.7 (55.9)	0.621	13.24	0.107	1.62	0.601	12.47	0.096	1.51
w/o mat. decomp.	971	2.96 (1.13)	724.8 (209.0)	0.604	11.69	0.091	1.24	0.589	12.01	0.086	1.44
Explicit injection	971	3.17 (0.67)	140.9 (56.9)	0.612	11.50	0.091	1.37	0.591	11.42	0.091	1.43

by the spatial discretization. Incorporating learnable delay lines [175] could help compensate for inaccurate propagation delays caused by geometric distortion, beyond what the unconstrained material matrices can absorb. Furthermore, while we introduced various techniques to improve efficiency, they do not fundamentally change the quadratic complexity in the number of radiances, which currently limits DART to geometries with up to around one thousand patches. Reducing this cost would allow handling finer or more complex geometries.

This chapter focused on echogram prediction, and obtaining RIRs with DART requires an additional auralization step. Appendix 3.D reports a preliminary investigation with a simple auralization pipeline, which shows promising results using the current full-band echogram predictions. Extending to frequency-dependent modeling is a natural next step that could further improve the auralization quality.

Beyond measurement-conditioned field estimation, DART could support goal-based acoustic design by optimizing its material parameters against prescribed acoustic measures [191].

Although DART is a pure signal processing model, it can be combined with neural networks. For instance, a neural network could predict material parameters from geometry and sparse measurements, enabling cross-scene generalization without per-scene optimization [140, 149]. Alternatively, the radiances could be parameterized by a neural network trained to satisfy the ARE [192, 193], or a neural network could model the residual of DART’s predictions [160, 161]. Ultimately, the challenge is to develop a model that leverages physical priors when data is scarce, yet scales flexibly to rich, data-driven scenarios.

Appendix 3.A Delay Network Interpretation of ART

ART is described with the discrete ARE (3.16), along with discretizations of the continuous injection (3.7) and detection (3.8). It can be viewed as a specific instance

of a delay network. In a general form, a delay network can be written as

$$\mathbf{p}[n] = \mathbf{A}[n] \circledast \mathbf{p}[n] + \mathbf{b}[n] \circledast u[n], \quad (3.51a)$$

$$y[n] = \mathbf{c}^\top[n] \circledast \mathbf{p}[n] + d[n] \circledast u[n], \quad (3.51b)$$

where $u[n]$ and $y[n]$ are scalar input and output signals, and $\mathbf{p}[n]$ is an $N \times 1$ vector of internal states. $\mathbf{A}[n]$, $\mathbf{b}[n]$, $\mathbf{c}[n]$, and $d[n]$ correspond to an $N \times N$ feedback mixing filter matrix, $N \times 1$ input filters, $N \times 1$ output filters, and a 1×1 bypass filter. The term “delay network” refers to the fact that parallel delay lines are embedded in the feedback recursion $\mathbf{A}[n]$.

ART and DART can be written in this form by identifying the filter components as

$$\mathbf{A}[n] = \hat{\mathbf{R}}[n] \approx \hat{\mathbf{M}}\hat{\mathbf{V}} \text{diag} \hat{\mathbf{D}}[n], \quad (3.52a)$$

$$\mathbf{b}[n] = \hat{\mathbf{L}}^{(0)}[n] \approx \hat{\mathbf{M}}(\text{diag} \hat{\mathbf{G}})^{-1} \hat{\mathbf{P}}^{(\text{in})}[n], \quad (3.52b)$$

$$\mathbf{c}[n] = \hat{\mathbf{W}}[n], \quad (3.52c)$$

$$d[n] = \hat{E}_D[n], \quad (3.52d)$$

where the two approximations are those introduced for DART, e.g., the kernel decomposition (3.21 and 3.25) and injection approximation (3.40). When the input is a unit impulse emission, $u[n] = \delta[n]$, (3.51a) is exactly the same as the discrete ARE (3.16), and the internal state corresponds to the discrete radiance: $\mathbf{p}[n] = \hat{\mathbf{L}}[n]$. Applying the output (detection) filter then yields the detected power, $\mathbf{c}^\top[n] \circledast \mathbf{p}[n] = \hat{E}_L[n]$, same as the discretized detection (3.43). Finally, (3.51b) computes the echogram $y[n] = \hat{E}[n]$. The number of parallel delay lines is given by $N = N_{\text{rad}}$ or $N = \bar{N}_{\text{rad}}$ when exploiting the radiance sparsity, as discussed in Section 3.3.3.

Appendix 3.B Derivations

3.B.1 Visibility-Material Decomposition (3.25)

First, we rewrite the scalar-valued matrix (3.24) by reparameterizing the source patch integral into a solid angle integral.

$$\hat{S}_{hj,ik} = \iint_{\mathcal{S}_{ik}} \iint_{\mathcal{A}_i} \iint_{\mathcal{S}_{hj}} \iint_{\mathcal{S}} V_h(x', \Phi) \rho_i(\Phi, \Omega) |\Phi \cdot \nu_i| \delta(\Phi + \Theta) d\Phi d\Theta \frac{d\mathcal{A}(x')}{|\mathcal{A}_i|} \frac{d\Omega}{|\mathcal{S}_{ik}|}. \quad (3.53)$$

Note that we also replaced the visibility term $V(x, x')$ with a patch-dependent one $V_h(x', \Phi)$, which denotes “a ray emitted from x' in direction Φ hits patch $\mathcal{A}_h$.” Next, by collapsing the integration over outgoing directions $\Theta \in \mathcal{S}_{hj}$ using the Dirac delta and decomposing the solid integral with the entire sphere into multiple incident directions, i.e., $\mathcal{S} = \cup_l \mathcal{S}_{il}$, we can write

$$\hat{S}_{hj,ik} = \sum_{l=1}^{N_{\text{dir}}} \iint_{\mathcal{S}_{ik}} \iint_{\mathcal{S}_{il}} \left[\iint_{\mathcal{A}_i} V_{hj}(x', \Phi) \frac{d\mathcal{A}(x')}{|\mathcal{A}_i|} \right] \rho_i(\Phi, \Omega) |\Phi \cdot \nu_i| d\Phi \frac{d\Omega}{|\mathcal{S}_{ik}|}. \quad (3.54)$$

Now, assume that the average visibility (the square-bracketed term) for the incident direction Φ is approximately constant within each discretized bin $\mathcal{S}_{il}$:

$$\iint_{\mathcal{A}_i} V_{hj}(x', \Phi) \frac{d\mathcal{A}(x')}{|\mathcal{A}_i|} \approx \hat{V}_{hj,il}, \quad \Phi \in \mathcal{S}_{il}, \quad (3.55)$$

Then, we can replace the visibility integral in (3.54) with the mean visibility matrix $\hat{V}_{hj,il}$ and take it out from the integrals. Under this approximation, the remaining integral reduces to the definition of the material matrix $\hat{M}_{il,ik}$, thereby yielding the visibility-material decomposition equation (3.25).

3.B.2 Material Matrix-Based Injection (3.40)

We can rewrite the discrete initial radiance as

$$\hat{L}_{ik}^{(0)}[n] = \sum_{l=1}^{N_{\text{dir}}} \frac{1}{|\mathcal{A}_i|} \iint_{\mathcal{S}_{ik}} \iint_S \Gamma_s(\Theta, o_s) D(x_s, x, n\Delta t) V_{il}(x_s, \Theta) \rho_i(-\Theta, \Omega) \frac{d\Theta}{4\pi} \frac{d\Omega}{|\mathcal{S}_{ik}|} \quad (3.56)$$

where we used the solid angle reparameterization and split the input directions into $\mathcal{S}_{il}$ by changing the visibility term to V_{il} .

Now, we introduce three approximations. First, analogous to the simplification step in the reflection kernel (3.21), we separate the delay term from the integral. Second, we assume that the BSDF ρ_i is approximately constant for incident direction bin $\mathcal{S}_{il}$ and outgoing direction bin $\mathcal{S}_{ik}$. Then, we can separate the material term from the integral.

$$\hat{L}_{ik}^{(0)}[n] \approx \sum_{l=1}^{N_{\text{dir}}} \hat{D}_{s,il}[n] \left[\iint_S \Gamma_s(\Theta, o_s) V_{il}(x_s, \Theta) \frac{d\Theta}{4\pi} \right] \tilde{M}_{il,ik}, \quad (3.57)$$

where

$$\tilde{M}_{il,ik} = \frac{1}{|\mathcal{A}_i|} \iint_{\mathcal{S}_{ik}} \iint_{\mathcal{S}_{il}} \rho_i(\Phi, \Omega) \frac{d\Phi}{|\mathcal{S}_{il}|} \frac{d\Omega}{|\mathcal{S}_{ik}|}. \quad (3.58)$$

Finally, we can insert the factor $|\Phi \cdot \nu_i|/|\Phi \cdot \nu_i| = 1$ into the integrand of the material equation (3.58) and split its denominator out of the integral using the following approximation:

$$|\Phi \cdot \nu_i| \approx \iint_{\mathcal{S}_{il}} |\bar{\Phi} \cdot \nu_i| \frac{d\bar{\Phi}}{|\mathcal{S}_{il}|}, \quad \Phi \in \mathcal{S}_{il}. \quad (3.59)$$

We then obtain

$$\tilde{M}_{il,ik} \approx \hat{M}_{il,ik} / \hat{G}_{il}. \quad (3.60)$$

Combining it with (3.57) gives the discretized injection equation with the material matrix (3.40).

Appendix 3.C Aliasing Suppression

Frequency-sampling the z -domain ARE (3.18) at $z = z_{T,f}$, i.e., T uniformly spaced points on the unit circle, and applying the IFFT yields length- T signals that suffer from time-aliasing [177]. For the exact radiance $\hat{\mathbf{L}}[n]$, its frequency-sampled, hence aliased, signal is given as

$$\hat{\mathbf{L}}_T[n] = \sum_{m=0}^{\infty} \hat{\mathbf{L}}[n + mT], \quad 0 \leq n < T. \quad (3.61)$$

The total aliasing error is

$$\mathcal{E}_T = \|\hat{\mathbf{L}}_T[n] - \hat{\mathbf{L}}[n]\| = \left\| \sum_{m=1}^{\infty} \hat{\mathbf{L}}[n + mT] \right\|, \quad (3.62)$$

where $\|\cdot\|$ denotes any p -norm ($p \geq 1$) over the first T samples. Because the radiance signals are nonnegative, aliasing always causes overestimation, with pronounced errors at the beginning of the signal where the true energy is zero.

To suppress the aliasing, we frequency-sample on a circle of radius $\gamma^{-1/T} > 1$. Since $z_{T,\gamma,f} = \gamma^{-1/T} z_{T,f}$, frequency-sampling at $z_{T,\gamma,f}$ is equivalent to sampling the original signal damped with an exponentially decreasing window $\gamma^{n/T}$ in the time domain. Therefore, applying the IFFT and compensating the damping with an exponentially increasing window $\gamma^{-n/T}$ yields the aliasing-suppressed signal

$$\hat{\mathbf{L}}_{T,\gamma}[n] = \gamma^{-n/T} \sum_{m=0}^{\infty} \gamma^{n/T+m} \hat{\mathbf{L}}[n + mT] \quad (3.63a)$$

$$= \sum_{m=0}^{\infty} \gamma^m \hat{\mathbf{L}}[n + mT]. \quad (3.63b)$$

The time-aliasing error therefore satisfies

$$\mathcal{E}_{T,\gamma} = \left\| \sum_{m=1}^{\infty} \gamma^m \hat{\mathbf{L}}[n + mT] \right\| < \gamma \mathcal{E}_T, \quad (3.64)$$

where the inequality uses nonnegativity of the radiance and each aliased tail ($m \geq 1$) being scaled by $\gamma^m < 1$.

We assumed the exact radiance for the above derivation, but a similar bound

holds for the truncated Neumann series. Furthermore, since high-order terms predominantly contribute to the tail $n \geq T$, the error of the truncated series $\mathcal{E}_{T, N_{\text{order}}}$ or its antialiased version $\mathcal{E}_{T, N_{\text{order}}, \gamma}$ is no larger than the original counterpart ($\mathcal{E}_T$ and $\mathcal{E}_{T, \gamma}$, respectively). In other words, increasing the truncation order beyond what is needed to capture the dominant energy within the echogram duration T yields no benefit, as the additional terms only add to the aliased tail.

Appendix 3.D RIR Estimation with DART Echograms

DART predicts echograms, i.e., time-energy responses, rather than waveform RIRs directly. For applications that require an RIR, the echogram must therefore be paired with an auralization stage. This appendix evaluates a simple retrieval-based auralization pipeline to verify that this does not prevent DART from supporting acoustic field rendering.

Given a target source–receiver pair, the pipeline retrieves a measured RIR $h^*[n]$ with the nearest source and receiver positions x_s^* and x_r^* . It then aligns the direct-path delay and shapes the retrieved RIR by the ratio between DART’s predicted echogram and the retrieved echogram:

$$\begin{aligned}\hat{h}[n] &= \sqrt{G[n]} h^*[n - \Delta d], \\ G[n] &= \text{upsample} \left[\frac{\hat{E}[n]}{\hat{E}^*[n]} \right], \\ \Delta d &= \frac{\|x_s - x_r\| - \|x_s^* - x_r^*\|}{c\Delta t}.\end{aligned}\tag{3.65}$$

Table 3.4 reports the resulting RIR-prediction metrics on the HAA and CR benchmarks. Even with this simple auralization stage, DART remains competitive with RIR-domain baselines, supporting the use of DART’s per-position echogram predictions as the acoustic component of an RIR-rendering pipeline.

Appendix 3.E Geometric Resolution

Since DART discretizes both room geometry $\mathcal{A}$ and radiance directions $\mathcal{S}$, this appendix further evaluates how these resolutions affect optimization when accurate geometry is available. This experiment is distinct from the geometry-distortion study in Section 3.6.2: instead of perturbing coarse HAA/CR meshes, it uses the `Apartments_idx_24` scene from `AcousticRooms`, where the reference geometry is known. Meshes with $N_{\text{pat}} \in \{125, 250, 500, 1000\}$ patches are produced by quadric edge-collapse decimation, and three direction grids are tested: $(8, 8)$, $(12, 12)$, and $(16, 16)$. The split is 2000/250/250 for training, validation, and testing, respectively.

Table 3.5 reports the unconstrained DART results. Performance generally improves as the geometry and directional resolutions become finer, but the gain is moderate: the best improvements are about 1.25% in reverberation time and 0.3dB in clarity. This suggests that the unconstrained material matrix can absorb some missing geometric detail even when the mesh is rough, while finer discretization still helps when enough measurements are available.

Table 3.4: Impulse response prediction results.

Methods	HAA						CR (random)					
	MAG	PHA	ENV	T60	EDT	C50	MAG	PHA	ENV	T60	EDT	C50
Nearest neighbor	0.374	1.611	3.957	7.55	0.222	1.60	0.349	1.584	2.494	4.91	0.179	1.50
Linear interpolation	0.389	1.612	3.946	7.24	0.209	1.55	0.320	1.582	2.433	4.79	0.178	1.48
DiffRIR [28]	0.327	1.608	3.712	8.32	0.171	1.76	0.327	1.579	2.563	14.62	0.196	1.63
INRAS [80]	0.420	1.621	4.608	9.97	0.248	1.85	0.636	1.618	3.707	17.93	0.351	3.15
AVR [142]	0.600	1.623	3.984	9.76	0.313	2.27	0.516	1.593	2.415	10.01	0.284	2.37
DART (UC) + retrieval	0.386	1.560	3.861	7.04	0.114	1.13	0.279	1.379	2.152	5.10	0.080	0.72
DART (P) + retrieval	0.369	1.564	3.853	16.60	0.100	1.84	0.304	1.384	2.196	5.85	0.082	0.76

Table 3.5: Effects of varying spatial and directional resolutions on DART (UC).

L1	8, 8				12, 12				16, 16				T60	8, 8				12, 12				16, 16				EDT	8, 8				12, 12				16, 16				C50														
	125	250	500	1000	0.603	0.566	0.557	0.555	0.602	0.561	0.555	0.553		0.600	0.561	0.555	0.555	4.60	5.70	4.85	4.60	4.78	4.55	4.54	4.86		125	250	500	1000	71.73	49.72	49.38	50.31	70.60	48.83	50.15	50.60		69.98	125	250	500	1000	0.890	0.621	0.593	0.605	0.889	0.613	0.593	0.604	0.876
125	0.603	0.566	0.557	0.555	0.602	0.561	0.555	0.553	0.600	0.561	0.555	0.555	0.600	0.561	0.555	0.555	4.60	5.70	4.85	4.60	4.78	4.55	4.54	4.86	125	250	500	1000	71.73	49.72	49.38	50.31	70.60	48.83	50.15	50.60	69.98	125	250	500	1000	0.890	0.621	0.593	0.605	0.889	0.613	0.593	0.604	0.876	0.601	0.590	0.607
250	0.566	0.557	0.555	0.553	0.561	0.555	0.555	0.553	0.561	0.555	0.555	0.555	0.561	0.555	0.555	0.555	5.31	4.84	4.85	4.78	4.55	4.54	4.86	4.63	250	500	1000	125	49.72	49.38	50.31	50.60	48.83	50.15	49.92	48.07	250	500	1000	125	0.621	0.593	0.605	0.604	0.613	0.593	0.604	0.607					
500	0.557	0.555	0.555	0.553	0.555	0.555	0.555	0.553	0.555	0.555	0.555	0.555	0.555	0.555	0.555	0.555	4.84	4.85	4.85	4.58	4.45	4.45	4.86	4.63	500	1000	125	500	49.38	50.31	50.60	49.92	48.07	48.07	48.07	48.07	500	1000	125	500	0.593	0.605	0.604	0.604	0.593	0.593	0.590	0.607					
1000	0.555	0.555	0.555	0.555	0.555	0.555	0.555	0.555	0.555	0.555	0.555	0.555	0.555	0.555	0.555	0.555	1000	4.85	4.85	4.85	4.45	4.45	4.45	4.86	1000	125	500	1000	50.31	50.60	49.92	49.97	49.97	49.97	49.97	49.97	1000	125	500	1000	0.605	0.605	0.604	0.604	0.604	0.604	0.604	0.607					

Chapter 4

Autoregressive Modeling of Neural Discrete Representations for Room Impulse Response Estimation

4.1 Introduction

We consider the problem of estimating a room impulse response (RIR) from indirect reference signals, such as reverberant speech or a source/target audio pair, rather than from a direct impulse response measurement. Note that RIR estimation can be seen as an extension of room acoustics parameter estimation [104, 105, 106, 107] as the RIR estimation can derive all these parameters afterwards [103, 123]. Several neural networks for RIR estimation have been proposed [82, 83, 84]. These models typically consist of two components: (i) an encoder that extracts RIR-relevant features from a reference and (ii) a generator that synthesizes the RIR from the extracted features. Their main architectural differences lie in the generator. For example, Filtered Noise Shaping (FiNS) [82] outputs attenuation envelopes for filtered noise together with an additional early reflection signal. This design choice stems from the observation that late reverberation exhibits perceptual similarities to time-varying filtered noise. While FiNS offers a balanced approach, further inductive bias can be introduced by using an artificial reverberation algorithm as a parameterized RIR generator [83], as in the differentiable artificial reverberation studied in Chapter 2. This approach offers benefits such as training data efficiency and computational efficiency during auralization.

However, as more RIR data [34, 194, 35, 182] and computation budget become available, more expressive neural architectures may be desirable despite higher data and computation demands. One direction is to remove the filtered-noise component from FiNS and develop a more general generator, as in the speech-to-IR generative adversarial network (S2IR-GAN) [84], which uses stacked convolutional layers for RIR synthesis. More broadly, neural networks developed for audio synthesis could be adapted. However, RIRs (and IRs in general) exhibit unique signal characteristics, such as sparsity and sharp transients; hence, architectures suitable for RIR generation may differ from generic audio generators and remain to be explored.

This chapter proposes a new RIR generator based on discrete representation learning and autoregressive token generation (Figure 4.1). A variational autoencoder with residual quantization (RQ-VAE) [85] learns a discrete time-frequency representation of measured RIRs, converting each RIR into an index tensor. Transformer models [195] then decode this tensor autoregressively. These choices are tailored to the structure of RIRs: the RQ-VAE represents them as local time-frequency patches, while the axial transformers exploit the resulting frequency-time-depth tensor rather than treating an RIR as a generic one-dimensional audio sequence. Each RIR index tensor is three-dimensional with frequency, time, and quantization depth axes; following [86, 89, 87], we explore various strategies to decode this tensor in a hierarchical and factorized manner. By encoding the references and providing the features as *prompt* tokens to the transformers, the same generator becomes a conditional model for RIR estimation. Along with the standard blind estimation (reverberant-speech-to-RIR), we also address the acoustic matching scenario, whose goal is to find an RIR that matches the source signal to the target signal’s reverberation characteristics. We evaluate the generator in analysis-synthesis, blind estimation, and acoustic matching settings, comparing it with filtered-noise, differentiable artificial reverberation, non-autoregressive, and alternative autoregressive baselines through both objective reverberation metrics and MUSHRA-like listening tests. The ablations isolate the contribution of the discrete representation and autoregressive decoding, showing that both components are important under the proposed framework.

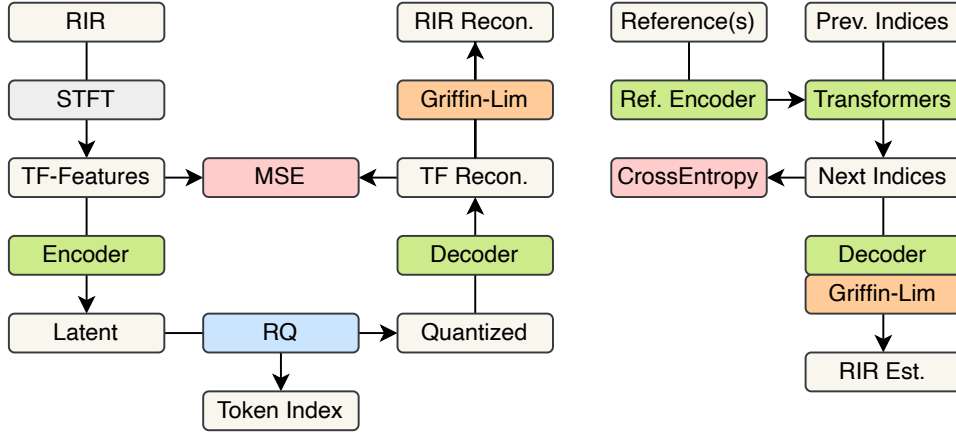


Figure 4.1: **The proposed method.** Left: codebook learning with the RQ-VAE. Right: RIR estimation via conditional token generation.

The final part of the chapter then develops a hybrid extension that combines this learned generator with differentiable signal processing structure. This extension instantiates the differentiable-processing-generative hybrid viewpoint of the thesis: the learned discrete generator provides a data-driven prior for complex measured RIR structure, while the differentiable filtered-noise decoder provides a domain prior for the statistically regular late reverberation tail. By assigning different temporal regions of the RIR to the prior that matches them better, the hybrid design improves modeling efficiency, reducing the token budget and CPU inference cost while keeping the late reverberation component tied to an interpretable signal processing model.

4.2 Discrete Representation Learning with RQ-VAE

4.2.1 Input Representation

We first convert a single-channel RIR $h \in \mathbb{R}^{49152}$ (1.1 second, 44.1kHz sampling rate) into a time-frequency representation with a short-time Fourier transform (STFT) using an FFT size of 512 and a hop size of 64. Compared with common audio STFT settings at this sampling rate, the FFT size is short and the hop size is only 1/8 of the analysis window. This intentionally prioritizes time-axis resolution

so that the representation can capture the sparse and sharp transient structure of RIRs, especially early reflections. We apply power compression [196] and stack the magnitude, real, and imaginary spectrograms to obtain a three-channel feature $\mathbf{H} \in \mathbb{R}^{3 \times 256 \times 768}$ (256 frequency bins and 768 time frames).

4.2.2 Autoencoding

We compress and expand the feature $\mathbf{H}$ using an autoencoder [197, 198] that stacks residual blocks and downsampling layers (Figure 4.2). For the frequency axis, we replace strided convolutions with nonlinear filterbank matrices to avoid over-compressing the low-frequency region; each matrix halves the frequency dimension, and the cascade of 3 filterbanks gives a frequency mapping that approximates the Mel scale. We downsample the time axis with strided convolutions. We reduce the frequency and time dimensions by 8 and 4 times and increase the channels to 192, resulting in a latent $\mathbf{S} \in \mathbb{R}^{192 \times 32 \times 192}$, whose axes denote channel, frequency, and time. The decoder mirrors the encoder with transposed filterbanks/convolutions.

4.2.3 Residual Quantization

We discretize the latent $\mathbf{S}$ with residual quantization (RQ) [197, 199, 85], training the RQ-VAE with up to 8 residual codebooks of 256 codes each with standard training practices (EMA updates, k-means initialization, stale code re-initialization, quantization dropout). For token generation, we retain only the first 3 residual depths because deeper codebooks provide diminishing reconstruction gains, as illustrated in Figure 4.3. The retained representation converts each RIR h into a quantized feature $\mathbf{S}^q \in \mathbb{R}^{192 \times 32 \times 192}$ or its corresponding indices $\mathbf{Q} \in \mathbb{N}^{32 \times 192 \times 3}$, whose axes denote frequency, time, and quantizer depth. We denote a single index with frequency f , time t , and quantizer depth d as $\mathbf{Q}_{f,t,d}$. A colon denotes a full axis, e.g., $\mathbf{Q}_{f,:,:} \in \mathbb{N}^{192 \times 3}$ denotes the codes of the f^{th} frequency bin. $\mathbf{Q}_{:,<t,:}$ denotes codes with a time index less than t .

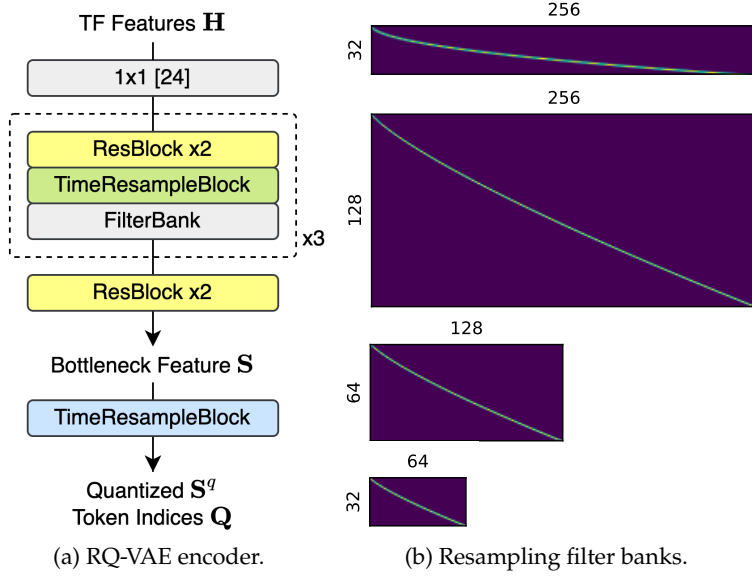


Figure 4.2: **RQ-VAE encoder and resampling filter banks.** (a) Encoder architecture, stacking residual blocks with downsampling along time and filter bank resampling along frequency. (b) Cascade of three filter banks producing a frequency mapping that approximates the Mel scale.

4.2.4 Differences to Existing Neural Audio Codecs

Unlike general audio codecs, our RQ-VAE encodes each frequency band separately, respecting the LTI property that each band responds independently. Each latent vector $\mathbf{S}_{:,f,t} \in \mathbb{R}^{192}$ (or its discrete tokens $\mathbf{Q}_{f,t,:} \in \mathbb{N}^3$) represents an overlapping time-frequency patch. Figure 4.3 illustrates the resulting coarse-to-fine reconstruction behavior across residual quantization depths.

4.3 Autoregressive Token Generation with Axial Transformers

Building on autoregressive transformer models, we explore three factorization strategies for decoding the discrete RIR tokens. The common design principle is to keep the frequency axis explicit rather than flattening the entire tensor into scalar tokens. Under the LTI approximation, convolution with an RIR corresponds

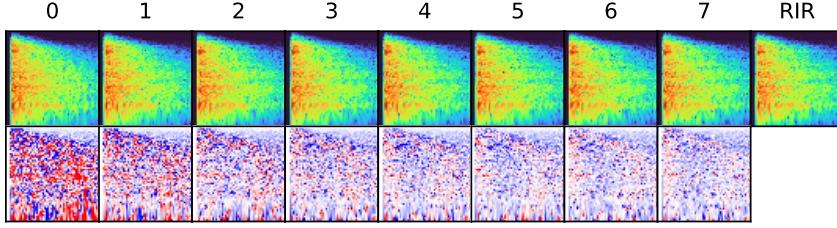


Figure 4.3: **Coarse-to-fine modeling of IRs with RQ-VAE.**

to frequency-domain multiplication, so each frequency band of the IR primarily shapes the corresponding band of the signal. We therefore generate a full frequency column at each autoregressive step, while allowing the axial transformer layers to exchange information across frequency bands internally. This reduces the autoregressive sequence length without discarding the band-aligned structure of the RIR representation.

4.3.1 Factorization Strategies

Since each RIR h is discretized, the generation objective is to model the index tensor $\mathbf{Q}$, whose axes correspond to frequency, time, and quantization depth. Applying autoregressive generation after flattening all three axes would yield a prohibitively long sequence (length 18432). We therefore factorize the decoding process. First, each frequency column $\mathbf{Q}_{:,t,d} \in \mathbb{N}^{32}$ is treated as a single decoding unit. The remaining time and depth axes are then decoded using the following three strategies.

AudioLM-Like Approach — We flatten the remaining axes, then decode each column autoregressively across the flattened axis (Figure 4.4). In this way, we model the probability distribution of the indices as follows:

$$p(\mathbf{Q}) = \prod_{t,d,f} p(\mathbf{Q}_{f,t,d} \mid \mathbf{Q}_{:,<t,:}, \mathbf{Q}_{:,<d,:}). \quad (4.1)$$

In the implementation, we initialize generation with a start column $\mathbf{x}_S \in \mathbb{R}^{384 \times 32 \times 1}$. We pass it to a Time-Frequency Transformer (TF-Transformer) to obtain a feature with the same shape as the input. With a linear projection, we predict the first

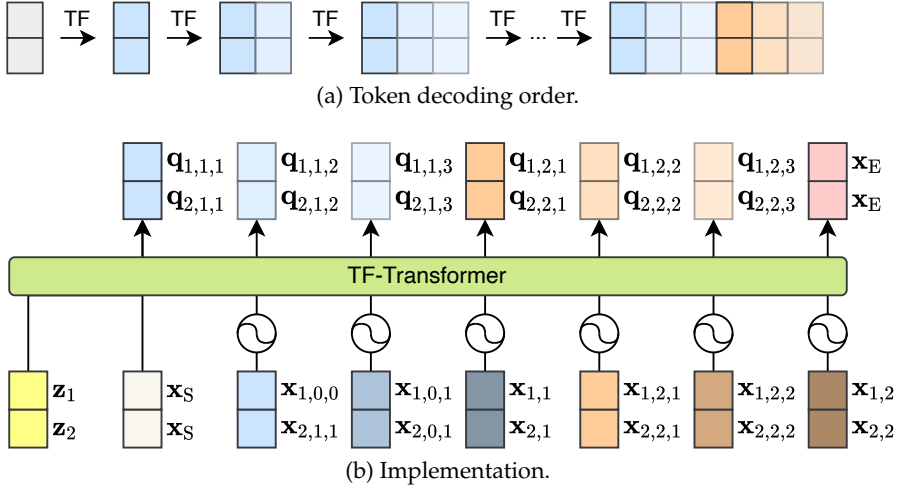


Figure 4.4: AudioLM-like token generation strategy.

column $\mathbf{Q}_{:,0,0} \in \mathbb{N}^{32}$ of the index tensor. The predicted indices are converted back to the latent with the codebook lookup and preprocessed with a linear layer, resulting in a transformer input $\mathbf{x}_{0,0} \in \mathbb{R}^{384 \times 32 \times 1}$. We concatenate it with the start column $\mathbf{x}_S | \mathbf{x}_{0,0}$, pass it to the TF-Transformer, and predict the next column $\mathbf{Q}_{:,0,1}$. We repeat this autoregressively, decoding $\mathbf{Q}_{:,0,2}$, $\mathbf{Q}_{:,1,0}$, $\mathbf{Q}_{:,1,1}$, and so on, until the model predicts an end-of-sequence. This approach follows AudioLM [86], extended to generate a frequency column at each step.

RQ-Transformer-Like Approach — Keeping the previous decoding order, separate models can be used to decode across time and depth (Figure 4.5).

$$p(\mathbf{Q}) = \prod_{t,f} p(\mathbf{Q}_{f,t,0} \mid \mathbf{Q}_{:,<t,:}) \prod_d p(\mathbf{Q}_{f,t,d} \mid \mathbf{Q}_{f,t,<d}, \mathbf{Q}_{:,<t,:}). \quad (4.2)$$

Specifically, the partially decoded latent $\mathbf{S}_{:,:,<t}^q$ is linearly projected to $\mathbb{R}^{384 \times 32 \times (t-1)}$ and processed by a TF-Transformer. Each output feature $\mathbf{o}_{f,t} \in \mathbb{R}^{384}$ is then passed to another transformer, called the Depth Transformer, to independently and autoregressively estimate the indices $\mathbf{Q}_{f,t,0}$, $\mathbf{Q}_{f,t,1}$, and $\mathbf{Q}_{f,t,2}$. This setup reduces the maximum input sequence length of the TF-Transformer from 576 to 192. Instead, the more lightweight Depth Transformer is evaluated 576 times, reducing the total memory and computation.

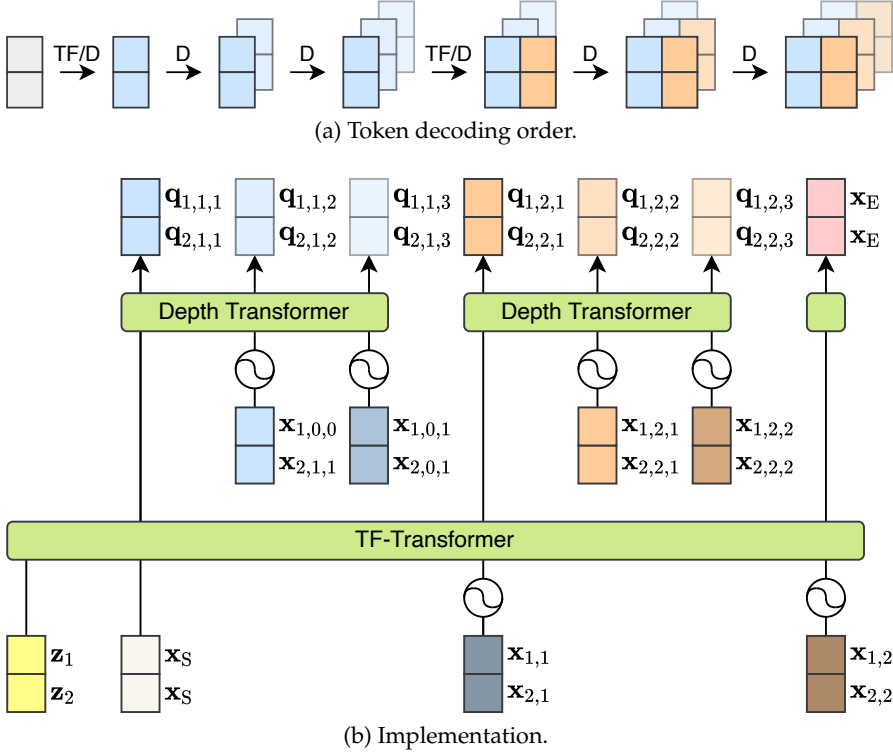


Figure 4.5: **RQ-Transformer-like token generation strategy.**

Except for the frequency axis, this approach follows RQ-Transformer [89].

VALL-E-Like Approach — The number of forward passes can be further reduced following VALL-E [87]. Instead of interleaving the time and depth axes, we first predict the depth-0 indices $\mathbf{Q}_{:::,0}$, then predict the remaining depths, $\mathbf{Q}_{:::,1}$ and $\mathbf{Q}_{:::,2}$, in two passes (Figure 4.6).

$$p(\mathbf{Q}) = \prod_{t,f} p(\mathbf{Q}_{f,t,0} \mid \mathbf{Q}_{:, < t, 0}) \prod_d p(\mathbf{Q}_{:::,d} \mid \mathbf{Q}_{:::, < d}). \quad (4.3)$$

The depth-0 decoding is identical to the first AudioLM-like approach. The decoded depth-0 tokens are converted back to a latent, linearly projected to $\mathbb{R}^{384 \times 32 \times 192}$, and passed to another TF-Transformer without the causal mask to estimate the following $\mathbf{Q}_{:::,1}$ and $\mathbf{Q}_{:::,2}$. We additionally append a depth embedding to the input of the second transformer.

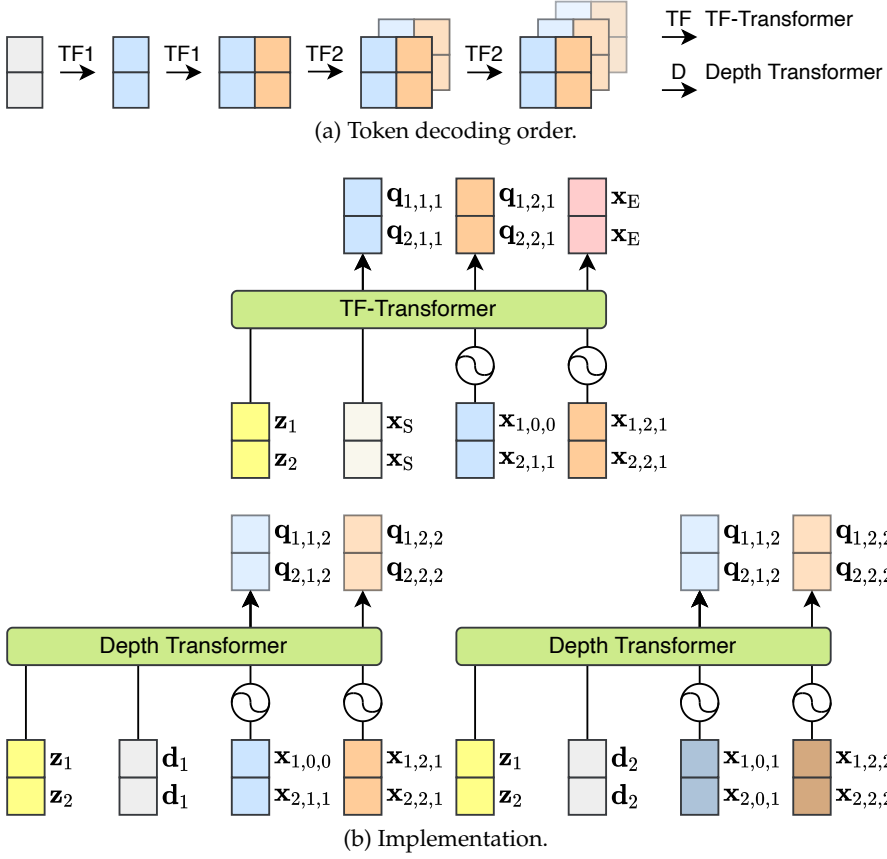


Figure 4.6: VALL-E-like token generation strategy.

4.3.2 Transformer Architecture

Figure 4.7 shows the axial transformer architecture used for token generation. The left part is the TF-Transformer, a stack of four blocks composed of a frequency-axis encoder layer and a time-axis decoder-only layer, which treat the time and frequency axes as batch axes, respectively. In this setup, the former exchanges information between different frequency bands, while the latter autoregressively processes the feature across time (and optional flattened depth). The right part is the Depth Transformer, which decodes the residual-quantizer depth axis conditioned on the TF-Transformer output. The Depth Transformer is a two-layer decoder-only model. All layers have pre-layer normalization [200], 384/1536 input/feed-forward dimension, 4 heads, and dropout of 0.1.

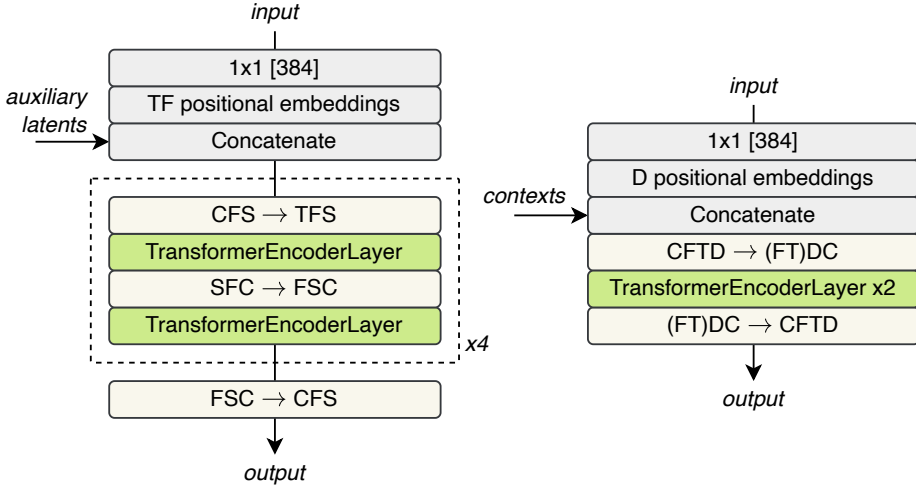


Figure 4.7: Axial transformer architecture.

4.3.3 Inference

At inference time, we employ top- k and top- p sampling, where the hyperparameters are set to $k = 10$ and $p = 0.995$. The decoded indices are converted back to the features, then fed into the frozen RQ-VAE decoder to obtain the spectrogram $\mathbf{H}$. We convert the spectrogram back to the time-domain RIR h with Griffin-Lim [201], initialized from the predicted real/imaginary components.

4.4 Conditional Generation with Reference Encoder

4.4.1 Reference Encoder

Providing conditioning information to the transformers casts various RIR estimation problems as conditional generative modeling. If we provide an RIR h as a reference, the task becomes *analysis-synthesis*. If reverberant speech $y = h * x$ is given, we tackle *blind estimation*. To condition the reference, we first encode it with a reference encoder. The blind estimation and acoustic matching tasks require extracting reverberation cues from speech, so we adapt the encoder part of MFTAA-Net [202], a speech enhancement architecture with time-frequency dilated convolutional layers and time-frequency axial self-attention layers. We re-

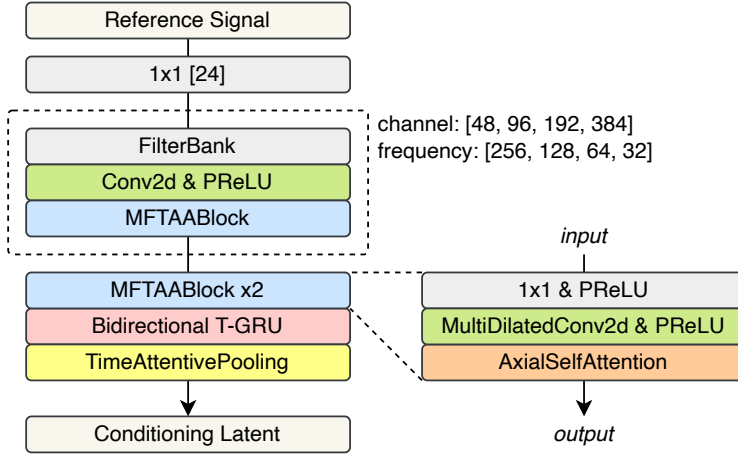


Figure 4.8: **Reference encoder architecture.**

place its frequency-axis strided downsampling with nonlinear filter bank matrices that follow the same Mel-like frequency mapping as the RQ-VAE, move the convolutional stride to the time axis, and double the number of channels. To the output of the modified MFTAA encoder, we apply attentive pooling across the time axis with 4 learnable queries, resulting in a latent tensor $\mathbf{z} \in \mathbb{R}^{384 \times 32 \times 4}$, i.e., 4 latent vectors $\mathbf{z}_f \in \mathbb{R}^{384 \times 4}$ for each frequency bin. The conditioning operation is a frequency-axis concatenation with the TF-Transformer input. As the reference encoder’s frequency mapping matches the RQ-VAE’s, the f^{th} band latent $\mathbf{z}_f$ is used to estimate the corresponding band’s indices $\mathbf{Q}_{f,:} \in \mathbb{N}^{192 \times 3}$. This band-aligned conditioning follows the LTI view of RIRs: by the convolution theorem, reverberation applies a band-wise spectral transformation to speech, so the conditioning features extracted from each frequency band are paired with the same band of the RIR token sequence. Figure 4.8 shows the resulting architecture. Its AxialSelfAttention consists of frequency-axis and time-axis self-attention, and its MultiDilatedConv2d is a 4-group two-dimensional convolutional layer, with a kernel size of (3, 7), a stride of (1, 2), and dilation factors [1, 2, 4, 8] assigned to the groups.

4.4.2 Acoustic Matching

We further tackle an *acoustic matching* problem: we predict the RIR h from a dry source x^s and wet target $y^t = h * x^t$, where the source and target dry signals are not necessarily the same. We encode both signals with the reference encoder and subtract the two embeddings to obtain the latent $\mathbf{z} = \mathbf{z}^t - \mathbf{z}^s$, which conditions the transformers. This trains the model to infer the relative acoustic transform from source to target. We find this setup more practical than blind estimation for applications where the estimated RIR is applied to another dry signal.

4.5 Experiments

4.5.1 Data

Room Impulse Responses — The RIR corpus combines the public datasets that form the RIR entries in Figure 4.9: ACE [35], the Aachen Impulse Response (AIR) database [203], BUT ReverbDB [204], the C4DM RIR database [205], the coupled-room dataset [182], the Detmold spatial RIR database [206], Echothief¹, the FOA-MEIR dataset [207], Fokke², the BBC Maida Vale Recording Studios dataset [208], the MIT Reverb Survey dataset [194], MYriAD v2 [209], OpenAIR [34], the Palimpsest RIR database³, and the RWCP sound scene database [210]. All of the RIRs are from real spaces, ranging from domestic environments, e.g., living rooms, to musical ones, e.g., concert halls. We collected 15k RIRs from 559 rooms and split them into 496/16/47 for the training/validation/test.

Microphone Impulse Responses — We used additional microphone IRs (MicIRs) [211], also including Vintage MicIRs⁴, to mimic the colored magnitude responses of real-world recordings. These two MicIR sources appear as the modern and vintage microphone entries in Figure 4.9. We collected a total of 8.4k MicIRs from 82 microphones and split them into 72/3/7. Figure 4.9 reports reverberation parameter statistics of the collected RIR/MicIR data. The green curve shows the

¹<http://www.echothief.com>

²<https://fokkie.home.xs4all.nl/IR.htm>

³<https://research.kent.ac.uk/sonic-palimpsest/impulse-responses>

⁴<https://micirp.blogspot.com>

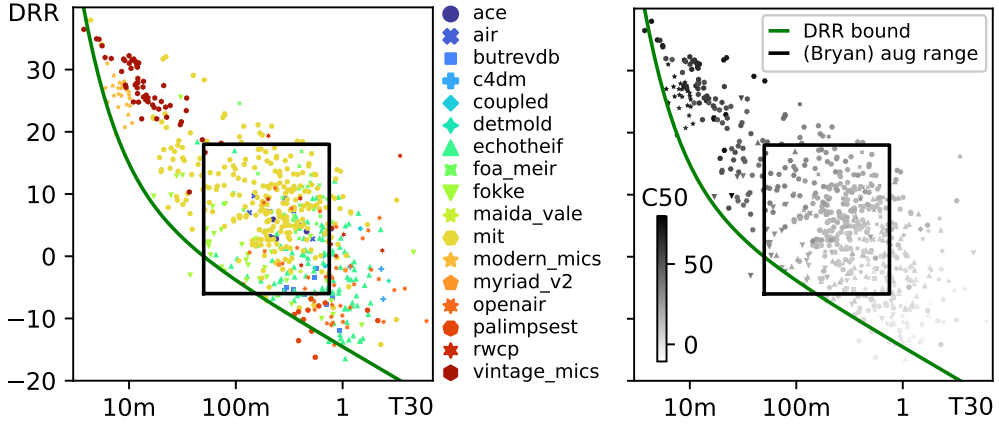


Figure 4.9: Reverberation parameter statistics of the IR datasets.

DRR expected under a single-slope exponential decay for a given T_{30} , which can be read as a lower-bound-like reference when multi-slope decay is absent. The black rectangle marks the T_{30} –DRR augmentation range used in the RIR augmentation study of Bryan [121].

Data Augmentation — Although the datasets above cover many real spaces, their combined coverage is still sparse relative to the diversity of room sizes, decay shapes, microphone responses, and recording conditions encountered in practice. Data augmentation is therefore a central part of the training pipeline rather than a minor regularization step. We modified the T_{30} and DRR by windowing/scaling [121], shifted pitch by resampling, applied a parametric equalizer to transform the overall magnitude response, and mixed ambisonic IR channels to mono. We applied the same augmentation procedure to both RIRs and MicIRs, but with different hyperparameters. Furthermore, we convolved an augmented RIR h^r with a MicIR h^m to obtain another IR $h^r * h^m$ and used it for target reverberant speech $y = h^r * h^m * x$. For acoustic matching, we optionally applied the MicIR h^m to the source speech to diversify the dry-signal distribution. In this case, we trained the models to estimate only the RIR h^r . All augmentations were performed on the fly, and we randomly chose whether to apply each of them.

Dry Speech — The usable speech or singing datasets depend on whether their recording conditions are consistent across utterances and speakers, as summa-

Table 4.1: IR estimation tasks and usable speech/singing dataset categories.

Task\Category	①	②	③	④
Blind estimation	✓			
Acoustic matching:				
– Different speaker, different utterance	✓	✓		
– Same speaker, different utterance	✓	✓	✓	
– Same speaker, same utterance	✓	✓	✓	✓

① *anchor*, ② *consistent*, ③ *in-speaker consistent*, ④ *inconsistent*.

rized in Table 4.1. The *anchor* category consists of DAPS [212], whose speakers were recorded in a common semi-anechoic condition; this makes it usable for blind estimation, where no separate dry source is provided. Among the consistent datasets, we chose DAPS as the anchor because it offered the best balance of diversity, recording quality, and quantity for this purpose. *Consistent* datasets share a fixed recording environment across speakers, so they can provide source/target pairs with different speakers for acoustic matching; this group includes VCTK [122], NUS [213], PTDB-TUG [214], JSUT [215], OpenSinger [216], NHSS [217], and Kiritan [218]. *In-speaker consistent* datasets are recorded consistently within each speaker but not necessarily across speakers, so they support same-speaker acoustic matching but not different-speaker matching; this group includes Att-HACK [219], TSP [220], HUI-German [221], SLR30, 37, 41–44 [222], SLR63–66, 78, 79 [223], and SLR70 [224]. We did not use datasets in the inconsistent category. We split DAPS speakers into 16/2/2 for training/evaluation/test, split the acoustic matching speakers into 3972/51 for training/validation, and used NUS solely for test. We used 3 and 12 seconds of dry signals for training and evaluations, respectively.

4.5.2 Metrics

Objective Metrics — We report log spectral distance (LSD), mean absolute error (MAE) of the energy decay relief (EDR) [74], and multi-scale spectral (MSS) loss [83, 82]. We also evaluate octave-band differences in T_{30} , DRR, and C_{50} (center frequencies 125Hz–8kHz), reporting median values.

Subjective Listening Test — We measured subjective reverberation similarity scores (SUBJ) with a standard MUSHRA-like listening test [129]. Eight subjects with critical listening experience rated the similarity between the reference reverberant speech and the dry speech convolved with estimated IRs, using an average of test RIRs as a low anchor (10 sets, 2 training sessions).

4.5.3 Training

We trained the RQ-VAE with mean square error, with the magnitude and complex STFT terms weighted 3 : 1. We trained the token generation stage with cross-entropy loss and label smoothing of 0.001. We used the AdamW optimizer [185], 5k linear warm-up steps with a $1e-4$ peak learning rate, and halved the learning rate every 50k steps. We set the batch sizes to 16 and 6 for the first and second stages, respectively. We stopped training when the validation metric did not improve over several epochs.

4.6 Evaluation Results

4.6.1 Autoencoding

Table 4.2 reports the evaluation results. We first assessed the autoencoder model’s performance with and without the quantization layer. Without quantization, the reconstruction is nearly perfect, showing 0.54dB LSD and 2.2% T_{30} error. Adding the vector quantization layer increases these errors to 1.85dB and 9.6%, indicating non-negligible quantization error. However, residual quantization consistently improves all metrics and achieves a SUBJ score of $87.6_{\pm 4.3}$, close to the hidden reference’s $90.5_{\pm 3.1}$. These reconstruction results provide approximate upper bounds for the transformer models, which must predict the RQ-VAE indices before decoding.

4.6.2 Analysis-Synthesis

We first trained the proposed transformer models to perform analysis-synthesis, where the reference itself is the target RIR. This setting evaluates whether the ref-

Methods	Objective metrics ($\downarrow$)						SUBJ ($\uparrow$)
	LSD	MSS	EDR	T_{30}	DRR	C_{50}	
Hidden reference	—	—	—	—	—	—	90.5 $\pm$ 3.1
Average test RIR	11.3	.206	19.4	4.94	9.21	13.4	14.9 $\pm$ 4.0
<i>Autoencoding results</i>							
No quantization	0.54	.010	0.61	.022	0.39	0.35	—
Vector quantization	1.85	.050	1.48	.096	1.63	1.37	—
Residual quantization	1.13	.025	0.76	.034	0.66	0.59	87.6 $\pm$ 4.3
<i>Analysis-synthesis results</i>							
AudioLM-like	2.15	.080	1.75	.063	1.45	0.97	84.0 $\pm$ 4.0
RQ-Transformer-like	2.31	.078	1.82	.061	1.51	1.05	—
VALL-E-like	2.28	.083	1.84	.074	1.58	1.23	—
Differentiable FVN	5.95	.098	1.88	.124	6.53	1.27	—
<i>Blind estimation results</i>							
AudioLM-like	2.67	.122	3.48	.121	3.25	1.69	74.9 $\pm$ 5.1
RQ-Transformer-like	2.84	.120	3.55	.123	3.78	1.63	—
VALL-E-like	2.83	.117	3.77	.149	2.82	1.60	—
Differentiable FVN	6.49	.112	3.20	.196	7.85	2.35	—
<i>Acoustic matching results</i>							
AudioLM-like	2.81	.124	3.65	.157	3.47	1.85	—
— Same speaker	2.76	.124	3.60	.157	3.03	1.75	70.9 $\pm$ 6.0
— Different speaker	2.85	.125	3.70	.158	3.74	1.87	66.2 $\pm$ 6.3
RQ-Transformer-like	3.05	.120	4.23	.168	2.96	1.79	70.4 $\pm$ 5.1
VALL-E-like	3.06	.123	4.57	.173	3.39	1.86	66.1 $\pm$ 6.0
Differentiable FVN	8.02	.115	3.69	.241	8.16	2.79	57.3 $\pm$ 5.3
Non-AR, discrete	3.69	.133	5.08	.202	5.45	4.53	45.3 $\pm$ 6.6
Non-AR, continuous	4.24	.132	5.33	.275	10.1	9.73	—

* LSD, EDR, DRR and C_{50} are mean absolute error (MAE) in dB.

* We report T_{30} error in terms of ratio.

* The subjective scores are mean values with 95% confidence intervals.

Table 4.2: **Evaluation results of the proposed methods and baselines.**

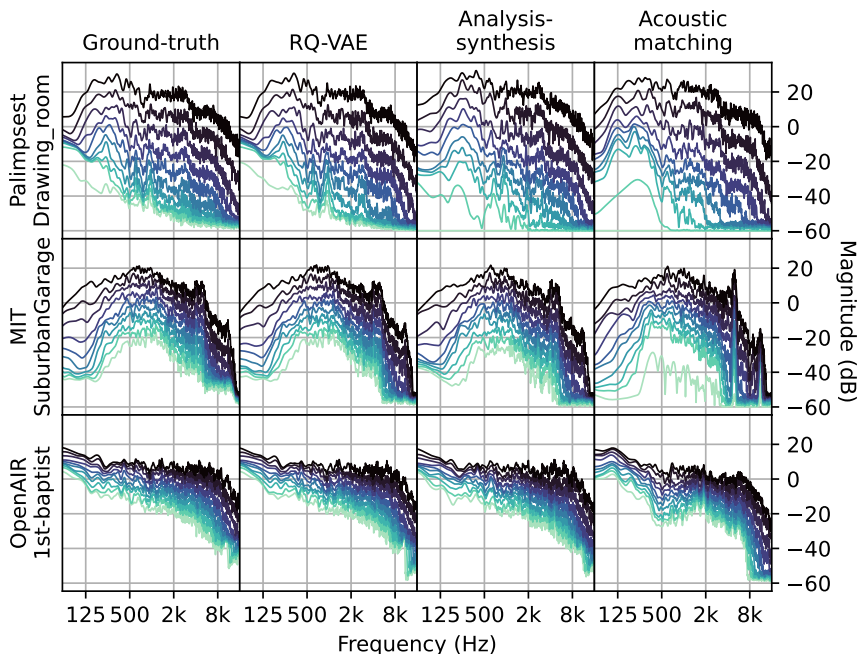


Figure 4.10: Energy decay relief waterfalls of ground-truth RIRs, RQ-VAE reconstructions, and RQ-Transformer estimations.

erence encoder and token generator can reconstruct an RIR when the conditioning signal directly contains the target acoustic response, before introducing the additional ambiguity of extracting the RIR from speech. As expected, these models report slightly worse results than the RQ-VAE, which has access to the ground-truth tokens. Nevertheless, the reconstruction remains perceptually similar to the reference; the AudioLM-like model achieved a SUBJ score of 84.0 ± 4.0 . Figure 4.10 qualitatively compares the EDR waterfalls of ground-truth RIRs, RQ-VAE reconstructions, and RQ-Transformer estimations, complementing the aggregate metrics in Table 4.2.

4.6.3 Blind Estimation

Estimating RIRs from reverberant speech is more challenging than analysis-synthesis, and the evaluation results indicate that the estimation error can be perceptually noticeable in terms of reverberation parameters. For example, all

three transformer models report T_{30} error greater than 12%. The DRR errors also increase significantly from the previous analysis-synthesis results (maximum 2.27dB increase). However, the LSD increases by only about 0.5dB, indicating that our models can still accurately capture the overall time-averaged characteristics. Figures 4.11 and 4.12 provide a more granular view of these errors through per-RIR scatter plots of estimated versus ground-truth reverberation parameters and per-octave-band comparisons of ground-truth and predicted magnitude responses across the same test set.

4.6.4 Acoustic Matching

Acoustic matching provides more information than blind estimation because the dry source is available; at the same time, it is also more general because the source can have arbitrary acoustic conditions, e.g., microphone coloration, which makes the task challenging. Indeed, the acoustic matching models report slightly worse results than the blind estimation counterparts, although the differences are not large, e.g., at most 0.8dB increase in EDR error. Furthermore, we analyzed whether providing different speakers for the source/target affects the performance. As shown in the AudioLM-like model’s results, the different-speaker scenario reports slightly higher errors. For the other models, we report the aggregated objective metrics of the same-speaker and different-speaker results, while the SUBJ scores are different-speaker results.

4.6.5 Decoding Strategy Comparison

In general, the more computationally expensive methods performed better, e.g., the AudioLM-like model outperformed the VALL-E-like model in most objective metrics, but the differences were relatively small, especially in the acoustic matching scenario. The SUBJ scores also indicate that such differences might not be perceptually significant ($66.2_{\pm 6.3}$, $70.4_{\pm 5.1}$, and $66.1_{\pm 6.0}$). This may be because the reference encoder is the dominant bottleneck once all transformer variants receive similarly noisy conditioning estimates. Moreover, the AudioLM-like model took approximately 3.5 and 6.2 times more inference time than the RQ-Transformer-like

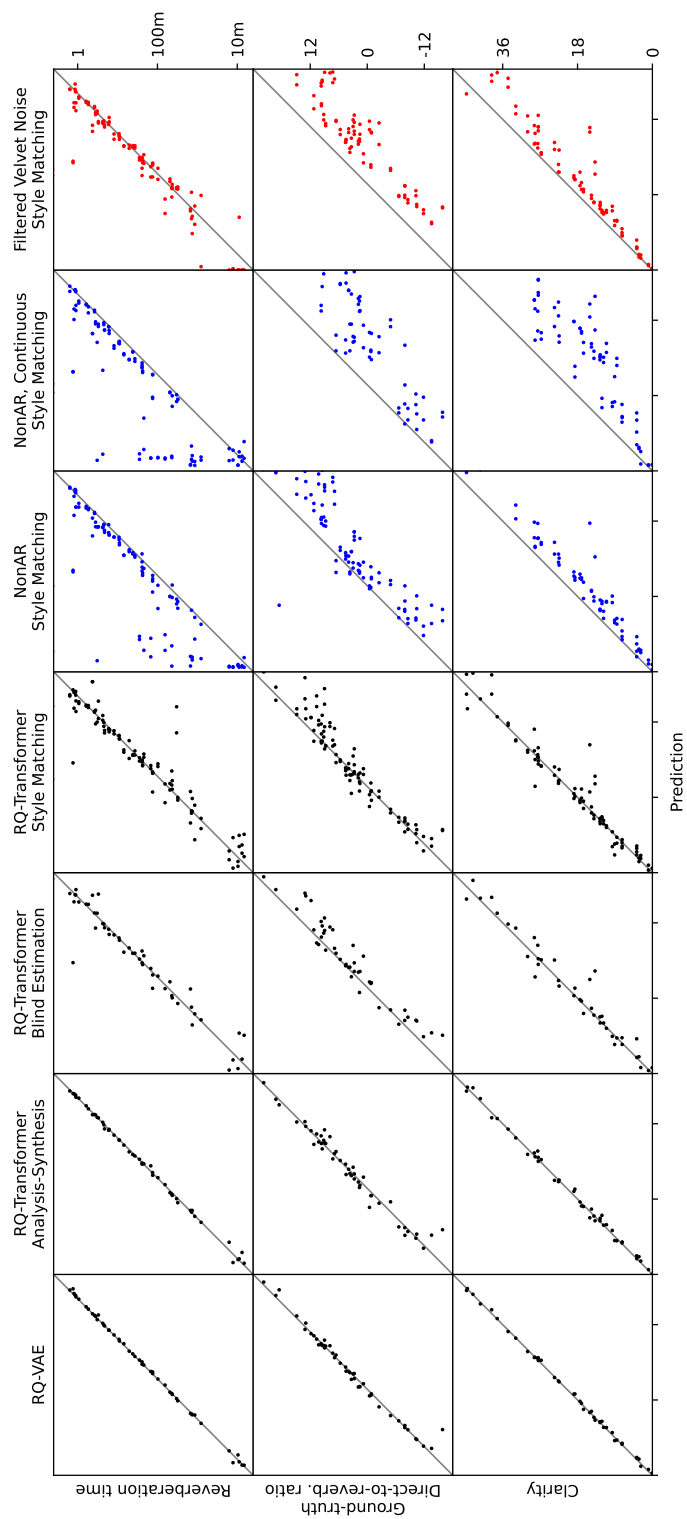


Figure 4.11: Scatter plot of estimated versus ground-truth reverberation parameters across test RIRs.

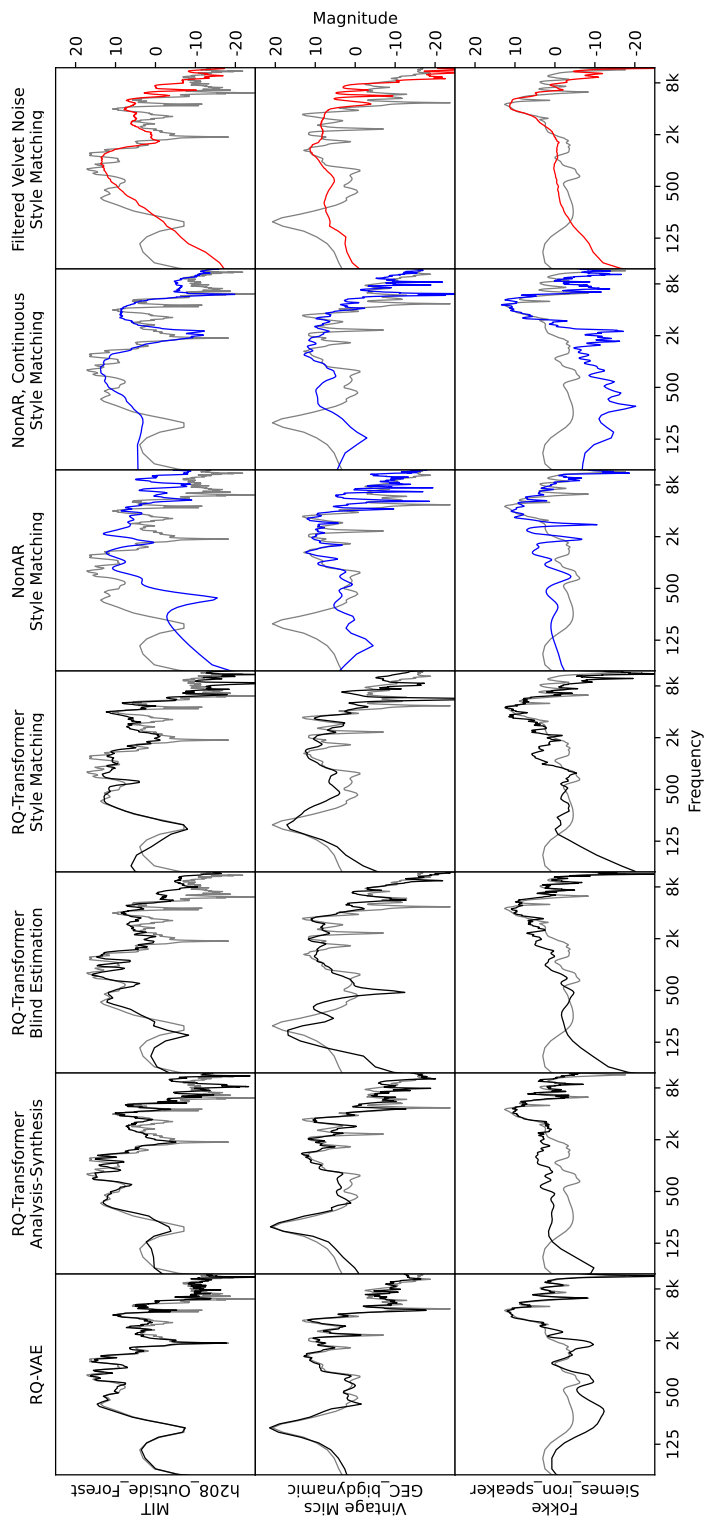


Figure 4.12: Ground-truth versus predicted per-octave-band magnitude responses, same test set.

and VALL-E-like models, respectively. Therefore, the optimal model choice may depend on the available computation budget.

4.6.6 Ablation Study and Baselines

To evaluate whether autoregressive modeling and discretization contribute to estimation performance, we trained an RQ-Transformer-like backbone to predict all indices simultaneously by removing the causal attention mask (Non-AR, discrete). This setup reports clearly degraded performance. We also trained a non-causal TF-Transformer to estimate the continuous latent from the autoencoder (Non-AR, continuous); the results are even worse, especially in terms of DRR and C_{50} error. We also attempted the autoregressive model with continuous features, but the training was unstable. Therefore, discretization and autoregressive modeling were essential for performance under the current framework. We compared against the filtered velvet noise (FVN) baseline from Chapter 2 [83, 73]. Because the FVN baseline was optimized with MSS loss, it performs competitively on MSS, but poorly on LSD due to the limited expressive power of the generator. It also showed high DRR and C_{50} errors, similar to the other baselines. In terms of the SUBJ scores, the differences between the autoregressive models and the non-autoregressive baselines were statistically significant (all $p < 1.67 \times 10^{-3}$ under the pairwise paired t-tests with Bonferroni correction). The feedforward estimation and regression objective likely degrade transient and onset modeling; verifying this is left for future work. Figure 4.13 compares the MUSHRA-like subjective similarity scores with objective losses and reverberation parameter errors.

4.7 Hybrid Extension: Efficient Acoustic Scene Transfer with Differentiable Decoding

The framework above estimates impulse responses by decoding discrete RQ-VAE tokens with conditional autoregressive transformers and reconstructing a time-domain IR from the decoded representation. This formulation is expressive, but its token budget grows with the modeled IR length, and CPU inference is dominated by transformer decoding. We therefore developed an acoustic scene transfer

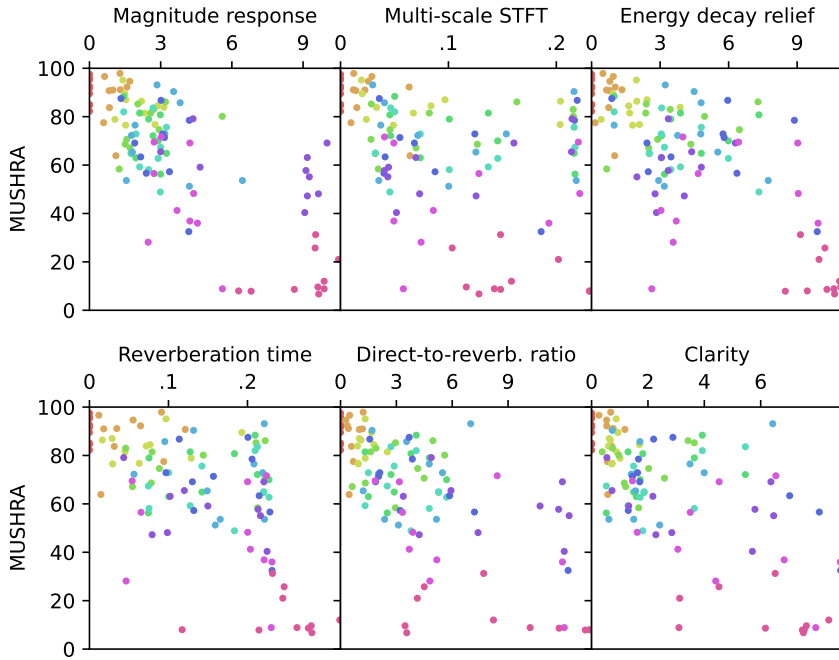


Figure 4.13: **Scatter plots between MUSHRA-like subjective similarity scores and objective losses or reverberation parameter errors.**

extension that pairs the generative pipeline of this chapter with differentiable signal processing structure (Chapter 2). The extension is motivated by a structural asymmetry within the impulse response: the early reflection portion is comparatively deterministic and benefits from a richer autoregressive description, whereas the late reverberation tail is quasi-stationary and stochastic and can be represented more compactly with a filtered-noise decoder. In this sense, the extension uses the learned generator as a data-driven prior for measured RIR structure and the differentiable decoder as a domain prior for late reverberation, improving modeling efficiency by assigning each temporal region to a better-matched representation.

4.7.1 Factorized Codec with a Filtered-Noise Late Reverberation Decoder

Exploiting this asymmetry, we factorize the codec along the time axis (Figure 4.14). The first $\sim 150\text{ms}$ of the IR, the early reflection (ER) part where deterministic

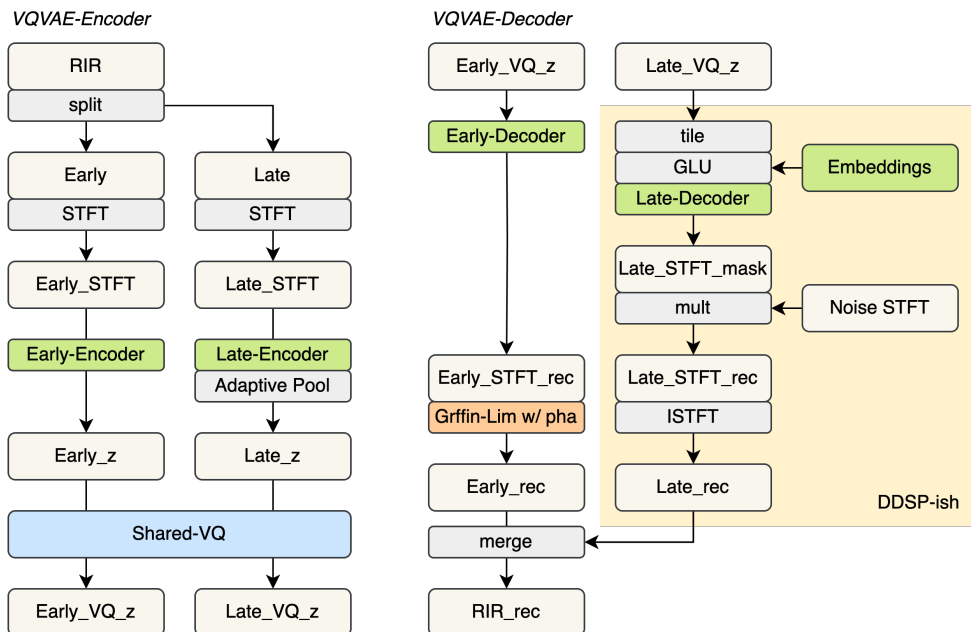


Figure 4.14: **Hybrid codec architecture.**

structure dominates, is encoded with one autoencoder, while the remaining $\sim 3.4s$ of late reverberation (LR) is encoded with a separate, more compressed branch whose decoder implements a differentiable filtered-noise model related to the artificial reverberation models of Chapter 2.

Early reflection branch — Departing from the residual quantization used elsewhere in this chapter, the ER branch uses a single-codebook vector-quantized VAE (VQ-VAE) with 4096 codes, versus 3 codebooks of 256 entries in the RQ-VAE, trading a small expressivity reduction for a substantially simpler autoregressive decoder with no depth axis, and hence no depth transformer. The encoder produces a 32×40 token tensor.

Late reverberation branch — The LR encoder produces a raw latent of size 32×144 , which is then reduced to 32×25 by non-uniform mean pooling along the time axis: pooling kernels grow toward the tail, exploiting the increasing temporal stationarity of the LR. The pooled latent is quantized with the same 4,096-entry codebook used by the ER branch. To decode, we revert the non-uniform pool-

ing by tiling, multiply the tiled latent elementwise with a learnable embedding tensor, which acts as a stationary band-energy template, and apply a stack of convolutional upsampling layers to obtain a magnitude mask in the STFT domain. This mask is multiplied with the magnitude STFT of white noise, yielding a time-varying filtered-noise signal in the spirit of classical late reverberation models. STFT-domain magnitude multiplication implies an underlying circular convolution, which we find acceptable given the LR’s quasi-stationary character.

Training and integration — The factorized VQ-VAE is trained with a power-compressed STFT-domain MSE term for the ER branch and MSS losses for both the ER and LR branches. At inference, the ER and LR token tensors are concatenated into a single 32×65 tensor that the autoregressive transformer decodes, versus 32×192 in the baseline of this chapter, and the modeled IR length grows from 1.1s to 3.55s. After separate spectrogram-to-time-domain conversion of the two branches, a 5ms crossfade joins them at the 150ms boundary.

Reference encoder — The extension also changes the conditioning encoder used for acoustic scene transfer. Instead of subtracting source and target embeddings from a shared encoder, it uses separate source and target encoders and conditions the target encoder on the source latent through gated linear units. The encoder backbone is changed from the previous MFTAA-Net-style model to an ECAPA2-like [225] model operating on Mel-domain features derived from power-compressed STFT real, imaginary, and magnitude channels. This update targets more expressive extraction of the target acoustic environment from reference speech, rather than adding a separate acoustic field model.

4.7.2 Time-Frequency Delay Pattern

The factorized parallel generation of all tokens at the same time step t is efficient, but it assumes that neighboring frequency bands can be sampled independently at that step. This is a reasonable approximation from the band-wise LTI perspective, yet it is not fully consistent with the codec: the encoder observes neighboring bands when constructing each token, and the decoder reconstructs adjacent bands jointly. Sampling the bands independently at each step can therefore produce inconsis-

tencies between adjacent bands, degrading the decoded output. To mitigate this without giving up the parallel-decoding speedup, we introduce a *time-frequency delay* pattern, which reintroduces a limited dependency between adjacent bands: the token at frequency f is delayed by Δ_f time steps before generation, so that

$$p(\mathbf{Q}) = \prod_{t,f} p(\mathbf{Q}_{f,t} \mid \{\mathbf{Q}_{\tilde{f},\tilde{t}} : \tilde{f} + \tilde{t} < f + t\}). \quad (4.4)$$

We pre-shift the entire index tensor, filling new positions with a null token, perform the same factorized autoregression, and revert the shift after generation; the end-of-sequence token is also delayed, and generation halts once EOS appears in every band. The delay adds $F - 1$ steps to the autoregression, but combined with the factorized VQ-VAE the net effective number of steps drops from 192 to 96.

4.7.3 Efficient CPU Inference

The hybrid formulation already yields cheaper inference: a smaller token budget, a lighter LR generator, and the elimination of the depth transformer. On top of this we apply standard inference engineering techniques. KV-caching in the time-axis decoder removes redundant computation across autoregressive steps and incidentally eliminates the costly $T \leftrightarrow F$ permutation that previously occurred between time- and frequency-axis transformer layers. Batch-norm folding into preceding convolutions reduces operation count throughout the encoder and decoder. Partial-inference paths skip the source encoder when the dry source signal is unavailable, degrading gracefully to blind estimation, or skip the LR decoder when the autoregressive transformer early-stops before reaching the LR portion of the token tensor; the model is exported to ONNX as separate components for these conditional paths. Together with the hybrid codec changes, these techniques reduce average CPU inference time from up to approximately one minute in the previous model to about 1.9s for the evaluation setup with source and target references both provided.

This trajectory exemplifies the hybrid combinations discussed in Section 1.5: differentiable processing supplies the structural prior that compresses the late reverberation description and yields an interpretable DSP component, while the

generative pipeline supplies the expressive autoregressive coverage needed for early reflections and heterogeneous measured IRs. The current result does not by itself establish real-time operation, but it substantially reduces the practical inference cost for post-production use.

4.8 Conclusion

We explored a two-stage framework for RIR generation and tackled several RIR estimation tasks via conditioned autoregressive decoding. We first obtained a discrete time-frequency token space of RIRs with RQ-VAE, then decoded the token indices with transformer variants. The evaluation results on analysis-synthesis, blind estimation, and acoustic matching show that the proposed generator performs favorably against structured and non-autoregressive baselines under objective metrics and listening tests. Consistent with broader findings in neural audio generation, both quantization and autoregressive modeling proved beneficial for performance; yet, *how much* autoregression yields the optimal performance-efficiency tradeoff remains an open question. Further work should include improving this tradeoff with lightweight models and optimized autoregressive modeling strategies, as well as evaluating acoustic scene transfer [226, 227] under more realistic source and target conditions.

Chapter 5

Differentiable and Efficient Computation of Audio Processing Graphs

5.1 Introduction

So far, this thesis has considered acoustic systems whose structure was fixed: compact reverberators, geometric acoustics models, or single RIR generators. We now turn to compositional audio processing, where the signal flow itself is part of the system description. Under the umbrella of *differentiable signal processing* [47, 49], numerous attempts have imported existing audio processors to automatic differentiation frameworks, e.g., `PyTorch` [183]. Differentiable processors, as standalone modules, allow gradient-based optimization of their parameters. Alternatively, they can be used to train a neural network as a parameter estimator [83, 57]. In either case, as the processors are identical to or approximate the real-world ones, the obtained parameters are easy to interpret and control. To further leverage this advantage of differentiable signal processing, it would be desirable to consider the *composition* of processors since it is a standard real-world practice. In a more general setting, this composition can be represented in a *graph* format [228]. Existing differentiable audio libraries [47, 90] are designed primarily around differentiable processors or specialized processing structures. Their scope is therefore complementary to the one considered here: they do not aim to cover the combination of mutable heterogeneous graphs, tensorized graph representations, graph-changing optimization loops, and batching over graph structure.

With this different target, this chapter formulates differentiable audio processing graphs as a practical framework for graph-structured audio systems. The framework is consolidated in `GRAFX`, an open-source `PyTorch` library. Its graph abstraction is fully multi-input multi-output (MIMO): each processor may expose multiple graph inlets and outlets, and each edge specifies which outlet is routed to which inlet. This graph representation and its differentiable computation are used by the following two chapters: Chapter 6 specializes the framework to single-input single-output (SISO) mixing graphs with explicit dry/wet controls, while Chapter 7 returns to the MIMO graph abstraction for blind graph estimation. For the inverse problems considered in the following chapters, the framework must provide two capabilities. First, it provides custom data structures for creating and modifying graphs, computing output signals, and converting graphs into tensors that can also be used as inputs to graph neural networks (GNNs) [228]. Second, it provides an optimized graph-computation algorithm that computes the output audio from graphs, source audio, and processor parameters. The resulting framework allows the graph to change at every optimization step. This is useful in multiple scenarios, e.g., training a GNN that predicts the parameters of given graphs [228] or pruning a graph with gradient descent (Chapter 6). Its batching is not merely source-level batching through a fixed graph: it groups same-type processors within a graph and can also batch different graphs by treating them as a single disconnected graph, instead of rendering every processor one by one. The remainder of the chapter specifies the graph representation and the computation strategy that makes the subsequent graph-optimization experiments practical.

5.2 Graph Definition and Representation

Recall from Section 1.2.4 that an audio processing graph is written as $G = (V, E)$ where V and E are the node and edge set, respectively. This chapter operationalizes that definition for differentiable computation. Each node $v_i \in V$ represents a processor f_i with a type t_i , e.g., reverb `r`. It takes M signal(s) $u_i[n]$ and a collection of parameter tensors p_i as input and produces N output(s) $y_i[n]$ (n denotes a time

index).

$$y_i^{(1)}[n], \dots, y_i^{(N)}[n] = f_i(u_i^{(1)}[n], \dots, u_i^{(M)}[n], p_i). \quad (5.1a)$$

$$u_i^{(l)}[n] = \sum_{(j,k) \in \mathcal{N}^+(i,l)} y_j^{(k)}[n]. \quad (5.1b)$$

Here, $\mathcal{N}^+(i, l)$ denotes a collection of nodes and channel indices that send their output signals to l^{th} input of node i . With this setup, each edge $e_{ij} \in E$ becomes a “cable” connecting two nodes. Note that each edge also requires a type attribute $t_{ij} = (k, l)$, a tuple of output and input port indices, unless every processor in a graph is SISO, i.e., $M = N = 1$. Here, SISO and MIMO refer to the number of graph inlets and outlets, not to the number of audio channels carried by each signal. In short, each node’s outputs can be computed by finding its inputs, aggregating those, and processing the sums with the parameters. The graph output can be obtained by repeating this procedure over all nodes in topological order, starting from input nodes i until we reach the output nodes o . We allow any graph except those with cycles, as feedback loops are hard to resolve and bottleneck the processing speed due to the forced sample-level recursion.

5.2.1 Mutable and Tensor Representations

There are two main use scenarios when handling the audio processing graphs. First, users may create and modify a graph. For this case, we provide a mutable data structure `GRAFX` (same as the library name). It inherits `MultiDiGraph` class from `networkx` [229] and provides additional functionalities, e.g., adding a serial chain of processors. Second, each graph can be used to compute output audio or fed into a GNN. In such cases, representing each graph as a collection of tensors is more convenient and efficient. Therefore, we provide a `GRAFXTensor` class, which is compatible with `Data` class from `torch_geometric` [230]. The following are the tensors we use to describe each graph. First, we have a node type vector $\mathbf{T}_V \in \mathbb{N}^{|V|}$, an edge index tensor $\mathbf{E} \in \mathbb{N}^{2 \times |E|}$, and an (optional) edge type tensor $\mathbf{T}_E \in \mathbb{N}^{2 \times |E|}$ where $|\cdot|$ denotes the size of a given set. All parameters are collected in a dictionary $\mathbf{P}$ whose key is a node type t and value $\mathbf{P}[t]$ contains the parameters of that type. In this chapter, we assume that the value is a single

tensor in a form $\mathbf{P}[t] \in \mathbb{R}^{|V_t| \times N_t}$ where $|V_t|$ and N_t are the number of nodes and parameters. In fact, we allow $\mathbf{P}[t]$ to be a tensor with more dimensions or even a dictionary of tensors; we omit such cases for simplicity. All sources are stacked to a single tensor $\mathbf{S} \in \mathbb{R}^{K \times C \times L}$ where K , C , and L are the number of sources, channels, and length, respectively. We ensure that all the tensors, $\mathbf{T}_V$, $\mathbf{T}_E$, $\mathbf{E}$, $\mathbf{P}$, and $\mathbf{S}$, share the same node order. For example, a k^{th} source $s_k[n]$ must correspond to the first k^{th} input i in the node type list $\mathbf{T}_V$. Likewise, an l^{th} type- t parameter $\mathbf{P}[t]_l \in \mathbb{R}^{N_t}$ must correspond to the l^{th} type t in the type list $\mathbf{T}_V$.

5.3 Batched Graph Rendering

For faster computation on the GPU, maximizing the parallelism is desirable, and batched processing is the most standard approach. There are three levels of batching in this setting, but only the latter two depend on the graph structure. *Source-level* batching processes multiple input sources with a single fixed graph and can be handled whenever every processor supports ordinary batched processing. *Node-level* batching is the graph-specific case: it processes multiple nodes with the same processor type simultaneously, even when those nodes appear at different positions in a heterogeneous graph.

Specifically, consider a sequence of $N_{\text{iter}} + 1$ node subsets $V_0, \dots, V_{N_{\text{iter}}} \subset V$ satisfying the following conditions.

- (i) It forms a *partition*: $\cup_n V_n = V$ and $V_n \cap V_m = \emptyset$ if $n \neq m$.
- (ii) It is *causal*: no path from $u \in V_n$ to $v \in V_m$ exists if $n \geq m$.
- (iii) Each subset V_n is *homogeneous*: it has only a single type t_n .

Then, we can compute a batch of output signals $\mathbf{Y}_n$ of each subset V_n sequentially, from $n = 0$ to N_{iter} . Consequently, we reduce the number of the gather-aggregate-process iterations from $|V|$ to N_{iter} (there is no processing for $n = 0$, as V_0 contains the input modules). Figure 5.1 shows an example. For a graph with $|V| = 21$ nodes (Figure 5.1a), we can obtain a sequence with $N_{\text{iter}} = 9$ (Figure 5.1b). Finally, equipped with this batched node processing, we can also achieve *graph-level* parallelism over different graphs; we can simply treat a batch of multiple graphs as

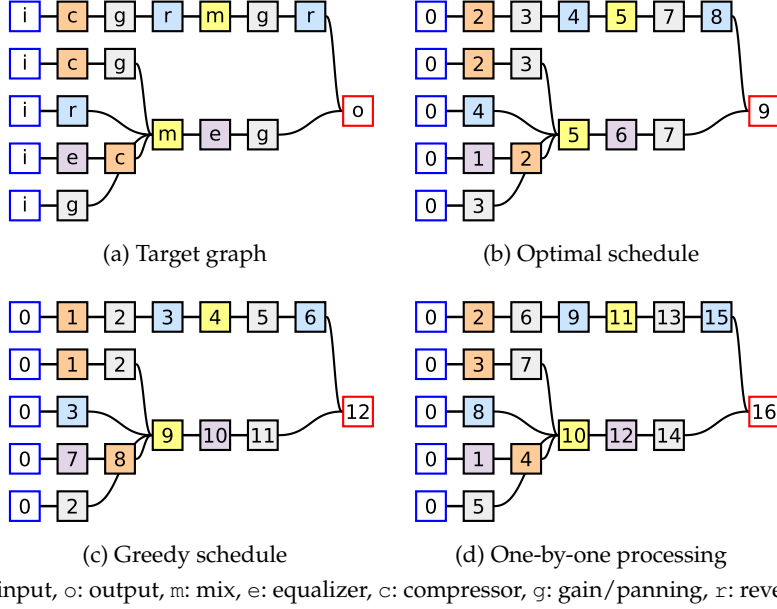


Figure 5.1: **Various schedules for batched node processing.** For each schedule, the processing orders are shown inside the nodes.

a single large disconnected graph. Therefore, we will focus on the batched *node* processing for the remainder of this section.

5.3.1 Type scheduling

For a maximized node-level parallelism, we want to find the shortest node subset sequence. This is a variant of the scheduling problem. First, we always choose a maximal subset V_i when the type t_i is fixed. This makes the subset sequence equivalent to a type string, e.g., *iecgrmegr* for Figure 5.1b. We also choose the first and the last subset, V_0 and $V_{N_{\text{iter}}}$, to have all of the input and output nodes, respectively. Since the search tree for the shortest sequence exponentially grows, the brute-force search is too expensive for most graphs. Instead, we may try the greedy method that chooses a type with the largest number of computable nodes (Figure 5.1c). However, this usually results in a longer sequence, thus slower processing. We can alleviate this issue with the beam search, i.e., keeping multiple best schedules as candidates instead of one. Intuitively, the batched node

Algorithm 1 Batch computation of audio processing graphs.

Input: Types $\mathbf{T}_V$ and $\mathbf{T}_E$, edges $\mathbf{E}$, parameters $\mathbf{P}$, and inputs $\mathbf{S}$

Output: Output signals $\mathbf{Y}$ and (optional) intermediate signals $\mathbf{U}$

```
1:  $\bar{\mathbf{T}}, N \leftarrow \text{ScheduleBatchedProcessing}(\mathbf{T}_V, \mathbf{E})$ 
2:  $\sigma \leftarrow \text{OptimizeNodeOrder}(\bar{\mathbf{T}}, \mathbf{T}_V, \mathbf{E})$ 
3:  $\mathbf{T}_V, \mathbf{E}, \mathbf{P} \leftarrow \text{Reorder}(\sigma, \mathbf{T}_V, \mathbf{E}, \mathbf{P})$ 
4:  $\mathbf{I}^G, \mathbf{I}^P, \mathbf{I}^A, \mathbf{I}^S \leftarrow \text{GetReadWriteIndex}(\bar{\mathbf{T}}, \mathbf{T}_V, \mathbf{E}, \mathbf{T}_E, \mathbf{P})$ 
5:  $\mathbf{U} \leftarrow \text{Initialize}(\mathbf{S}, \mathbf{T}_V)$ 
6: for  $n \leftarrow 1$  to  $N$  do
7:    $\bar{\mathbf{U}}_n \leftarrow \text{Gather}(\mathbf{U}, \mathbf{I}_n^G)$  ▷ index_select
8:    $\mathbf{U}_n \leftarrow \text{Aggregate}(\bar{\mathbf{U}}_n, \mathbf{I}_n^A)$  ▷ scatter
9:    $\mathbf{P}_n \leftarrow \text{Gather}(\mathbf{P}[\bar{t}_n], \mathbf{I}_n^P)$  ▷ slice
10:   $\mathbf{Y}_n \leftarrow \text{Process}(\bar{t}_n, \mathbf{U}_n, \mathbf{P}_n)$ 
11:   $\mathbf{U} \leftarrow \text{Store}(\mathbf{U}, \mathbf{Y}_n, \mathbf{I}_n^S)$  ▷ slice
12: end for
13:  $\mathbf{Y} \leftarrow \mathbf{Y}_N$ 
14: return  $\mathbf{Y}, \mathbf{U}$ 
```

processing is effective for graphs with fewer types and a certain structure, e.g., ones that apply processors in the same order for every input (Figure 5.1a).

5.3.2 Implementation details

Algorithm 1 obtains the output $\mathbf{Y}$ from the prescribed inputs; the numbers in parentheses below refer to the algorithm’s line numbers. First, we schedule the batched node processing and obtain a node type list $\bar{\mathbf{T}} \in \mathbb{N}^{N_{\text{iter}}+1}$ (1). Next, as the main batched processing loop (6-12) contains multiple memory reads/writes, we calculate the node reordering σ that achieves contiguous memory accesses and improves the computation speed (2). This procedure allows memory accesses via `slice`, as shown in the comments of Algorithm 1. After reordering the graph tensors with σ (3), we retrieve lists of indices, $\mathbf{I}^G$, $\mathbf{I}^P$, $\mathbf{I}^A$, and $\mathbf{I}^S$, used for the tensor read/writes in the main loop (4). Note that all these steps (1-4) are done on the CPU and, in most cases, in multiple separate threads. Therefore, they do not bottleneck the GPU and optimization. After the preprocessing, we create an intermediate output tensor $\mathbf{U}$, which will have a shape of $|V| \times C \times L$ if all processors are SISO systems or $N_{\text{sum}} \times C \times L$ where N_{sum} denotes the total number of outputs in the graph (5). As we put all the inputs to be the first partition V_0 , it can be initialized

with simple concatenation: $\mathbf{U} = \mathbf{S} \oplus \mathbf{0}$. The remaining repeats batched processing and necessary reads/writes (6-12). For each n^{th} iteration, we collect the previous outputs $\bar{\mathbf{U}}_n$ that are routed to the current partition nodes. We achieve this by accessing the intermediate tensor $\mathbf{U}$ with the index $\mathbf{I}_n^G$ with `index_select` (7). Then, we aggregate them using `scatter` if multiple edges are connected to some nodes (8). We can similarly obtain a parameter tensor $\mathbf{P}_n$ with its corresponding index $\mathbf{I}_n^P$. Especially, our node reordering (3) makes this a simple `slice`, faster than the usual `index_select`. With the obtained input signals $\mathbf{U}_n \in \mathbb{R}^{|V_n| \times C \times L}$ and parameters $\mathbf{P}_n \in \mathbb{R}^{|V_n| \times N_t}$, we batch-compute the node outputs $\mathbf{Y}_n$ (10). Then, we save them to the intermediate output tensor $\mathbf{U}$ with the `slice` index $\mathbf{I}_n^S$ (11) so that the remaining steps can access them as inputs. After the iteration, we have all node outputs saved in $\mathbf{U}$. The final graph outputs are given as the output of the last step $\mathbf{Y} = \mathbf{Y}_N \in \mathbb{R}^{|V_N| \times C \times L}$ since we set the last node partition V_N to collect all output nodes.

5.4 Music Mixing Applications

5.4.1 Mixing Graph Construction

Figure 5.2 shows a simplified GRAFX construction of the music mixing console used in Chapter 6. The example illustrates the two representations described above. The graph is first built as a mutable object by adding input, processor, and mix nodes and connecting them with edge operations (lines 3–9); the displayed serial chain corresponds to the first upper-left row of the full mixing console in Figure 6.2a. The graph is then converted to tensors (line 10), scheduled for batched node processing with `compute_render_data` (line 11), and computed from source signals, differentiable processors, and parameter tensors with `render_grafx` (lines 12–21). The resulting output can be used directly in an audio-domain loss for gradient-based matching to a target mix.

5.4.2 Speed benchmark

We evaluated the efficiency of batched node processing with various scheduling methods. We benchmarked with a single RTX3090 GPU and the *pruned* graphs

```

1 import torch
2 from grafx import *

3 G = GRAFX()
4 in_id = G.add("in")
5 chain = ["eq", ..., "reverb"]
6 start_id, end_id = G.add_serial_chain(chain)
7 mix_id = G.add("mix")
8 G.connect(in_id, start_id)
9 G.connect(end_id, mix_id) ...

10 G_t = convert_to_tensor(G)
11 render_data = compute_render_data(G_t)
12 source = torch.rand(1, 5, 2, 2**17)
13 processors = {
14     "eq": ZerophaseFIREqualizer(), ...
15     "reverb": MidSideFilteredNoiseReverb(),
16 }
17 parameters = {
18     "eq": torch.zeros(7, 1024), ...
19     "reverb": torch.zeros(7, 768),
20 }
21 output = render_grafx(source, processors, parameters, render_data)

```

Figure 5.2: Music mixing graph construction with GRAFX.

where negligible processors are removed from the mixing consoles (Chapter 6). Figure 5.3 reports the results. The optimal schedule achieves 11.8 processor calls on average. The beam (32 candidates), greedy, and one-by-one schedules report 12.6, 16.4, and 77.6 calls, respectively. The one-by-one especially increases the processor calls linearly to the graph size. Consequently, the batched node processing improves the speed across all graph sizes, especially for large ones. While the greedy method performs slightly worse than the optimal, the beam search method closes this gap.

5.5 Conclusion

This chapter formulated differentiable audio processing graphs as a computational framework for graph-structured audio systems. The framework connects the graph definition introduced in Section 1.2.4 to tensorized graph representations, mutable graph construction, and efficient batched computation. Its

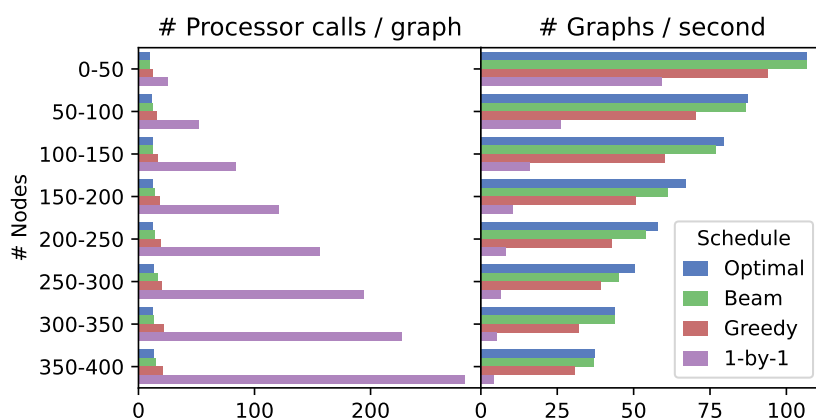


Figure 5.3: **Speed benchmark results of various scheduling methods.**

implementation in `GRAFX` consolidates these components into the forward model used in the following chapters: Chapter 6 uses it for differentiable optimization and pruning of mixing graphs, while Chapter 7 uses the same graph abstraction for blind graph estimation and later revisits differentiable graph computation as an extension. In this way, the chapter serves as the computational bridge from fixed-structure differentiable processors to the compositional graph problems considered in the remainder of the thesis.

Chapter 6

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

6.1 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 [231, 232, 233]. Alternatively, one can develop a computational method that estimates the mixing process from input source tracks and output mix. This reverse engineering strategy has several advantages. First, it is useful on its own as it allows one to analyze and understand the individual mixes. Furthermore, we can collect large-scale mixing data from various public music datasets [234, 235, 236] that provide mixes along with their source tracks. This enables more scalable, data-driven downstream applications. For example, we can conduct statistical analysis of the mixing practices [32, 33, 237]. The data can also be used to develop automatic mixing systems [15, 238, 239, 14, 228], which could include neural network components.

Reverse engineering of music mixing was initially tackled by Barchiesi and Reiss [29]. 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 [30, 93] employed differentiable signal processing [47, 49] 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.

6.1.1 Problem Formulation

Following the audio processing graph framework presented in this thesis, we describe the mixing process with a directed acyclic graph (DAG) $G = (V, E)$ whose nodes v_i are processors or auxiliary modules with a type attribute t_i , and whose edges e_{ij} are “cables” routing one node’s output signal to another node’s input. All processors considered in this chapter are single-input and single-output (SISO): each processor has one graph inlet and one graph outlet, each carrying a stereo audio stream. Thus, unlike the MIMO graph definition in Chapter 5, explicit inlet/outlet edge types are unnecessary.

We consider 7 processor types: 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 ; their differentiable implementations are detailed in Section 6.3. 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 $\hat{y}$. We denote a series of nodes with a string, e.g., $ecns$.

The reverse engineering task is defined as follows: Given the dry sources $\mathbf{S} = [s_1, \dots, s_K]$ and the ground-truth mix y , our 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, we employ an audio-domain loss function L_a . In addition, we need some regularization L_r 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

$$G^*, \mathbf{P}^* = \arg \min_{G, \mathbf{P}} [L_a(\hat{y}, y) + L_r(G, \mathbf{P})]. \quad (6.1)$$

6.1.2 Proposed Strategy

Exploring candidate graphs without restrictions might seem 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, we propose a search method based on pruning (see Figure 6.1). Following Colonel and Reiss [30, 93], we first create a “mixing console” G_c . To each input source, the mixing console applies a processor chain, which consists of all the 7 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 $\hat{y}$. We assume that the subgrouping information is known a priori. Note that this structure resembles the traditional hybrid mixing console.¹ Using differentiable processors, we implement the mixing console in `PyTorch` [183]. This allows us to optimize all console parameters $\mathbf{P}_c$ with given objectives via gradient descent. After this, we proceed to the pruning stage, where we reduce the number of processors as much as possible while 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.” We adopt a standard iterative pruning approach [94, 240]: we optimize the remaining parameters after each pruning step and repeat this process multiple times.

¹<https://steemit.com/sound/@jamesub/the-mixing-console-split-inline-and-hybrids>

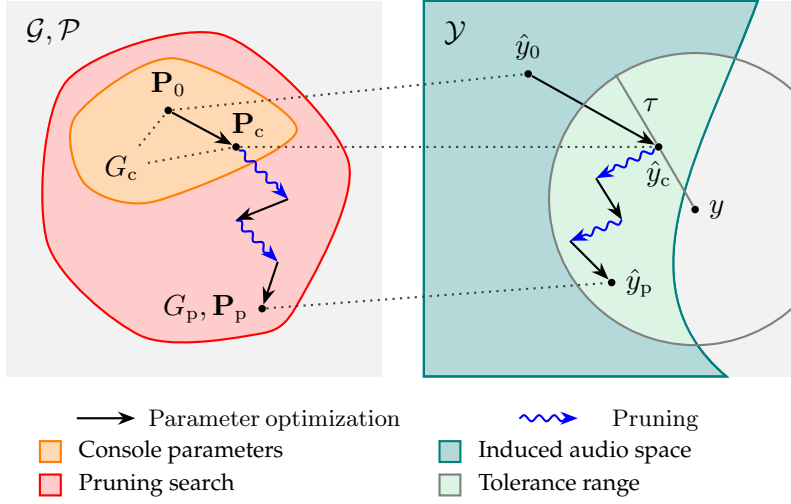


Figure 6.1: **The proposed mixing graph search method.** Gradient descent first optimizes a full mixing console, and iterative pruning then removes processors while keeping the target-matching loss within a tolerance of the optimized console loss.

Figure 6.2 gives a concrete example: the full console applies the same ordered processor chain to each source and subgroup, whereas the pruned graph removes processors that are unnecessary under the tolerance criterion.

Casting the graph search into pruning reduces the search space, which has several tradeoffs. First, the pruning only removes the processors from the pre-defined processor chains, i.e., it does not consider all other possibilities. As a result, the pruned graph G_p does not surpass the original mixing console G_c in match quality. Nevertheless, it aligns with real-world practices by selectively applying appropriate processors, promoting sparsity, and enhancing the interpretability of the estimated mixing process. Also, it significantly reduces the search cost and allows us to obtain a practical algorithm, which might be challenging with other alternatives [241, 242]. Our evaluation results show that about 60–70% 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, we employ two strategies. First, our search repeatedly computes the mix, which is a major source of bottleneck. This repeated rendering is made practical by the batched graph-rendering algorithm of Chap-

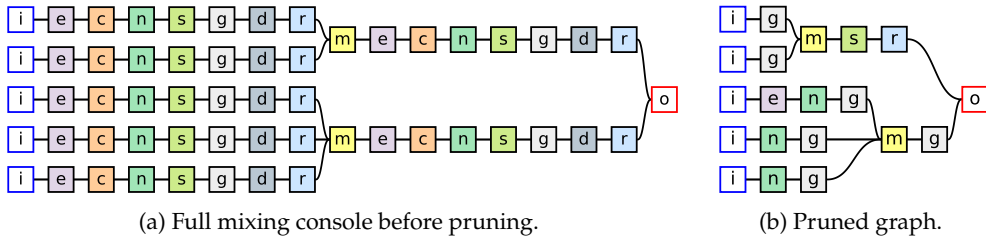


Figure 6.2: **Example full and pruned mixing graphs.** Initial letters in the nodes denote processor or auxiliary-module types: i: input, o: output, m: mix, e: equalizer, c: compressor, n: noisegate, s: stereo imager, g: gain/panning, r: reverb, and d: multitap delay.

ter 5, which computes multiple processors within a graph in parallel. This is especially effective here, as our mixing graphs are much larger and more parallelizable than those in prior works [242, 228]. Second, pruning is another source of bottleneck because it includes many trials, each involving audio processing and loss evaluations. Thus, we focus on developing efficient pruning methods. In this process, we aim to identify a subset of nodes that minimally impact match quality. To this end, we interpret each processor’s “dry/wet” parameter as an approximate importance score and leverage it to select pruning candidates. We combine this method with a brute-force node-by-node check, deriving a hybrid method that has a reasonable tradeoff between the computational cost and resulting sparsity.

This chapter is based on our earlier publication [54], with extended evaluation metrics, experiments, and subjective listening tests. The batched graph-rendering algorithm it builds on is presented separately in Chapter 5 [91].

6.1.3 Open-Source Implementation

The batched graph-rendering algorithm used here is provided by the GRAFX library (Chapter 5). The graph-pruning examples, audio samples, and implementation corresponding to Sections 6.4, 6.5, and 6.6 are available on the project page.²

²<https://sh-lee97.github.io/grafx-prune>

Table 6.1: A summary and comparison of previous works on the estimation of compositional audio signal processing.

Work	Aspect	Summary
[90]	<i>Task & domain</i>	Sound matching $[x] \rightarrow [\mathbf{P}]$. The synthesizer parameters $\mathbf{P}$ were estimated to match the reference (target) audio x .
	<i>Processors</i>	Oscillators, envelope generators, and filters that allow parameter modulation as an optional input.
	<i>Graph</i>	Any pre-defined DAG, e.g., a subtractive synthesizer with 2 oscillators, 1 amplitude envelope, and 1 lowpass filter.
	<i>Method</i>	Trained a reference encoder with separate parameter heads. Optimized with a parameter loss and intermediate-output spectral losses.
[243]	<i>Task & domain</i>	Sound matching $[x] \rightarrow [\mathbf{P}]$. An FM synthesizer matches monophonic violin, flute, and trumpet recordings with an empirically selected operator graph.
	<i>Processors</i>	Differentiable sinusoidal carrier/modulator oscillators with pre-defined frequencies. An FIR reverb is added for post-processing.
	<i>Graph</i>	DAGs with at most 6 operators. Different graphs for different target instruments.
	<i>Method</i>	Trained a convolutional neural network that estimates envelopes from the target loudness and pitch.
[242]	<i>Task & domain</i>	Sound matching $[x] \rightarrow [G, \mathbf{P}]$. Similar setup to the above [243] plus additional estimation of the operator graph G .
	<i>Processors</i>	Identical to [243], except for the frequency ratio that can be searched.
	<i>Graph</i>	A subgraph of a supergraph, which resembles a multi-layer perceptron (modulator layers followed by a carrier layer).
	<i>Method</i>	Trained a parameter estimator for the supergraph and found the appropriate subgraph G with an evolutionary search.
[244]	<i>Task & domain</i>	Reverse engineering $[s, y] \rightarrow [G, \mathbf{P}]$ of an audio effect chain from a subtractive synthesizer (commercial plugin).
	<i>Processors</i>	5 audio effects: compressor, distortion, equalizer, phaser, and reverb. Non-differentiable implementations.
	<i>Graph</i>	Chain of audio effects generated with no duplicate types (therefore 32 possible combinations) and random order.
	<i>Method</i>	Trained a next effect predictor and parameter estimator in a supervised (teacher forcing) manner.
[245]	<i>Task & domain</i>	Blind estimation $[y] \rightarrow [G]$ and reverse engineering $[s, y] \rightarrow [G]$ of guitar effect chains.
	<i>Processors</i>	13 guitar effects, including non-linear processors, modulation effects, ambiance effects, and equalizer filters.
	<i>Graph</i>	A chain of guitar effects. Maximum 5 processors and a total of 221 possible combinations.
	<i>Method</i>	Trained a convolutional neural network with synthetic data to predict the correct combination.
[246]	<i>Task & domain</i>	Blind estimation $[y] \rightarrow [G, \mathbf{P}, s]$ of audio effect chains, the original dry signal, and all the intermediate signals.
	<i>Processors</i>	4 guitar effects: distortion, delay, chorus, and reverb.
	<i>Graph</i>	A chain of audio effects.
	<i>Method</i>	Trained a hybrid waveform-spectrogram transformer that estimates each processing step autoregressively.

Continued on next page

Table 6.1: A summary and comparison of previous works on the estimation of compositional audio signal processing (continued).

Work	Aspect	Summary
[14]	<i>Task & domain</i>	Automatic mixing $[\mathbf{S}] \rightarrow [\mathbf{P}]$. Estimated parameters of fixed processing chains from source tracks ($K \leq 16$).
	<i>Processors</i>	7 differentiable processors: 4 exact implementations and a pre-trained network approximating equalizer, compressor, and reverb together.
	<i>Graph</i>	Tree structure: applied a fixed chain of the 7 processors for each track, and then summed the chain outputs together.
	<i>Method</i>	Trained a parameter estimator (convolutional neural network) with a spectrogram loss end-to-end.
[56]	<i>Task & domain</i>	Reverse engineering of music mastering $[s, y] \rightarrow [\mathbf{P}]$.
	<i>Processors</i>	A multi-band compressor, graphic equalizer, and limiter. Gradient approximated with a finite difference method.
	<i>Graph</i>	A serial chain of the processors.
	<i>Method</i>	Optimized parameters with gradient descent.
[228]	<i>Task & domain</i>	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 ($K \leq 6$).
	<i>Processors</i>	33 processors, including filters and control generators. Some are MIMO, e.g., allowing auxiliary modulations. Non-differentiable implementations.
	<i>Graph</i>	Complex DAG; splits (e.g., multi-band processing) and merges (e.g., sum and modulation). 30 processors max.
	<i>Method</i>	Trained a reference encoder and transformer decoder with direct supervision from synthetic graphs.
[93]	<i>Task & domain</i>	Reverse engineering $[\mathbf{S}, y] \rightarrow [\mathbf{P}]$ of music mixing. Estimated parameters of a fixed chain for each track.
	<i>Processors</i>	6 differentiable processors: gain, equalizer, compressor, distortion, panning, and reverb.
	<i>Graph</i>	A per-track chain of 5 processors; the reverb is applied to the mixed sum. Other DAGs can also be used.
	<i>Method</i>	Parameters were optimized with spectrogram loss end-to-end via gradient descent.
Ours	<i>Task & domain</i>	Reverse engineering $[\mathbf{S}, y] \rightarrow [G, \mathbf{P}]$ of music mixing with $K \leq 130$. Estimated processors and parameters for each track and submix.
	<i>Processors</i>	7 differentiable processors: gain/panning, stereo imager, equalizer, reverb, compressor, noisegate, and delay.
	<i>Graph</i>	A tree of processing chains with a subgrouping structure. Processors can be omitted but should follow the fixed order.
	<i>Method</i>	Jointly optimized dry/wet masks and processor parameters with a spectrogram loss, followed by hard pruning stages.

6.2 Related Works

6.2.1 Composition of Audio Processors

While our work focuses on music mixing, it is valuable to review representative works that incorporate the compositional aspects of musical signal processing in a broader scope, such as sound matching [90, 243, 242], effect chain estimation [244, 245, 246, 56, 228], and music mixing estimation [14, 228, 93]. Table 6.1 highlights the key differences among these approaches. These works vary in their tasks, domains, processors, graph structures, and estimation methods. We denote each task 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 [228, 244, 245, 93]. Some fixed the graph and estimated only the parameters [14, 90, 93, 47, 56]. Others tried to predict the graph [245] or both [228, 244, 242, 243]. Also, with several exceptions [228, 242], most works consider simple compositions, e.g., serial chains, that do not require graph terminology. Yet, we treat those cases as simple graphs. This chapter presents a graph reverse engineering method based on end-to-end parameter optimization with gradient descent.

6.2.2 Graph Search

Various research domains search for graphs with desired properties, including neural network pruning [94, 240] (conceptually identical to our approach), neural architecture search (NAS) [247], and reinforcement learning-based graph generation [248, 249]. One particularly relevant work is differentiable architecture search (DARTS) [241], 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 one of N processors). DARTS is clearly more flexible and general. However, it also greatly increases the computational cost, as we must compute all N processors to compute their weight sum for every node.

We note that another possible strategy for graph estimation is simply to train a neural network as a graph estimator in a supervised manner [228]. However, in our

case, the ground-truth graphs for training are not available. 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 [228] that 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.

6.3 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:

$$\bar{y}_i[n] = f_i(u_i[n], \bar{p}_i), \quad (6.2a)$$

$$y_i[n] = w_i \bar{y}_i[n] + (1 - w_i) u_i[n]. \quad (6.2b)$$

That is, after the main processing f_i with parameters $\bar{p}_i$, we mix the input and the result $\bar{y}_i$ with a “dry/wet” weight $w_i \in [0, 1]$. Separating the dry/wet mix from the other processing is helpful for the latter discussion on pruning. We denote the full parameter as $p_i = (\bar{p}_i, w_i)$ and use it when the distinction of the two is unnecessary. In the graph search, this dry/wet weight also serves as an approximate importance score for pruning candidates. Note that, as the processor v_i is used inside the graph $G = (V, E)$, its input will be given as a sum of other nodes’ outputs.

$$u_i[n] = \sum_{j \in \mathcal{N}(i)} y_j[n] \quad (6.3)$$

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

Gain/panning — We multiply a learnable gain for each input channel. The parameter $\bar{p}_g \in \mathbb{R}^2$ is the concatenation of the log gains, and the output is given as

$$\bar{y}[n] = \exp(\bar{p}_g) \cdot u[n]. \quad (6.4)$$

Stereo Imager — We adjust the stereo width 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

$$y_m[n] = u_l[n] + u_r[n], \quad (6.5a)$$

$$y_s[n] = \exp(\bar{p}_s) \cdot (u_l[n] - u_r[n]). \quad (6.5b)$$

Then, these are converted back to the stereo output with

$$\bar{y}_l[n] = (y_m[n] + y_s[n])/2, \quad (6.6a)$$

$$\bar{y}_r[n] = (y_m[n] - y_s[n])/2. \quad (6.6b)$$

Equalizer — We use a zero-phase FIR filter with log-magnitudes used as a parameter $\bar{p}_e$. We can obtain the time-domain response with exponentiation followed by inverse FFT (IFFT) and Hann-windowing $v^{\text{Hann}}[n]$. As a result, the length- N FIR is

$$h_e[n] = v^{\text{Hann}}[n] \cdot \frac{1}{N} \sum_{k=0}^{N-1} \exp \bar{p}_e[k] \cdot w_N^{kn} \quad (6.7)$$

where $-(N-1)/2 \leq n \leq (N-1)/2$ and $w_N = \exp(j \cdot 2\pi/N)$. We convolve the same FIR to the left and right input channels to compute the output.

$$\bar{y}_x[n] = u_x[n] * h_e[n] \quad (x \in \{l, r\}). \quad (6.8)$$

The FIR length is set to $N = 2047$, i.e., the parameter $\bar{p}_e$ is a 1024-dimensional vector. For all causal convolutions, `FlashFFTConv` [250] is used to speed up the processing and save memory.

Reverb — We employ a stereo FIR filter where each FIR is a variant of the time-varying filtered noise [47]. We start from uniform noise, $u_m[n]$ and $u_s[n] \sim \mathcal{U}(-1, 1)$, one for the mid channel and the other one for the side channel, respectively (both are 2 seconds long). Then, each noise is converted to its STFT $U_x[k, m]$ and multiplied with an independently parameterized magnitude mask $M_x[k, m]$.

$$H_x[k, m] = U_x[k, m] \odot M_x[k, m] \quad (x \in \{m, s\}). \quad (6.9)$$

Here, k and m denote frequency and time frame index, respectively. We create each mask M_x with two parameters: a log-magnitude vector $H_x^0[k]$ that corresponds to the initial coloration and another log-magnitude vector $H_x^\Delta[k]$ that models the frequency-dependent decay of the reverb.

$$M_x[k, m] = \exp(H_x^0[k] + (m - 1)H_x^\Delta[k]). \quad (6.10)$$

The masked STFTs, H_m and H_s , are converted back to the time-domain responses with the inverse STFT and further converted to the stereo FIR $h_r[n]$. Channel-wise convolution between the input $u[n]$ and the FIR yields the desired output $\bar{y}[n]$. We set the FFT and hop lengths to 384 and 192, respectively. Each reverb processor has 768 parameters (2 channels, each with 2 log-magnitude vectors of size 192).

Compressor — We follow the canonical implementation of the digital feed-forward compressor [10]. We first calculate a mid-channel signal $u_m[n]$ by summing the left and right channels. Then, we square the signal and smooth it with the following recursive filter called “ballistics.”

$$g_u[n] = \alpha[n]g_u[n - 1] + (1 - \alpha[n])u_m^2[n]. \quad (6.11)$$

In standard compressors, values of the smoothing coefficient $\alpha[n]$ are set differently for an “attack” and “release” phase (where $g_u[n]$ increases and decreases, respectively). However, this part bottlenecks the computation speed in GPU. To mitigate this, following [57], we 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

$$h^{\text{env}}[n] = (1 - \alpha)\alpha^n. \quad (6.12)$$

Then, we can compute the energy envelope $g_u[n]$ by a simple convolution of the energy signal $u_m^2[n]$ with the above IIR $h^{\text{env}}[n]$ truncated to length N . Next, we calculate the compressed energy envelope $G_{\bar{y}}[n]$ from the log of the input envelope $G_u[n] = \log g_u[n]$. We use a quadratic knee that interpolates the compression and

the bypass region. For a given threshold T and half of the knee width W ,

$$G_{\bar{y}}[n] = \begin{cases} G_{\bar{y}}^{\text{above}}[n] & G_u[n] \geq T + W, \\ G_{\bar{y}}^{\text{mid}}[n] & T - W \leq G_u[n] < T + W, \\ G_{\bar{y}}^{\text{below}}[n] & G_u[n] < T - W \end{cases} \quad (6.13)$$

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

$$G_{\bar{y}}^{\text{above}}[n] = T + \frac{G_u[n] - T}{R}, \quad (6.14a)$$

$$G_{\bar{y}}^{\text{mid}}[n] = G_u[n] + \left(\frac{1}{R} - 1\right) \frac{(G_u[n] - T + W)^2}{4W}, \quad (6.14b)$$

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

$$\bar{y}_x[n] = \exp(G_{\bar{y}}[n] - G_u[n]) \cdot u_x[n] \quad (x \in \{l, r\}). \quad (6.15)$$

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

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

$$G_{\bar{y}}^{\text{mid}}[n] = G_u[n] + (1 - R) \frac{(G_u[n] - T - W)^2}{4W}, \quad (6.16a)$$

$$G_{\bar{y}}^{\text{below}}[n] = T + R(G_u[n] - T). \quad (6.16b)$$

Multitap Delay — We observed some mixes in our dataset use heavy delay (or “echo”) effects; hence, we introduce an FIR filter that can mimic this. The left and right channels use independent FIRs, and they are convolved with the input channel-wise. For simplicity, we omit this in the following text and equation. Under this setup, the multitap delay's FIR is given as

$$h_d[n] = \sum_{m=1}^M c_m[n] * \delta[n - d_m] \quad (6.17)$$

where $\delta[n]$ is a unit impulse signal and c_m is a 39-tap FIR implemented the same as

the zero-phase equalizer. We use $M = 20$ delays, with each delay d_m constrained to its own 100ms range. We want to optimize each discrete delay length $d_m \in \mathbb{N}$ with gradient descent. To achieve this, we utilize the property that each delayed impulse corresponds to a complex sinusoid in the frequency domain. Following [58], the complex angular frequency $z_m \in \mathbb{C}$ of such a sinusoid can be optimized using gradient descent if we allow it to be in the unit disk, i.e., $|z_m| \leq 1$. We leverage this finding; from the angular frequency z_m , we compute a damped sinusoid. Then, its inverse FFT can be used as a surrogate of the delayed unit impulse.

$$\delta[n - d_m] \approx \frac{1}{N} \sum_{k=0}^{N-1} z_m^k w_N^{kn}. \quad (6.18)$$

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, we use it only for calculating the gradients with straight-through estimation [39]. Moreover, we apply normalization to the gradient and add a regularization that guides the parameters to increase their radius. We empirically observed that this improves the performance. To summarize, each gradient is updated as follows:

$$\frac{\partial L}{\partial z_m^*} \leftarrow \text{sgn} \left(\sum_n \frac{\partial L}{\partial h_d[n]} \frac{\partial \tilde{h}_d[n]}{\partial z_m^*} \right) + \gamma(|z_m| - 1) \text{sgn}(z_m^*) \quad (6.19)$$

where $\tilde{h}_d[n]$ is the surrogate FIR obtained with the continuous delay, $\text{sgn}(z) = z/|z|$ and γ is a regularization strength set to 0.01. This multitap delay has a parameter vector $\bar{p}_d$ 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).

6.4 Music Mixing Consoles

Before exploring the pruning, we first evaluate the match quality of the mixing consoles, which will serve as a performance upper bound. In addition to evaluating the full console, we analyze the contribution of each processor type to the overall performance. To achieve this, we begin with a base graph that simply sums all

the inputs. Then, we sequentially add each processor type to every processor chain (refer to Tables 6.2 and 6.3). Each preliminary graph is then optimized and evaluated for every song in the validation set. This experiment is similar to the evaluation in [93], albeit the structure of mixing consoles and the processors differ.

6.4.1 Optimization

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

$$L_a = \alpha_{lr}L_{lr} + \alpha_m L_m + \alpha_s L_s, \quad (6.20)$$

where each term L_x ($x \in \{lr, m, s\}$, lr: left/right, m: mid, s: side) is a multi-resolution STFT (MRSTFT) loss [251]

$$L_x = \sum_{i=1}^I \left[\frac{\|\log Y_x^{(i)} - \log \hat{Y}_x^{(i)}\|_1}{N} + \frac{\|Y_x^{(i)} - \hat{Y}_x^{(i)}\|_F}{\|Y_x^{(i)}\|_F} \right]. \quad (6.21)$$

Here, $Y_x^{(i)}$ and $\hat{Y}_x^{(i)}$ 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 [252]. The loss weights are $\alpha_{lr} = 0.5$, $\alpha_m = 0.25$, and $\alpha_s = 0.25$. We use `auraloss` [253] for the implementation. Additionally, we include the following regularization term to encourage gain staging, a common practice to ensure the total energy of the input and output remains approximately equal:

$$L_g = \sum_{v_i \in V_g} |\log \|(\bar{y}_i)_m\|_2 - \log \|(u_i)_m\|_2| \quad (6.22)$$

where $(\cdot)_m$ 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 us to remove redundant gains introduced by these LTI processors and constrain the parameters to be within a reasonable range. Therefore,

the full optimization loss is

$$L(\bar{\mathbf{P}}, \mathbf{w}) = L_a(\bar{\mathbf{P}}, \mathbf{w}) + \alpha_g L_g(\bar{\mathbf{P}}) \quad (6.23)$$

where the weight of the gain staging regularization is set to $\alpha_g = 10^{-3}$ (a different notation was used from Equation (6.1) 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 $(0, 1)$. We optimize each console for 12k steps using AdamW optimizer [185] with a 0.01 learning rate. Each step randomly samples a 3.8s 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.

6.4.2 Evaluation Metrics

We follow [254] and compare the optimized mix and the ground truth with the following metrics. First, we measure Fréchet Audio Distance (FAD) [255], which calculates the distance between the feature distributions obtained with a pre-trained audio encoder. We also report the scale-invariant signal-to-distortion ratio (SI-SDR), widely used in source separation literature, which involves sample-wise comparison [256]. Furthermore, we measure the mean square errors (MSEs) of simple analysis features used in music information retrieval (MIR). These are referred to as “MIR feature distances.” We split the mix and its match into disjoint 8-second segments, $y^{(1)}, \dots, y^{(M)}$ and $\hat{y}^{(1)}, \dots, \hat{y}^{(M)}$. Then, we calculate the MSEs of features extracted from those segments. We further apply logarithm to compress the values, as we empirically found it to correlate better to the subjective similarity. We consider the same set of features in [254]: root mean square (RMS), crest factor (CF), stereo width (SW), stereo imbalance (SI), and bark spectrum (BS) [257].

$$d_x = \log_{10} \left[\frac{1}{MK_x} \sum_{m=1}^M \left\| F_x(y^{(m)}) - F_x(\hat{y}^{(m)}) \right\|_2^2 \right]. \quad (6.24)$$

Table 6.2: Matching performance of mixing console configurations. Dataset: MedleyDB.

	MRSTFT ↓				FAD ↓ SI-SDR ↑		MIR Feature Distance ↓					
	L_a	L_{lr}	L_m	L_s			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	.662	-1.99	-2.22	2.46	
+ Gain/panning	.550	.583	.485	.549	.719	-7.20	-5.01	.484	-3.52	-2.38	-1.56	
+ Stereo imager	.541	.564	.483	.553	.733	-6.97	-5.02	.488	-3.54	-2.48	-1.54	
+ Equalizer	.450	.453	.390	.504	.573	-5.25	-5.45	.415	-3.71	-2.65	-1.62	
+ Reverb	.368	.361	.360	.390	.405	-3.94	-5.54	.356	-4.17	-3.55	-1.66	
+ Compressor	.315	.304	.297	.356	.312	-2.95	-6.15	.173	-4.33	-3.61	-1.52	
+ Noisegate	.302	.288	.281	.353	.264	-2.60	-6.12	.065	-4.22	-3.55	-1.51	
+ Multitap delay (full)	.296	.288	.284	.324	.222	-2.66	-6.17	.093	-4.34	-3.78	-1.46	

Table 6.3: **Matching performance of mixing console configurations. Dataset: MixingSecrets.**

	MRSTFT ↓				FAD ↓ SI-SDR ↑		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	.401	
+ Gain/panning	.876	.856	.819	.973	1.49	-10.0	-4.63	1.21	-2.33	-1.82	-.842	
+ Stereo imager	.847	.834	.791	.928	1.46	-10.3	-4.64	1.20	-2.27	-1.94	-.855	
+ Equalizer	.699	.698	.622	.780	.933	-9.62	-5.00	1.14	-2.79	-2.20	-1.23	
+ Reverb	.614	.601	.579	.674	.642	-6.89	-5.07	1.04	-2.98	-2.74	-1.39	
+ Compressor	.558	.542	.512	.637	.526	-6.10	-5.13	.987	-2.81	-2.70	-1.20	
+ Noisegate	.548	.532	.502	.625	.523	-7.04	-4.98	.927	-2.61	-2.60	-1.17	
+ Multitap delay (full)	.545	.529	.502	.618	.506	-6.85	-4.98	.985	-2.84	-2.66	-1.13	

where $x \in \{\text{RMS}, \text{CF}, \text{SW}, \text{SI}, \text{BS}\}$, $\|\cdot\|_2$ 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.

6.4.3 Data

For each song, we need dry sources $\mathbf{S}$ and a mixture y to run the optimization. Also, we need additional subgroup information to construct the initial mixing console. Thus, we utilize the `MedleyDB` dataset [234, 235], as it provides all the necessary components. The `MixingSecrets` library [236] only provides the audio; therefore, we manually create the subgroup information. For evaluation purposes, we use a random 24-song subset for each dataset, and the results are reported separately. Finally, to collect the graph data, we use the full data and also include our private `Internal` dataset crowdsourced from multiple engineers, which comprises Western music recordings. A total of 1239 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 30kHz sampling rate.

6.4.4 Results

Table 6.2 and 6.3 report the results for the `MedleyDB` and `MixingSecrets` evaluation subsets, respectively. We 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, as most source tracks are mono (hence, so is the naive sum) while the target mixes have wide stereo images. With the gain/pannings and stereo imagers, we achieve “rough mixes” with a loss of 0.541. Then, we fill in the missing details 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 dataset 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 (-2.66 and -6.85 dB on the two subsets). We suspect two causes: 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. While 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, not the overall spectral balance. Similarly, as 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, we note that the average of all MIR feature distances on the two subsets consistently decreases, from -1.22 to -2.63 .

We can conduct further qualitative analysis by comparing representative log-magnitude spectrograms of the targets, selected mixing console matches, and their errors (Figures 6.10 and 6.11). Again, we observe that adding each processor type improves the match. Furthermore, each song has different types that are more crucial; for the song *NewSkin* (Figure 6.10), there is significant error reduction when reverbs are introduced. However, this is not the case for *RockSteady* (Figure 6.11). Instead, for the latter, different from the average trend, the multitap delays improve the match more than the reverbs. This is not so surprising as the original mix introduces heavy delay effects.

6.5 Music Mixing Graph Search

The previous full mixing console G_c serves as an upper bound for matching performance. Now, our goal is to identify a sparser graph that maintains a similar match quality. This is achieved by pruning the console as much as possible while

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 $\mathbf{P} = (\bar{\mathbf{P}}, \mathbf{w})$

```
1:  $\bar{\mathbf{P}}, \mathbf{w} \leftarrow \text{Initialize}(G_c)$ 
2:  $\bar{\mathbf{P}}, \mathbf{w} \leftarrow \text{Train}(G_c, \bar{\mathbf{P}}, \mathbf{w}, \mathbf{S}, y)$ 
3:  $L_a^{\min} \leftarrow \text{Evaluate}(G_c, \bar{\mathbf{P}}, \mathbf{w}, \mathbf{S}, y)$ 
4:  $G_p \leftarrow G_c$ 
5: for  $n \leftarrow 1$  to  $N_{\text{iter}}$  do
6:    $V_{\text{pool}}, \mathbf{m} \leftarrow \text{GetAllProcessors}(V), 1$ 
7:   while  $\text{TryPrune}(V_{\text{pool}}, \mathbf{w}, \mathbf{m})$  do
8:      $V_{\text{cand}}, \mathbf{m}_{\text{cand}} \leftarrow \text{SampleCandidate}(V_{\text{pool}}, \mathbf{w})$ 
9:      $L_a \leftarrow \text{Evaluate}(G_p, \bar{\mathbf{P}}, \mathbf{w} \odot \mathbf{m} \odot \mathbf{m}_{\text{cand}}, \mathbf{S}, y)$ 
10:    if  $L_a < L_a^{\min} + \tau$  then
11:       $L_a^{\min} \leftarrow \min(L_a^{\min}, L_a)$ 
12:       $\mathbf{m} \leftarrow \mathbf{m} \odot \mathbf{m}_{\text{cand}}$ 
13:    end if
14:     $V_{\text{pool}} = \text{UpdatePool}(V_{\text{pool}}, V_{\text{cand}})$ 
15:  end while
16:   $G_p, \bar{\mathbf{P}}, \mathbf{w} \leftarrow \text{Prune}(G_p, \bar{\mathbf{P}}, \mathbf{w}, \mathbf{m})$ 
17:   $\bar{\mathbf{P}}, \mathbf{w} \leftarrow \text{Train}(G_p, \bar{\mathbf{P}}, \mathbf{w}, \mathbf{S}, y)$ 
18: end for
19: return  $G_p, \bar{\mathbf{P}}, \mathbf{w}$ 
```

ensuring that the loss increase remains within a pre-defined tolerance threshold τ . This requirement is illustrated as a circle in Figure 6.1. We write this objective as

$$\text{minimize } |V_p| \quad \text{s.t.} \quad \min L_a(G_p) \leq \min L_a(G_c) + \tau \quad (6.25)$$

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

6.5.1 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, we assume that the processors independently contribute to the match and adopt a greedy pruning strategy. We incrementally remove processors whenever the tolerance condition is met, fine-tuning between rounds [94, 240]. Ideally, intermediate pruned graphs should be fine-tuned before assessing the tolerance condition. However, for a reasonable computational complexity, we omit this step, paying the cost of potentially missing some removable processors.

Our method is described in Algorithm 2, with specific steps referenced in the following parentheses. We 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, we alternate between pruning and optimizing the remaining parameters and weights, i.e., fine-tuning (5-18). Each pruning stage involves multiple trials, where 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 $\mathbf{m}_{\text{cand}}$ (9). After the trials, we perform the actual pruning of nodes so that we can discard the unnecessary processing and accelerate further optimization (16). Note that pruning occasionally improves the match quality. In such cases, the pruning threshold is updated accordingly (11).

6.5.2 Candidate Sampling

Implementation-wise, we have 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). We explore the following 3 approaches.

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

Second, to improve efficiency from the above method, we need an informed

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 $\mathbf{P} = (\bar{\mathbf{P}}, \mathbf{w})$

```
1:  $\bar{\mathbf{P}}, \mathbf{w} \leftarrow \text{Initialize}(G_c)$ 
2:  $\bar{\mathbf{P}}, \mathbf{w} \leftarrow \text{Train}(G_c, \bar{\mathbf{P}}, \mathbf{w}, \mathbf{S}, y)$ 
3:  $L_a^{\min} \leftarrow \text{Evaluate}(G_c, \bar{\mathbf{P}}, \mathbf{w}, \mathbf{S}, y)$ 
4:  $G_p \leftarrow G_c$ 
5: for  $n \leftarrow 1$  to  $N_{\text{iter}}$  do
6:    $T_{\text{pool}}, \mathbf{m} \leftarrow \text{GetProcessorTypeSet}(V), 1$ 
7:   for  $t$  in  $T_{\text{pool}}$  do
8:      $V_t, \mathbf{w}_t \leftarrow \text{Filter}(V, t), \text{Filter}(\mathbf{w}, t)$ 
9:      $N_t, r_t \leftarrow |V_t|, 0.1$ 
10:  end for
11:  while  $T_{\text{pool}} \neq \emptyset$  do
12:     $t \leftarrow \text{SampleType}(T_{\text{pool}})$ 
13:     $\bar{V}_t, \bar{\mathbf{m}} \leftarrow \text{GetLeastWeightNodes}(V_t, \mathbf{w}_t, \lfloor N_t r_t \rfloor)$ 
14:     $L_a \leftarrow \text{Evaluate}(G_p, \bar{\mathbf{P}}, \mathbf{w} \odot \mathbf{m} \odot \bar{\mathbf{m}}, \mathbf{S}, y)$ 
15:    if  $L_a < L_a^{\min} + \tau$  then
16:       $L_a^{\min} \leftarrow \min(L_a^{\min}, L_a)$ 
17:       $\mathbf{m} \leftarrow \mathbf{m} \odot \bar{\mathbf{m}}$ 
18:       $V_t \leftarrow V_t \setminus \bar{V}_t$ 
19:    else
20:      if  $\lfloor N_t r_t \rfloor > 1$  then
21:         $r_t \leftarrow r_t/2$ 
22:      else
23:         $T_{\text{pool}} \leftarrow T_{\text{pool}} \setminus \{t\}$ 
24:      end if
25:    end if
26:  end while
27:   $G_p, \bar{\mathbf{P}}, \mathbf{w} \leftarrow \text{Prune}(G_p, \bar{\mathbf{P}}, \mathbf{w}, \mathbf{m})$ 
28:   $\bar{\mathbf{P}}, \mathbf{w} \leftarrow \text{Train}(G_p, \bar{\mathbf{P}}, \mathbf{w}, \mathbf{S}, y)$ 
29: end for
30: return  $G_p, \bar{\mathbf{P}}, \mathbf{w}$ 
```

Table 6.4: Pruning results with different strategies and tolerance τ . Dataset: MedleyDB.

	τ	$L_a \downarrow$	Pruning Ratio $\uparrow$								FAD $\downarrow$		SI-SDR $\uparrow$		MIR Feature Distance $\downarrow$				
			μ	g	s	e	r	c	n	d	d _{RMS}	d _{CR}	d _{SW}	d _{SI}	d _{BS}				
Mixing console	—	.296	—	—	—	—	—	—	—	—	.222	—2.66	—6.17	.093	—4.34	—3.78	—1.46		
Brute-force	.01	.305	.626	.343	.811	.540	.772	.669	.711	.532	.262	—3.41	—5.94	.147	—4.14	—3.54	—1.51		
Dry/wet	.01	.302	.561	.315	.791	.416	.736	.568	.669	.433	.229	—1.87	—6.02	.066	—4.23	—3.62	—1.55		
Hybrid	.001	.295	.442	.216	.728	.259	.612	.503	.540	.237	.227	—2.18	—6.11	.108	—4.36	—3.77	—1.51		
	.01	.301	.606	.322	.805	.483	.741	.666	.708	.519	.247	—3.03	—5.97	.096	—4.15	—3.58	—1.52		
	.1	.374	.829	.586	.926	.828	.918	.804	.840	.898	.406	—4.38	—5.47	.335	—3.78	—3.07	—1.57		

Table 6.5: Pruning results with different strategies and tolerance τ . Dataset: MixingSecrets.

	τ	$L_a \downarrow$	Pruning Ratio $\uparrow$								FAD $\downarrow$ SI-SDR $\uparrow$		MIR Feature Distance $\downarrow$				
			μ	g	s	e	r	c	n	d	d_{RMS}	d_{CR}	d_{SW}	d_{SI}	d_{BS}		
Mixing console	—	.545	—	—	—	—	—	—	—	—	.506	—6.85	—4.98	.985	—2.84	—2.66	—1.13
Brute-force	.01	.566	.654	.498	.823	.429	.710	.605	.766	.748	.548	—7.13	—5.06	.991	—2.83	—2.54	—1.19
Dry/wet	.01	.561	.566	.472	.818	.222	.623	.539	.744	.546	.513	—6.79	—4.96	1.02	—2.68	—2.48	—1.13
Hybrid	.001	.550	.433	.345	.765	.134	.421	.429	.552	.383	.523	—6.66	—4.94	1.00	—2.68	—2.58	—1.11
	.01	.563	.622	.490	.848	.344	.637	.599	.789	.646	.519	—6.35	—5.05	.995	—2.78	—2.61	—1.22
	.1	.647	.843	.703	.914	.751	.864	.831	.927	.909	.805	—7.98	—4.82	1.03	—2.78	—2.33	—1.19

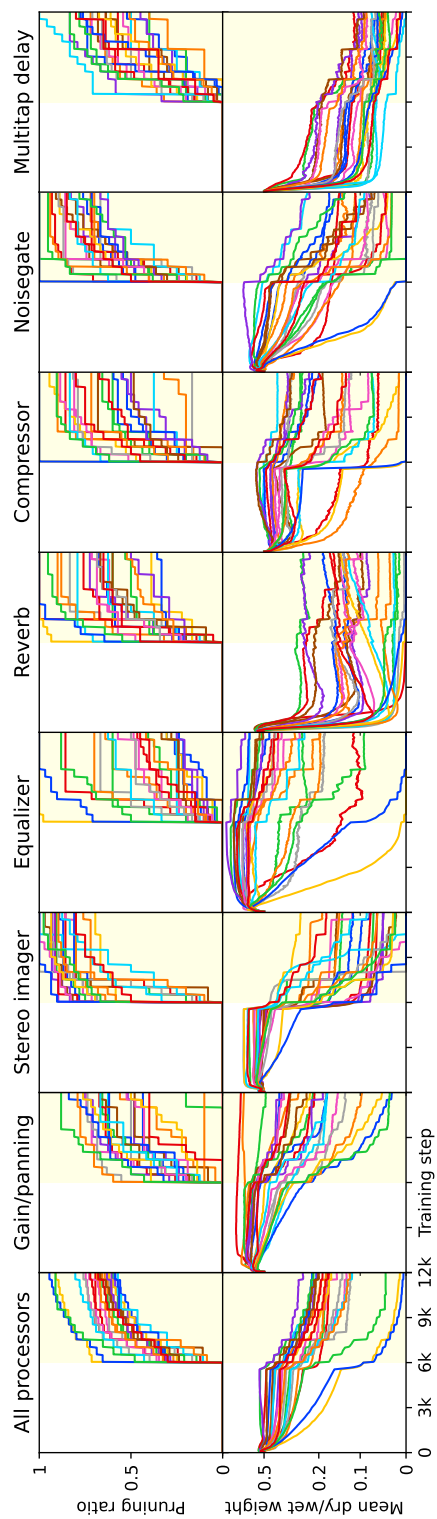


Figure 6.3: **Process of iterative pruning (hybrid, $\tau = 0.01$).** 24 random-sampled songs are shown; each color represents an individual song. The upper and lower rows show the pruning ratios and mean dry/wet weights. The yellow-shaded regions show the pruning phase.

guess of each node’s importance. Recall that each processor mixes its input and output with a dry/wet weight, $y_i = w_i \bar{y}_i + (1 - w_i)u_i$ (Section 6.3). Intuitively, we can use each dry/wet weight w_i for approximate importance as $w_i = 0$ 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, we create a set of remaining types T_{pool} and count the processors of each type N_t (6-10). Then, for each trial, we 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, we perform an iterative search to find the maximum proportion for each type. When the trial succeeds, we prune those candidates and continue the search for that type t (18). If the trial fails, we perform the following: if the number of candidates is larger than 1, we halve the ratio for the next search of the type t . Otherwise, we terminate the search of the type t 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. We mitigate this by combining the above two methods, running the brute-force method for every 4th iteration and the dry/wet method for the remaining. We call this a hybrid method, which is, along with the tolerance of $\tau = 0.01$, our default choice throughout the experiments.

6.5.3 Optimization

We 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.

$$L(\bar{\mathbf{P}}, \mathbf{w}) = L_a(\bar{\mathbf{P}}, \mathbf{w}) + \alpha_g L_g(\bar{\mathbf{P}}) + \alpha_p L_p(\mathbf{w}). \quad (6.26)$$

We begin by training the console for 6k steps. Following this, we perform $N_{\text{iter}} = 12$ rounds of pruning, with each round including 0.5k 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 in loss, this approach is justified due to strict resource constraints. During the initial 4k steps of the pruning phase, we increase the sparsity coefficient α_p linearly from 0 to 10^{-4} .

6.6 Results

6.6.1 Sparsity and Quality Degradation

Same as the mixing console experiments, we report results for `MedleyDB` and `MixingSecrets` datasets separately (Table 6.4 and 6.5, 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 to 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., 0.483 and 0.805 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 0.5dB for SI-SDR).

6.6.2 Choice of Tolerance

Even with a very low tolerance of $\tau = 0.001$, we can nearly halve the number of processors (average pruning ratio of $\mu = 0.442$ and 0.433 for the respective subsets). 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 while achieving reasonable sparsity. This can be verified qualitatively with the pruning-tolerance rows in the representative

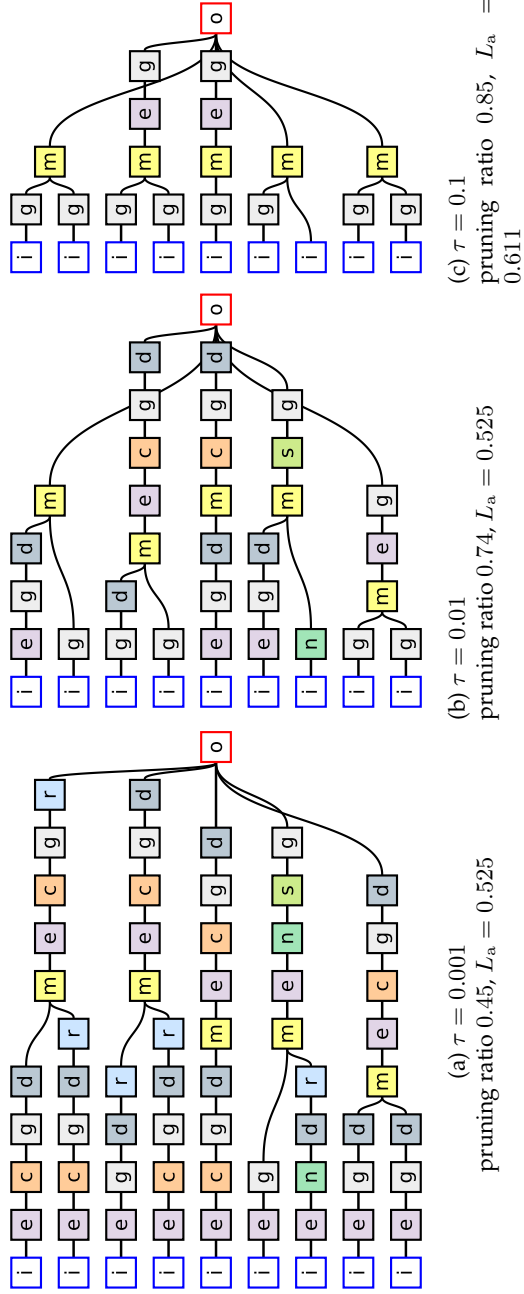


Figure 6.4: Pruning results (hybrid) with various tolerances. Song: TablaBreakbeatScience_RockSteady.

spectrogram examples (Figures 6.10 and 6.11) and the latter subjective listening test results. 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 Figure 6.4, 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).

6.6.3 Pruning Process

Figure 6.3 shows how the pruning progresses. During the first 6k steps of initial console optimization, the dry/wet weights adapt freely to optimize the audio loss L_a since 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.

6.6.4 Sampling Method Comparison

We compare the three proposed candidate sampling variants under the same tolerance value $\tau = 0.01$. 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` datasets, 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, while results with MIR feature distances are more mixed.

To investigate the differences in sparsity, we 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,

$$\Delta_i = L_a(G \setminus \{v_i\}) - L_a(G). \quad (6.27)$$

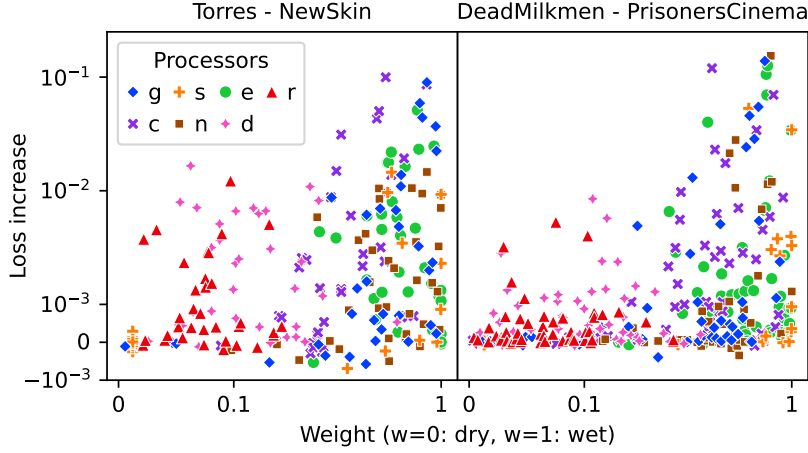


Figure 6.5: Each node’s weight and loss increase when pruned.

Figure 6.5 shows scatterplots for 2 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, we cannot completely remove the weakness of the dry/wet method, leading us to the hybrid strategy as a compromise.

The three methods differ not only in sparsity but also in the tradeoffs between sparsity and match performance. We 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. Figure 6.6 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 minutes for each song, respectively, using 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 tradeoff is perceptually significant, raising the question of whether the finer search is worth the cost.

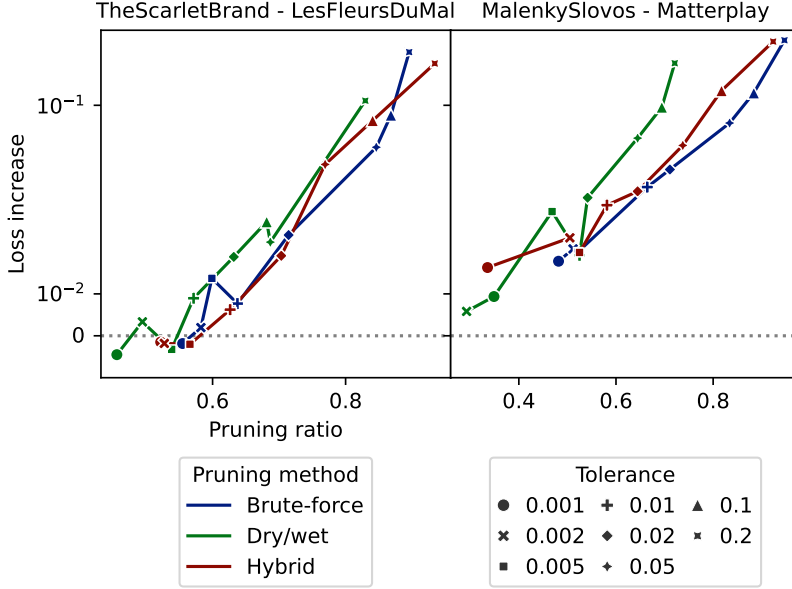


Figure 6.6: Loss increases from the mixing console and pruning ratios for different pruning methods and tolerances.

6.6.5 Subjective Listening Test

To further investigate the performance of our methods, we conducted a Multiple Stimuli with Hidden Reference and Anchor (MUSHRA) test [129]. We asked participants to compare the original mix (reference) and the others with scores ranging from 0 to 100. A total of 8 stimuli were evaluated, including (i) the hidden reference, (ii) naive summation as a low anchor, and the mixing consoles with (iii) 2 and (iv) 4 processor types (as in Table 6.2) as mid anchors. The remaining four are the (v) full mixing console and its pruned versions with the hybrid method and the tolerance τ of (vi) 0.1, (vii) 0.01, and (viii) 0.001. 18 subjects participated in the test: 9 were professional mixing engineers and 6 were amateurs (4.5 years of experience on average). The remaining 3 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 2 training and 12 main sessions. Each session was a random 8-second excerpt from the validation set. We used WebMUSHRA [130] to conduct the test remotely.

Figure 6.7 summarizes the results. The hidden reference scored 92.5 ± 1.3 (95%

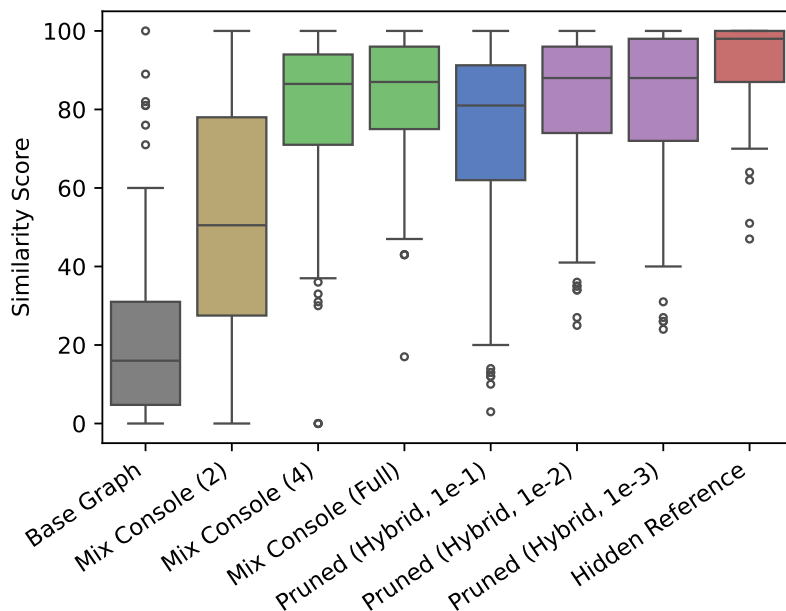


Figure 6.7: **Subjective listening test results.**

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 2 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. Figure 6.8 shows scatter plots between

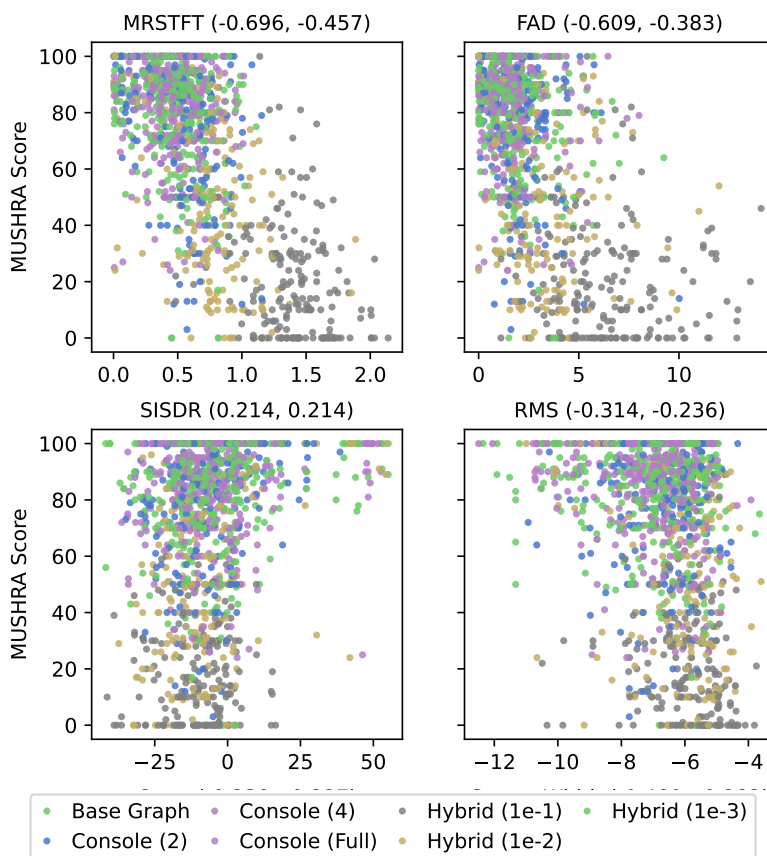


Figure 6.8: **Scatter plots of MUSHRA scores and objective metrics.**

the MUSHRA score and the loss/metrics, continued in Figure 6.9. For each pair, we also calculated 2 Pearson correlation coefficients (PCCs): 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 results are, in many cases, too different from the target and not of interest. These PCC values are also reported in Figure 6.8 (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 pre-trained for that purpose. The MIR feature distances have slightly weaker correlations, which is expected as each feature reflects certain specific aspects of the mix. The SI-SDR has the

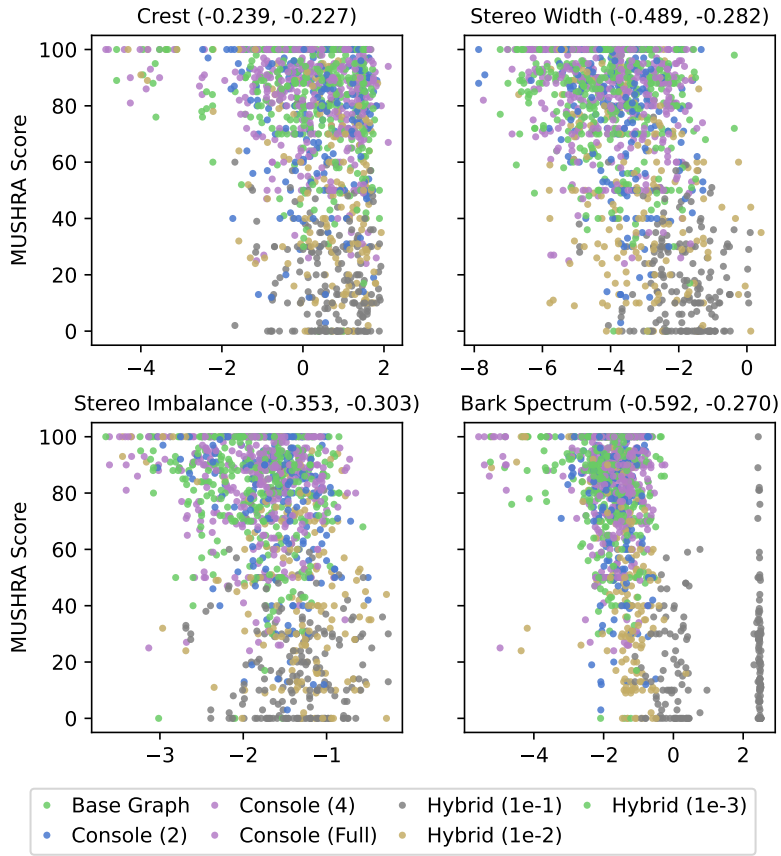


Figure 6.9: **Scatter plots of MUSHRA scores and objective metrics (cont.).**

weakest correlation, possibly because of the aforementioned misalignment and sample-level match not being perceptually important.

After the test, we asked the subjects what kind of differences they could find between the reference and the other stimuli, especially the full consoles and the pruning with $\tau \leq 0.01$. 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, we also conducted an informal AB test to compare the three candidate selection methods with the same tolerance

$\tau = 0.01$. However, no statistically 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 $\tau = 0.001$.

6.6.6 Characteristics of Iterative Pruning

Here, we provide example cases to illustrate the behavior of the pruning method. First, we can qualitatively check that the pruning indeed removes the unnecessary processors. Figures 6.10 and 6.11 collect representative spectrogram examples for two songs, including selected mixing console configurations and selected pruning tolerances. Recall that adding reverbs to the song `RockSteady` did not significantly improve performance (Figure 6.11). It is reasonable to anticipate that those reverbs will be pruned with a moderate tolerance τ accordingly. Figure 6.4 shows that this is the case; only 5 reverbs remain among 14 when $\tau = 0.001$ and zero 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 Figure 6.4b, the bottom 2 inputs are processed with 3 gain/pannings; at least one of them can be removed by modifying the parameters of the remaining. While this specific case is easy to detect and correct, it hints that sparser graphs might exist, which are harder to find.

Another notable behavior is that each pruning run of the same song results in a slightly different graph. We ran the pruning of `GirlOnABridge` 10 times; 4 of them are shown in Figure 6.12. 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, while the remaining will adapt and be further optimized, changing their “importance scores.”

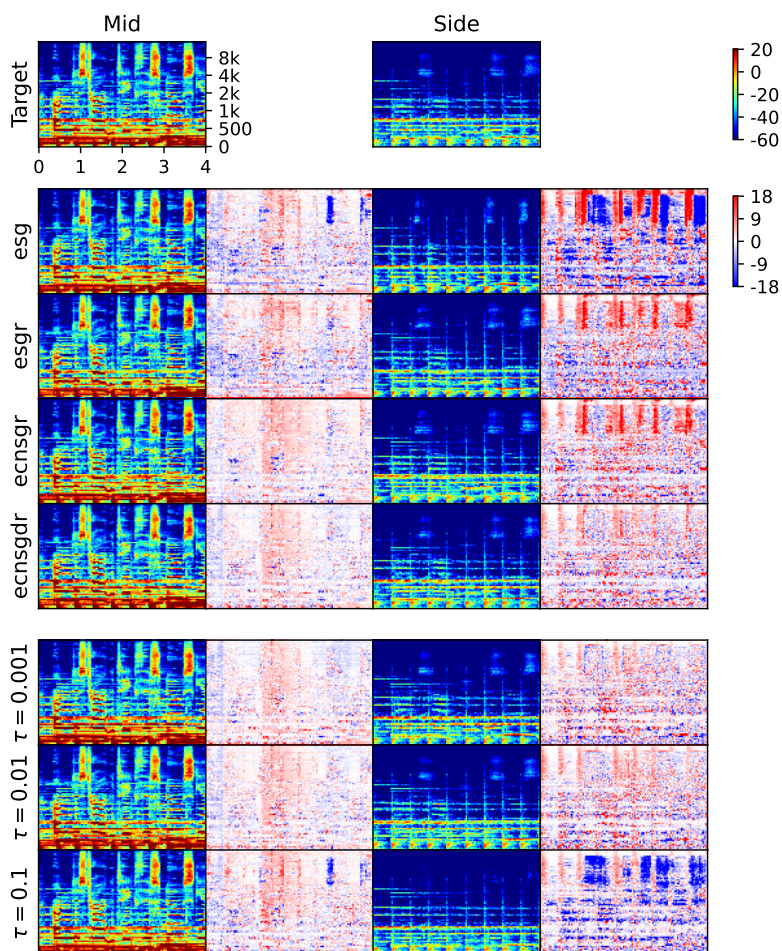


Figure 6.10: **Representative log-magnitude spectrograms of target mixes and matches.** Selected mixing console configurations and pruning tolerances are shown. Song: Torres_NewSkin.

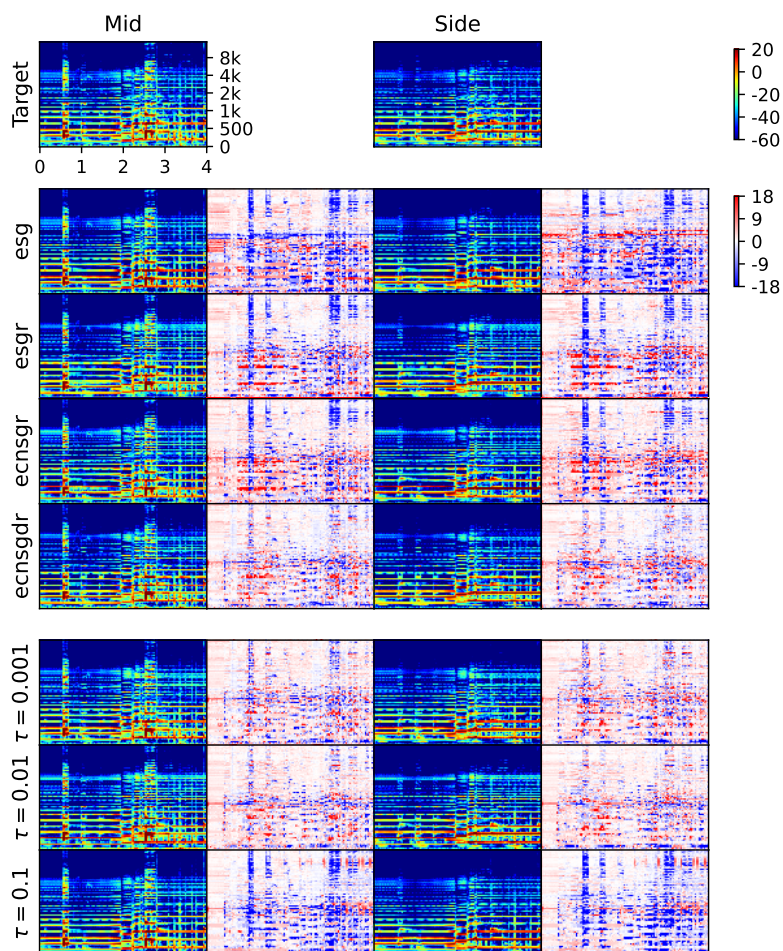


Figure 6.11: **Representative log-magnitude spectrograms of target mixes and matches (cont.).** Selected mixing console configurations and pruning tolerances are shown. Song: TablaBreakbeatScience_RockSteady.

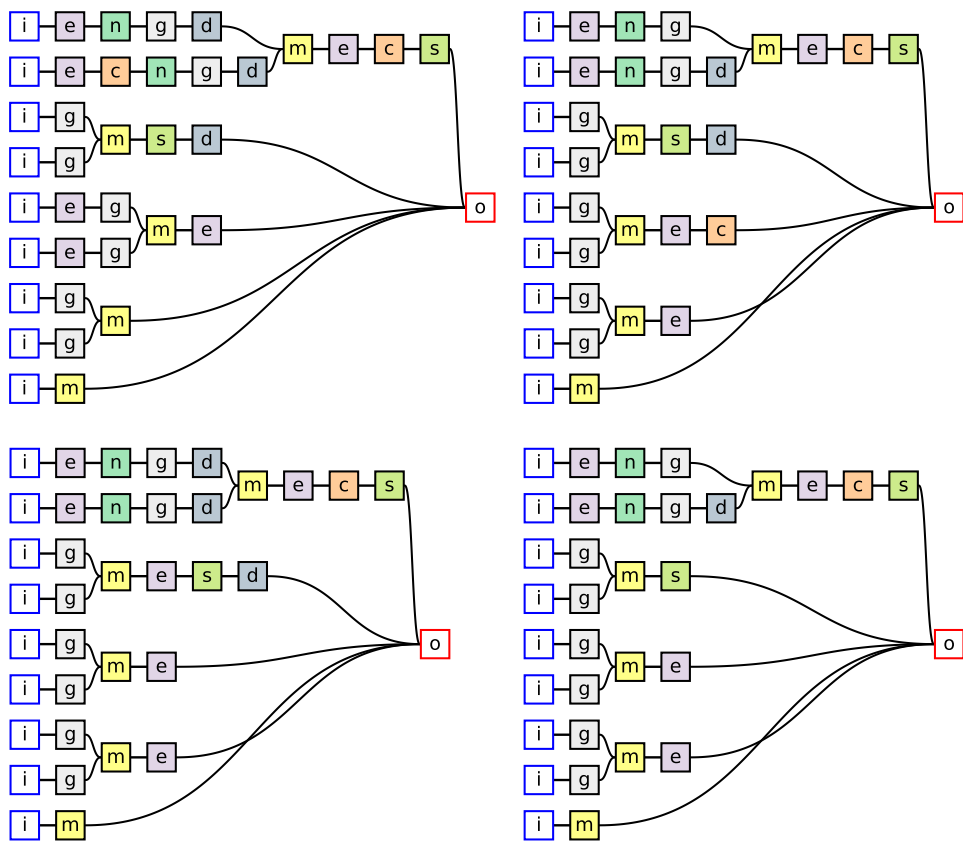


Figure 6.12: **Graphs obtained with multiple pruning runs (default setting).** Song: EthanHein_GirlOnABridge.

6.6.7 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, we can try a “consistency check,” where we treat the pruned result as a synthetic ground truth. Then, we can rerun the pruning with the matched mix $\hat{y}$ as a target and measure the similarity of the obtained graphs $\tilde{G}$ 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 6.6 summarizes the evaluation results; we report the per-type and micro-

Table 6.6: **Processor detection performance with consistency check.** The pruning method is evaluated under the default setting.

	μ	g	s	e	r	c	n	d
Accuracy $\uparrow$	.902	.888	.968	.923	.956	.914	.909	.757
Precision $\uparrow$	.846	.848	.871	.954	.931	.848	.773	.682
Recall $\uparrow$	.900	.955	.903	.917	.931	.935	.874	.768
F1 Score $\uparrow$	.872	.898	.887	.935	.931	.889	.820	.722

average metrics (denoted as μ). The average accuracy is 0.902, i.e., the pruning correctly predicts about 9 out of 10 whether a certain source track or subgroup mix is processed with a given processor type (or not). In terms of the F1 score, the micro-average 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 is not so surprising; as the pruning operates with a certain tolerance, it is biased towards false negatives rather than false positives.

We note that the mean audio loss result L_a of this experiment was 0.112, which is much lower than the default-setting results (0.301 and 0.563). This is because each target mix $\hat{y}$ is already projected to the subspace that our graph can express (shown as a teal-shaded area in Figure 6.1). Nevertheless, the loss value is not negligible. We suspect two reasons: the pruned graph $\tilde{G}$ is slightly different from the original G and the gradient descent struggles to find better parameters.

6.6.8 Full Results

Finally, we pruned the full dataset ensemble and collected the graph data for the downstream tasks. The evaluation results are reported in Table 6.7 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 Figure 6.13. Among the datasets, MedleyDB has the fewest source tracks, averaging 17.6 per song. In contrast, the Internal dataset has the most (28.8), closely followed by MixingSecrets (27.9). The Internal dataset also tends to have more subgroups, leading to larger mixing consoles. We suspect these contributed

Table 6.7: **Pruning results of the full dataset.** The default setting is used.

	$L_a \downarrow$	Pruning Ratio $\uparrow$								FAD $\downarrow$ SI-SDR $\uparrow$					MIR Feature Distance $\downarrow$				
		μ	g	s	e	r	c	n	d						d_{RMS}	d_{CR}	d_{SW}	d_{SI}	d_{BS}
MedleyDB	.431	.631	.365	.842	.444	.688	.735	.773	.570	.488	-4.11	-5.75	-.237	-3.67	-3.20	-1.32			
MixingSecrets	.625	.637	.498	.876	.334	.635	.633	.798	.685	.621	-12.2	-5.03	.254	-2.83	-2.82	-1.01			
Internal	.434	.755	.705	.871	.550	.728	.854	.857	.723	.280	0.28	-5.88	-.387	-3.72	-3.45	-1.47			
Total	.506	.692	.574	.868	.451	.686	.751	.821	.685	.443	-5.15	-5.54	-.120	-3.37	-3.17	-1.27			

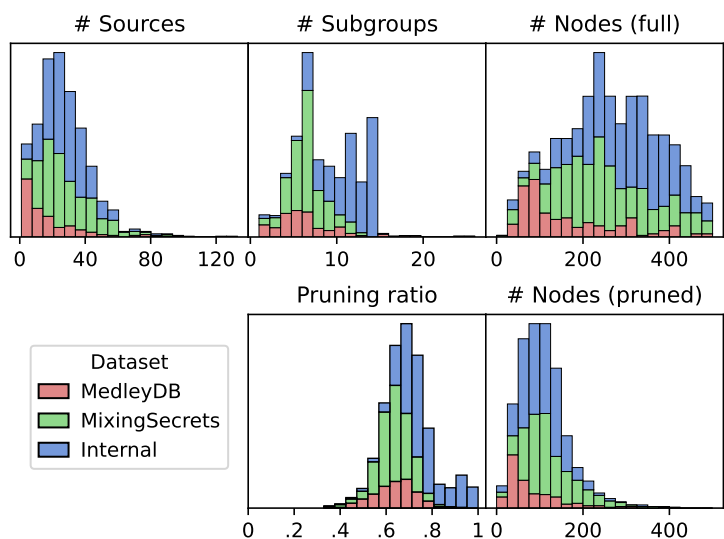


Figure 6.13: **Statistics of the consoles and pruned graphs (full data).** Per-dataset results are stacked to form the full histograms.

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), the per-song pruning ratios average 0.692.

6.7 Conclusion

We formulated the reverse engineering of the music mix as a search of the audio processing graph and its processor parameters. Then, we restricted the search to the iterative pruning of a pre-defined mixing console and derived a computationally feasible algorithm. 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. We chose our processors, 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 [37] and artificial reverberation [83] exist. Also, when we added the dynamic processors to the mixing consoles, the listening test results showed only a slight improvement. Approximated ballistics could be a potential cause, as recently reported [13]. We could expand the processor set to include effects like saturation [258] or modulation [7]. 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. We could add send-return loops, post-equalizers for compressors, and even multi-input or multi-output processors, such as those with sidechains or crossover filters. While we used already available subgroup information to construct a mixing console, it is possible to estimate it with audio feature analysis [259]. We could also jointly optimize the subgrouping information. However, we should extend the current framework, 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 [49, 260]. Exploring alternative objectives [261] 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, we explored three pruning variants with different granularities in search. The real-world results show that a more detailed search yields sparser graphs at a comparable match quality. However, it is not clear if this difference is important enough to justify the extra cost given the limited computation budget. We note that the final experiment, which consisted of more than a thousand individual optimization runs, took about three days even with 10 RTX 3090 GPUs.

While our method finds a sparse and interpretable mixing graph, it is still unclear how closely the pruning method finds the actual underlying graph. We

only have the following conservative conclusions: Our 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. Besides that, the graph’s plausibility is supported by the design of mixing consoles, their processors, and several qualitative case studies.

We list possible extensions of the pruning method. First, differentiability is not required for the pruning objective, allowing broader design choices, such as the MIR feature distances we used for the evaluation. Secondly, our 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 [240] could improve the candidate node selections. Finally, we can apply domain-specific methods, e.g., merging LTI processors into a single one, which could further increase sparsity.

Other than the aforementioned extensions, we may take a radically different approach, e.g., adopting the neural architecture search by relaxing the prior restrictions on graph structures [241, 242], or reinforcement learning [249, 248]. While these are promising, the main challenge will be balancing various aspects, such as the graph’s plausibility, match quality, sparsity, and the search method’s computation cost.

This chapter 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. We can extend prior works on statistical analysis of music mixes [32, 33, 237] by directly assessing the mixing graphs, not just the audio features. We can also treat the collected graphs as pseudo-labels and train neural networks to estimate the graphs (and their corresponding parameters). We can modify existing methods, e.g., automatic mixing [238, 239, 31, 262, 14, 263] and style transfer [264], to output the graphs, which lets end users interpret and control the outputs. While we collected graphs from an ensemble of multiple datasets, the amount of data could still be insufficient for the neural network training. Mitigating overfitting and improving generalization are left as future work.

Chapter 7

Blind Estimation of Audio Processing Graphs

7.1 Introduction

The reverse engineering method of Chapter 6 assumed access to the dry source tracks, so candidate processing graphs could be rendered on known inputs and compared against a target mix. This chapter removes that assumption and tackles *blind audio processing graph estimation*. Given only a processed audio signal y , we aim to estimate the processor types, parameters, and their routings without observing the original dry source. Since the dry source is not available, it is a much more ill-posed one-to-many task, and we address this by training a deep conditional graph generation model.

The blind graph estimation task poses several challenges to overcome. First, large paired graph-audio datasets with suitable processor vocabularies and topology coverage are not available. Hence, we generate a *synthetic dataset* for controlled evaluation. Second, audio processing graphs are heterogeneous: processor nodes and edges carry categorical types and continuous parameters. To mitigate this, we propose a two-stage decoding method consisting of (i) *prototype graph decoding*, which estimates graph connectivity and node/edge types, and (ii) *parameter estimation*, which decides the remaining parameters. We achieve this with a convolutional reference encoder [265] and a transformer-based graph decoder [195, 96].

We evaluate our method on singing voice effects and drum mixing estimation tasks. The evaluation results show that, while our model fails to perfectly recon-

struct the original graph in most cases, it preserves essential local/global graph structures, and rendered audio from the estimated graph can be perceptually similar to the reference. We also show that the proposed graph decoding strategy is preferable to other possible alternatives.

7.2 Training Data Generation

Graph notation — We use the same multiple-input multiple-output (MIMO) audio processing graph abstraction introduced in Chapter 5, so we only recall the attributes needed to define the supervised training data. A graph G is a heterogeneous directed acyclic graph (DAG) whose processor nodes are connected by edges between typed outlets and typed inlets. As in Chapter 1, G contains the connectivity and categorical node and edge types, while P denotes the collection of continuous node and edge parameters. We use uppercase P for a parameter collection and lowercase p with subscripts for an individual node, edge, or token parameter block. The complete audio processing graph is therefore described by (G, P) and rendered as $G(x; P)$ for an input x .

Processor vocabulary — Processor nodes may receive and output audio or control signals, and their output audio signals are normalized so that the total energy remains unchanged. Each node carries a categorical type t_i and continuous-valued parameters p_i as attributes. In this chapter, we instantiate the abstraction with 33 conventional processors listed in Table 7.1. Unless stated otherwise, we use the simplest, canonical implementations of the audio processors [9], which keeps the generated graphs interpretable within the chosen processor vocabulary. Note that, for the main experiments, the processors are treated as non-differentiable black boxes. The vocabulary includes various (possibly MIMO) processors such as crossovers, mid/side splitters/mergers, dynamic range controllers that accept sidechain input, and dedicated control signal generators such as LFOs and envelope followers.

Edge attributes — Edges keep the routing information needed to disambiguate MIMO connections. The outlet attribute m_{ij} and inlet attribute n_{ij} specify which output port of the node i connects to which input port of the node j ; these are

Processor(s): [inlets, optional*] → [outlets]; [parameters].
Low-order linear filters [5] • Second-order low/band/highpass, bandreject, and fourth-order low/band/highpass: [in, frequency*] → [out]; [frequency, q]. • Parametric equalizer filters - low/highshelf and bell (peaking filter): [in, frequency*, gain*] → [out]; [frequency, q, gain]. • Crossover: [in, frequency*] → [low, high]; [frequency]. • Phaser: [in, mod] → [out]; [frequency, feedback, mix].
High-order linear filters [266] • Chorus/flanger/vibrato: [in, mod] → [out]; [delay, feedback, mix]. • Mono and pingpong delay: [in] → [out]; [delay, feedback, mix, frequency, q, stereo_offset]. • Reverb (mono and stereo): [in] → [out]; [size, damping, width, mix].
Nonlinear filters • Distortion [267]: [in] → [out]; [gain, hardness, asymmetry]. • Bitcrush: [in] → [out]; [bit]. • Dynamic range controllers - compressor/noisegate/expander [10]: [in, sidechain*] → [out]; [threshold, ratio, attack, release, knee]. • Pitchshift: [in] → [out]; [semitone].
Utility processors • Mix: [in] → [out]; []. • Panning: [in, pan*] → [out]; [pan]. • Imager: [in] → [out]; [width]. • Mid/side splitter: [in] → [mid, side]; []. • Mid/side merger: [mid, side] → [out]; [].
Control signal generators • Low-frequency oscillator (mono and stereo): [] → [lfo]; [frequency, phase, stereo_offset]. • Envelope follower: [in] → [env]; [attack, release, gain].

Table 7.1: List of the processors and their configurations.

graph ports, not audio channels. We define the edge type t_{ij} as the outlet–inlet pair (m_{ij}, n_{ij}) . When multiple edges connect to the same inlet, their incoming signals are summed. Each edge also has a scalar gain parameter p_{ij} , which controls the contribution of that connection before summation. Thus, $P = \{p_i\}_{i \in V} \cup \{p_{ij}\}_{(i,j) \in E}$ collects all processor and edge parameters.

7.2.1 Prototype Graph Sampling

We first sample a *prototype graph* G , which contains the graph topology, processor types t_i , and outlet–inlet edge types t_{ij} , but not the continuous processor

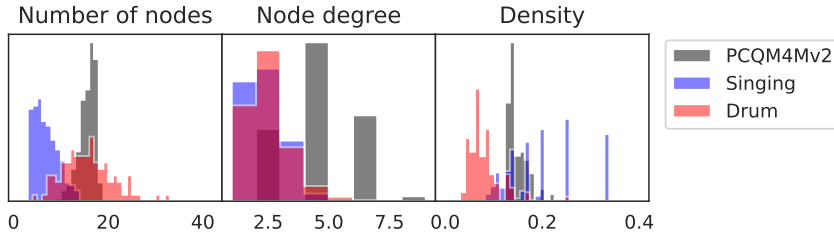
parameters p_i or edge parameters p_{ij} . The prototype graph is generated with a hand-designed graph sampling method rather than by sampling arbitrary DAGs (e.g., sampling triangular adjacency matrices), because audio processing graphs have specific nontrivial structures. For example, they often contain serial processor chains, local effect subgraphs, multiband splitting and merging, sidechain/control paths, and dry/wet or parallel branches.

To reflect this, we build a graph sampling method with stochastic motifs, where each motif is a small graph-generating primitive rather than a fixed processor label. For example, a distortion motif may include pre/post filtering around a nonlinear processor, a delay/reverb motif may include filtered or gated branches, and an imaging or pitch motif may create parallel or mid/side paths. The sampled motifs are usually arranged in series, but the sampling method can also insert `[crossover]` nodes for multiband processing and attach auxiliary control signals such as LFOs, envelope followers, and sidechains.

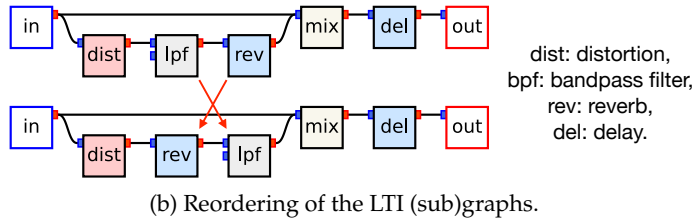
For singing voice graphs, we sample optional pre-processing filters or EQ, zero or more main effect motifs, optional multiband branches, and optional post-processing EQ. For drum mixing graphs, we first generate source-specific insert subgraphs for active tracks such as `[kick]`, `[snare]`, `[hat]`, `[tom]`, `[ride]`, and `[crash]`, then combine them through one or two mixing bus subgraphs with additional processing. This source-wise construction gives the drum graphs a mixing bus chain and several source chains. See Figures 7.1b, 7.4, and 7.5 for examples of the synthesized graphs.

7.2.2 Adaptive Parameter Randomization

Next, we decide the remaining parameters of each prototype graph. Specifically, we randomize them adaptively to the incoming signals, reducing the probability of inactive nodes that do not contribute to the final output signal. For example, when a `[highshelf]` receives a `[crash]` signal, its shelf frequency should be constrained to the high-frequency region to change the frequency response of the input audio. For frequency-selective filters, we compute the cumulative spectral-energy distribution of the input signal. We then sample the relevant frequency parameter from bins whose cumulative energy lies between 0.2 and 0.8. We use



(a) Statistics of the singing and drum graphs compared to the PCQM4Mv2.



(b) Reordering of the LTI (sub)graphs.

Figure 7.1: **Properties of the synthetic audio processing graphs.**

analogous procedures for the other low-order linear filters. For dynamic range controllers, we use the input energy envelope to determine their thresholds. This procedure generated 300k and 450k graph-reference audio pairs for the singing voice and drum mixing tasks, respectively. Each reference audio signal is stereo, 3.63s long, and sampled at 44.1kHz.

7.2.3 Properties of Synthetic Graphs

Graph Statistics — Figure 7.1a reports the graph statistics of our synthetic graphs. For context, we compare the synthetic audio graphs with the PCQM4Mv2 molecular graph dataset [268]. The singing voice and drum graphs are comparable in node count to PCQM4Mv2, but they are sparser, with lower node degree and density. This sparse long-range structure can limit standard message-passing GNNs [269] in capturing global graph structure.

LTI Reordering — Different audio processing graphs can produce the same output, making the blind graph estimation a one-to-many problem. In particular, serially connected SISO LTI systems can be reordered without changing the system response. We resolve only this SISO LTI ambiguity by canonicalizing their order according to size and processor type (see Figure 7.1b).

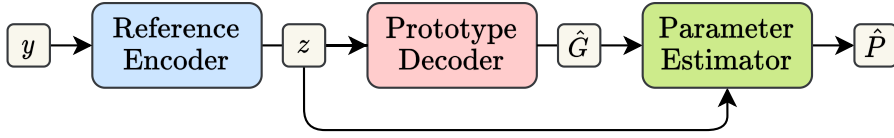


Figure 7.2: The proposed blind estimation framework.

7.3 Blind Estimation System

Our blind estimation system encodes the reference y into latent vectors z , then reconstructs the graph in two stages that mirror the synthetic data generation procedure: first, it decodes the prototype $\hat{G}$ autoregressively; second, it estimates the remaining processor and edge parameters $\hat{P}$ (see Figure 7.2).

7.3.1 Reference Encoder

We first apply 7 two-dimensional convolutional layers to the reference Mel spectrogram. Then, we flatten and project the channel and frequency axes of the output feature map into 512-dimensional vectors. Finally, we perform an attentive pooling across the time axis with learnable queries to obtain latent vectors z . The FFT size, hop length, and the number of Mel filter banks of the reference Mel spectrogram are 1536, 384, and 256, respectively. The convolutional backbone is a VGGish model [265], modified with depthwise separable convolutions [270], grouped dilated channels given by 1, 2, 4, and 8, and layer normalization [271]. We use a transformer decoder layer with learnable queries for the pooling; we use 1 and 6 queries for the singing and drum graph estimation, respectively.

7.3.2 Prototype Decoder

Token-by-Token Generation — Most autoregressive models [272, 95, 273] generate graphs *node-by-node*; for each step, they observe a partially decoded subgraph, then estimate the next node and its edges. Instead, we present an alternative method called *token-by-token* generation with Tokenized Graph Transformer (TokenGT, see Figure 7.3) [96]. TokenGT treats both nodes and edges as input tokens

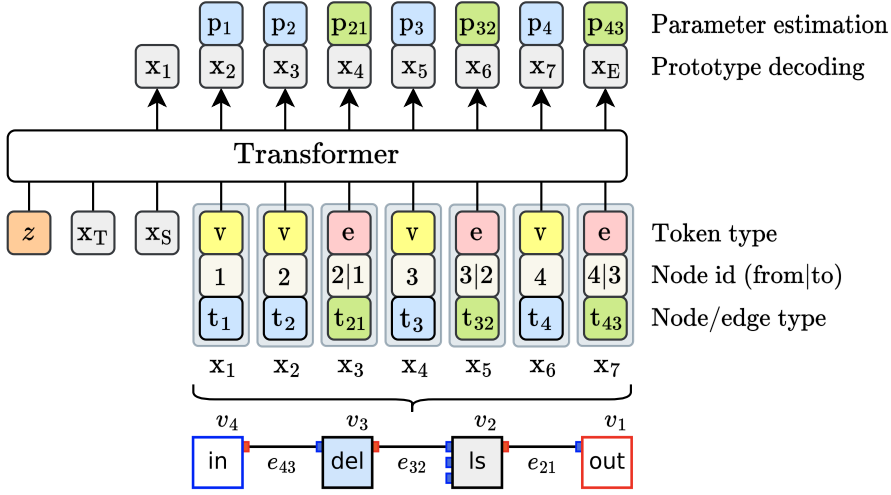


Figure 7.3: **Blind estimation with Tokenized Graph Transformer.**

to a standard transformer [195], using token type, node id, and node/edge type embeddings to encode graph structure. Note that, when using transformer architecture for graphs, a more conventional approach is to encode only nodes as tokens and inject the connectivity in the attention matrix [274]. We take the TokenGT approach, however, as it makes the autoregressive graph generation more straightforward. Treating edges as separate autoregressive tokens is also important for audio processing graphs because routing, modulation, and outlet–inlet choices can change the rendered audio as much as the processor identities themselves.

We serialize the prototype graph $G = (V, E)$ into $K = |V| + |E|$ graph tokens $\mathbf{x}_{1:K}$. The k -th token is

$$\mathbf{x}_k = (\tau_k, a_k, b_k, c_k, m_k, n_k), \quad (7.1)$$

where τ_k is the token type, a_k and b_k are node-ID fields shared by node and edge tokens, c_k is the processor type of a node token, and m_k and n_k are the outlet and inlet IDs of an edge token. Fields that do not apply to a token type are masked.

The token embedding is

$$\text{Emb}(\mathbf{x}_k) = \mathbf{e}_{\text{tok}}(\tau_k) + \begin{cases} \text{PE}(a_k) + \text{PE}(b_k) + \mathbf{e}_{\text{proc}}(c_k), & \tau_k = \text{node}, \\ \text{PE}(a_k) + \text{PE}(b_k) + \mathbf{e}_{\text{out}}(m_k) + \mathbf{e}_{\text{in}}(n_k), & \tau_k = \text{edge}, \\ \mathbf{0}, & \tau_k = \text{end} \end{cases} \quad (7.2)$$

Here, $\text{PE}(\cdot)$ denotes the sinusoidal node-position embedding. For a node token corresponding to the j -th decoded node, $a_k = b_k = j$ and c_k is the processor type. For an edge token from the j -th decoded node to the ℓ -th decoded node, $a_k = j$, $b_k = \ell$, and (m_k, n_k) specifies the source outlet and destination inlet.

We prepend a start token $\mathbf{x}_0 = \mathbf{x}_S$, append an end token $\mathbf{x}_{K+1} = \mathbf{x}_E$, and write $\mathbf{x}_{<k} = (\mathbf{x}_0, \dots, \mathbf{x}_{k-1})$. Because the serialization is canonical, the decoder models

$$p_\theta(G \mid y) = p_\theta(\mathbf{x}_{1:K+1} \mid z) = \prod_{k=1}^{K+1} p_\theta(\mathbf{x}_k \mid \mathbf{x}_{<k}, z). \quad (7.3)$$

Implementation-wise, the fields of one token are predicted in parallel from the same hidden state $h_k = f_\theta(\text{Emb}(\mathbf{x}_{<k}), z)$. This gives the factorized approximation

$$p_\theta(\mathbf{x}_k \mid \mathbf{x}_{<k}, z) \approx p_\theta(\tau_k \mid h_k) \begin{cases} p_\theta(a_k \mid h_k) p_\theta(b_k \mid h_k) p_\theta(c_k \mid h_k), & \tau_k = \text{node}, \\ p_\theta(a_k \mid h_k) p_\theta(b_k \mid h_k) p_\theta(m_k \mid h_k) p_\theta(n_k \mid h_k), & \tau_k = \text{edge}, \\ 1, & \tau_k = \text{end}. \end{cases} \quad (7.4)$$

We empirically find this approximation reasonable and to have minimal effect on performance.

GNN Expressivity — The architecture used here is a tokenized graph transformer rather than the message-passing neural network (MPNN) conventionally used for graph-structured input [269]. The first reason is that estimating an audio processing graph requires global reasoning. Message passing communicates through local neighborhoods, whereas processors can be jointly identifiable only through their combined behavior. In a serial equalizer chain, for example, similar over-

all responses can result from different allocations of gain and bandwidth across stages. Comparing such coupled processors through local aggregation requires multiple layers and can create bottlenecks when information must travel over long graph distances [275]. TokenGT instead represents every node and edge as a token, allowing any two processors or connections to interact directly through global self-attention [96].

The second reason is structural expressivity. Standard message-passing architectures have known limitations in distinguishing graph structures, whereas tokenized graph transformers can represent richer structural distinctions [276, 96]. Treating edges as explicit tokens also preserves typed routing relations rather than absorbing them into neighborhood aggregation. These properties suit the joint estimation of processor identities, routing structure, and parameters.

Decoding Order — The token serialization follows a fixed chain-wise reverse order. It lists the output or mixing bus chain first, then source chains in a fixed canonical order; for drums, this means the mixing bus subgraph is decoded before the individual source-track subgraphs. Within each chain, nodes are listed in breadth-first order. After each node token, edge tokens from the new node to previously listed nodes are emitted in increasing destination-node index.

Architecture Details — For the graph decoder, we use a standard 6-layer transformer encoder with 512 hidden dimension, pre-layer normalization [200], and 16 heads. The original TokenGT uses eigenvectors of the normalized Laplacian matrix as node-id encodings. We replace them with sinusoidal node-position embeddings, as during autoregressive decoding only partially decoded graphs are available, i.e., the final-graph eigenvectors are inaccessible.

7.3.3 Parameter Estimator

We reuse the prototype decoder for the parameter estimation; we add another projection head, append a task token $\mathbf{x}_T$ to differentiate the two tasks, and remove the causal attention mask. Because the full prototype is already available when parameters are estimated, removing the causal mask lets each parameter attend to the entire graph rather than only the preceding tokens, giving every parameter

head bidirectional context over the whole structure. Since each parameter has a different range and scale, we translate and rescale the ground-truth value to fit into the $[0, 1]$ range.

7.3.4 Training

We train the prototype decoder and parameter estimator jointly. With teacher forcing, the prototype loss is the sum of four label-smoothed cross-entropy terms:

$$\mathcal{L}_{\text{Prototype}} = \mathcal{L}_{\text{Token}} + \mathcal{L}_{\text{ID}} + \mathcal{L}_{\text{Node}} + \mathcal{L}_{\text{Edge}}. \quad (7.5)$$

$\mathcal{L}_{\text{Token}}$ supervises the token type τ_k ; $\mathcal{L}_{\text{ID}}$ supervises the node-ID fields a_k and b_k , which are shared by node and edge tokens; $\mathcal{L}_{\text{Node}}$ supervises the node type c_k of node tokens; and $\mathcal{L}_{\text{Edge}}$ supervises the outlet and inlet IDs m_k and n_k of edge tokens. We use a label-smoothing weight of 0.1 for all four terms. The parameter estimator is trained on the oracle prototype G with $\mathcal{L}_{\text{Parameter}}$, a weighted mean ℓ_1 loss on the normalized processor parameters p_i and edge parameters p_{ij} . The processor- and edge-parameter terms are weighted by 8 and 0.4, respectively. The overall objective is the sum $\mathcal{L}_{\text{Prototype}} + \mathcal{L}_{\text{Parameter}}$. We use AdamW [185] optimizer, a linear learning rate scheduler with $5\text{e-}4$ peak learning rate, 50k warmup steps, 200k total training steps, and batch size of 32.

7.4 Experimental Setup

7.4.1 Data

Singing — We used the OpenSinger dataset [216], which has 50h clean recordings from 76 speakers. We used 90% of the audio from the 71 speakers for the training. We use the remaining 10% and the other 5 speakers’ recordings as two separate validation sets, *seen* and *unseen* speaker datasets, respectively, to compare the effect of dry source distribution shift on the model performance.

Drum — We collected the source signals by ourselves; we rendered the `[kick]`, `[snare]`, `[hat]`, `[tom]`, `[ride]`, and `[crash]` tracks separately with 14 com-

mercial sampling libraries and MIDI files from the Groove MIDI Dataset [277]. To generate each reference audio, we sampled a random segment from the dry tracks. Then, we generated a graph with the input nodes corresponding to non-zero energy tracks so that there are no dummy subgraphs. This indicates that our system should also perform a drum instrument recognition task. We used the 11 kits and the MIDI files from the `drummer1/session1-3` subset for the training. For validation, the MIDI files come from `drummer1/eval_session`. Paired with the same 11 kits, they form the *seen* kit validation set; paired with the remaining 3 kits, they form the *unseen* kit validation set.

Dry Signal Normalization — The blind estimation task assumes that the reference audio is generated from a normalized dry signal and an audio processing graph. Without this assumption, source-dependent spectral coloration can be absorbed into the estimated graph. We therefore normalized the dry sources before rendering the synthetic graph data. This is similar in spirit to Fx-normalization for automatic music mixing with out-of-domain data [31], but here we use only equalization to define the dry source distribution. For singing, we equalized the dry recordings to a common long-term magnitude profile. For drums, we equalized each kit so that each instrument type has the same average frequency response across kits. This preprocessing reduces the tendency to explain speaker or kit coloration as processing effects.

7.4.2 Metric

Prototype Decoding — For the teacher-forced decoding, we evaluate the node and edge error rate. The node error rate counts a decoded node token as wrong if its token type, node id, or node type is incorrect, and the edge error rate counts a decoded edge token as wrong if its token type, endpoints, or edge type is incorrect; both are averaged over the decoding steps of the validation set. We also compare each complete decoded graph with the ground truth using two graph-level metrics. The invalid graph rate counts a graph as invalid if it is cyclic, disconnected, or missing necessary connections. The node-type intersection-over-union (IOU) ignores edge structure, but checks whether the required input nodes and processors are decoded somewhere in the graph; for the drum task, we calculate the IOU for

each track and mixing subgraph and average the values.

Parameter Estimation — We report the parameter loss $\mathcal{L}_{\text{Parameter}}$, computed on the oracle prototype.

Rendered Audio Similarity — We compare rendered audio using a power-compressed multi-scale spectral (MSS) loss $\mathcal{L}_{\text{MSS}}$ [47]. At each scale, we compute the ℓ_1 distance between magnitude spectrograms power-compressed with an exponent of 0.3, averaged over channels and time-frequency bins. We use FFT/hop-size pairs (512, 128), (1024, 256), (2048, 512), and (4096, 1024), and normalize both signals by the RMS of the reference before computing the distances. Let $G(x; P)$ denote the audio rendered from a prototype G with parameters P on the dry input x , let $\hat{G}$ and $\hat{P}$ be the estimates, P_0 the default parameters, and $y = G(x; P)$ the reference. The three reported variants apply the same loss to different renders,

$$\begin{aligned} (\text{MSS-default}) &= \mathcal{L}_{\text{MSS}}(\hat{G}(x; P_0), G(x; P_0)), \\ (\text{MSS-oracle}) &= \mathcal{L}_{\text{MSS}}(G(x; \hat{P}), y), \\ (\text{MSS}) &= \mathcal{L}_{\text{MSS}}(\hat{G}(x; \hat{P}), y), \end{aligned} \tag{7.6}$$

so that MSS-default fixes both renders to the default parameters and isolates the decoded structure, MSS-oracle keeps the ground-truth structure and isolates the estimated parameters, and MSS measures the fully decoded graph end to end. MSS-oracle is undefined for the single-stage baseline, which produces no separate oracle-prototype rendering.

Listening Test — We measured subjective scores with Multiple Stimuli with Hidden Reference and Anchor (MUSHRA) test [129, 130]. We asked 8 graduate students to score the similarity between the reference and the rendered audio with the estimated graph. A total of 48 sets were scored (24 sets for each task and 12 sets for each *seen* and *unseen* speaker/kits).

7.4.3 Baselines

The ③ proposed model is the blind estimator described above: it decodes the prototype G token by token, then estimates the parameters P on the completed

prototype with the causal mask removed. We compare it against two reference bounds, a source-conditioning variant, and two alternative decoding strategies. Unless noted, they share the same TokenGT backbone, and each is marked with a symbol used throughout the results.

Reference bounds — The ① graph autoencoder replaces the reference encoder with a second TokenGT that encodes the ground-truth graph, including its parameters, into the latent z . The decoder is then conditioned on a near-complete description of the target, so this upper-bounds what the decoder can reconstruct and verifies that the latent dimension is sufficient. The ② no-conditioning decoder sets $z = 0$, removing all information about the target graph; it lower-bounds the metrics and exposes the prior the decoder learns from the data distribution alone.

Source-conditioning variant — The ④ source-conditioning variant additionally receives the dry source x , concatenated with the reference y along the channel axis at the encoder input, so the convolutional front end sees both spectrograms. This is a reverse engineering configuration rather than a blind estimator, since x is unobserved in the blind setting. Relative to the proposed model, it can be read as an architecture-matched upper bound under the additional dry source information, isolating the degradation caused by blind estimation.

Decoding-strategy baselines — Two baselines change how the prototype is decoded. The ⑤ node-by-node decoder [95] keeps the two-stage split but serializes the prototype over nodes rather than tokens. Let $G^{<j}$ denote the partial graph produced by the first $j - 1$ node-wise decoding steps. The next node and its incoming edges are factorized as

$$p_\theta(G \mid y) = \prod_{j=1}^{|V|} \left[p_\theta(v_j \mid G^{<j}, z) \prod_{\ell < j} p_\theta(e_{\ell j} \mid v_j, G^{<j}, z) \right], \quad (7.7)$$

a next-node-type prediction followed by typed edge predictions between the new node and previously decoded nodes. Thus, $G^{<j}$ is a node-wise partial graph, whereas $\mathbf{x}_{<k}$ in Equation (7.3) is a prefix of the token serialization. Because the baseline’s edge decisions are made per node pair rather than as first-class tokens in the output sequence, the decoder can emit inconsistent connectivity, the source

of its higher invalid rate. The ⑥ single-stage variant keeps the token serialization but collapses the two stages into one autoregressive pass, emitting the categorical token and continuous parameters jointly at each step:

$$p_{\theta}(G, P \mid y) = \prod_{k=1}^{K+1} p_{\theta}(\mathbf{x}_k, p_k \mid \mathbf{x}_{<k}, P_{<k}, z). \quad (7.8)$$

Here, p_k is the processor parameter block p_i for a node token or the scalar edge parameter p_{ij} for an edge token; the end token has no parameter, and $P_{<k} = (p_1, \dots, p_{k-1})$ is the parameter prefix. Each parameter is conditioned only on the causal prefix $(\mathbf{x}_{<k}, P_{<k})$, whereas the two-stage parameter estimator drops the causal mask and conditions every parameter on the completed prototype.

7.5 Evaluation Results

For the singing voice and drum mixing tasks, Table 7.2 summarizes the graph decoding, parameter estimation, audio-domain, and listening test results. Figures 7.4 and 7.5 show representative decoded graphs and illustrate the main remaining failure modes.

7.5.1 Sanity Check

Before training the blind estimation models, we first verified the ① graph autoencoder. Its evaluation results, e.g., 0 node error rate, confirm that the graph decoder is powerful enough and the dimension of the latent z is sufficiently large to reconstruct the original graph. Furthermore, its MUSHRA score is comparable to the hidden reference’s, agreeing with the objective metrics. Next, we evaluated the ② graph decoder with latent vectors set to 0. Its node error rate (0.502 and 0.602) is better than what a random guess would achieve. This is because (i) the probability distribution of the processors is nonuniform, (ii) we sorted the LTI subgraphs, and (iii) the ground-truth intermediate prototype is available to the network, which can be exploited for a better guess.

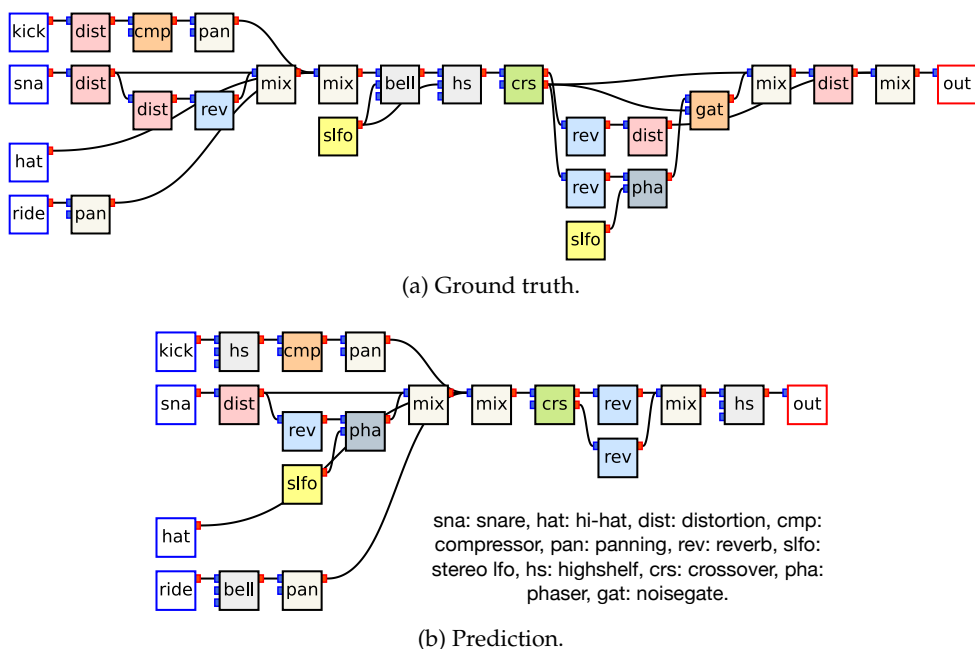


Figure 7.4: **An inference result from the proposed method ③ on the drum mixing estimation task.** First, our model correctly predicted the source instrument types: [kick], [snare], [hat], and [ride]. The ground-truth graph panned [kick] and [ride], which is also reconstructed by the prediction. However, the prediction failed to estimate (i) the correct multi-band processing of [reverb], (ii) multiple stages of [distortion] processors, and (iii) modulation of linear filters in the mixing subgraph.

7.5.2 Performance Analysis

On the *seen* dry speaker/kit sets, the ③ proposed model reports the node error rate of 0.215 and 0.335 for the singing and drum task, respectively, indicating that the perfect reconstruction of the prototype graph is rare. Yet, audio rendered from the estimated graph can be perceptually close to the reference, reporting the MUSHRA score of $76.6_{\pm 4.6}$ and $69.8_{\pm 5.4}$. On the *unseen* dry speaker/kit sets, the evaluation results are degraded in most metrics, confirming that the graph estimation from unseen sources is challenging. We note that the majority of the errors come from either (i) the wrong order of processors (see Figure 7.4 and 7.5a) or (ii) some processors which destroy the signal and make preceding processors harder to notice (see Figure 7.5b). Finally, to check the difficulty of blind estimation, we trained the same model but with ④ dry sources also provided as input by

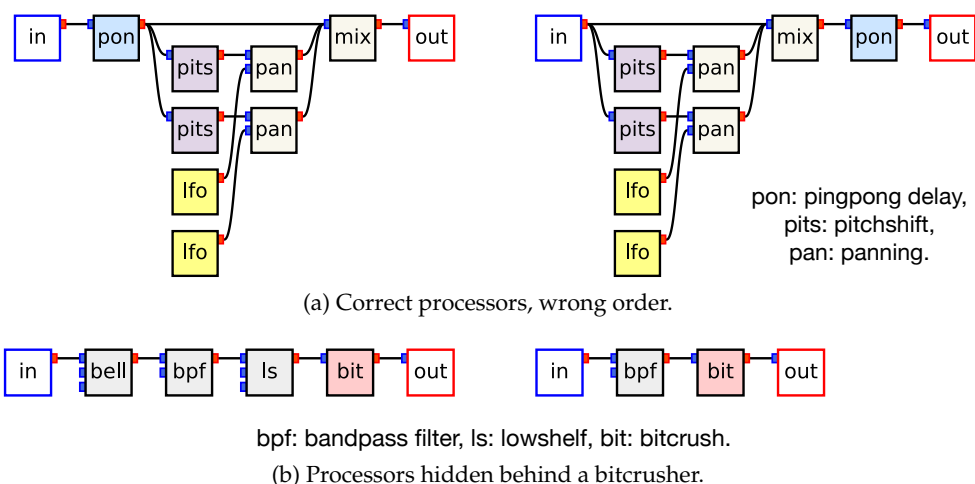


Figure 7.5: **Inference results from the proposed method ③ on the singing voice effect estimation task.** Left: ground truth, right: prediction. In (a), our model successfully predicted that the input is processed with a [pingpong_delay] and two parallel [pitchshift] modules (each with panning modulation), but failed to estimate their correct order. In (b), it failed to decode the [bell] and [lowshelf]. This is not surprising, because if the bit parameter of the [bitcrush] is set to a very low value, it destroys the signal and eventually makes estimation of the processors applied before the [bitcrush] challenging.

concatenating them with the reference across the channel axis. The estimation performance improves by a noticeable margin, indicating that extracting the graph-relevant information solely from the reference is a challenging task. Since this configuration observes the dry source, it should be read as a reverse engineering diagnostic rather than a blind estimator.

7.5.3 Blind Estimation Strategy Comparison

We compare the ③ token-by-token decoder with the ⑤ node-by-node and ⑥ single-stage alternatives. The node-by-node decoder performs similarly or slightly worse overall, and because it predicts each edge per node pair after the node rather than as an explicit token, it produces inconsistent connectivity and a high invalid rate (0.072 and 0.132, against 0 for the proposed model). The single-stage decoder conditions each parameter only on the causal prefix instead of the completed prototype, so its parameter estimation is markedly worse (parameter loss 0.168 and

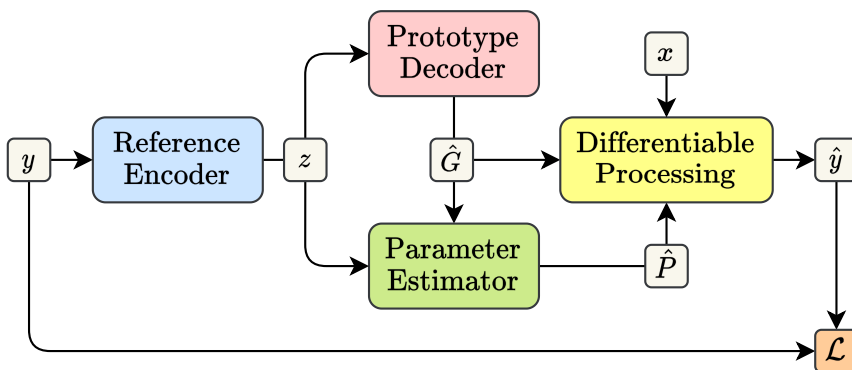


Figure 7.6: End-to-end refinement with differentiable processing.

0.213, against 0.125 and 0.161 on the same unseen sets) and its MUSHRA lower, even though its node error rate stays comparable; its MSS-oracle is undefined, as it has no separate oracle-prototype pass.

7.6 End-to-End Refinement with Differentiable Processing

The system described so far is supervised by a discrete prototype loss and continuous parameter regression. Neither term directly measures audio similarity, and the parameter loss is computed under teacher forcing rather than through the audio rendered by the estimated graph. This mismatch motivates an end-to-end refinement stage in which the predicted graph is rendered and its audio output provides the training signal. We therefore investigate whether differentiable rendering can refine the estimator according to an audio-domain objective.

7.6.1 Fine-Tuning with Differentiable Rendering

Figure 7.6 illustrates the resulting fine-tuning procedure. Starting from the converged supervised model, we render the predicted graph and backpropagate through it with the batched differentiable graph renderer of Chapter 5, using differentiable counterparts of the processors included in this extension, and opti-

Table 7.2: **Evaluation results of the proposed method and other baselines.** U and S denote unseen and seen speaker/kit validation set, respectively. Left: singing voice effect estimation. Right: drum mixing estimation.

Methods	Singing Voice Effect						Drum Mixing					
	Node error rate	Edge error rate	IOU	MSS-default	MSS-oracle	MUSHRA	Node error rate	Edge error rate	IOU	MSS-default	MSS-oracle	MUSHRA
Hidden reference	—	—	—	—	—	92.0 $\pm$ 3.4	—	—	—	—	—	90.0 $\pm$ 3.2
① Autoencoding	0 .002 .005	1 .002 .008 .004 .004				85.7 $\pm$ 5.0	0 .001	0	1 .001 .002 .018 .018			90.3 $\pm$ 4.8
② No latent conditioning	.502 .014	0 .365 .116 .200 .095 .107				22.1 $\pm$ 4.5	.602 .024	0 .267 .203 .207 .138 .187				32.8 $\pm$ 8.0
③ Token-by-token, two-stage	S .215 .007	0 .839 .044 .120 .056 .058				76.6 $\pm$ 4.6	.335 .019	0 .659 .100 .145 .080 .085				69.8 $\pm$ 5.4
	U .242 .009	0 .800 .054 .125 .057 .059				75.6 $\pm$ 4.7	.446 .015	0 .530 .159 .161 .088 .119				46.4 $\pm$ 5.7
④ ③ with source conditioning	U .169 .011	0 .906 .034 .123 .054 .052				77.5 $\pm$ 5.1	.411 .019	0 .550 .137 .160 .092 .110				59.3 $\pm$ 5.9
⑤ Node-by-node, two-stage	U .267 .052 .072 .742 .064 .142 .065 .066					60.9 $\pm$ 6.0	.461 .045 .132 .443 .134 .193 .106 .134					39.2 $\pm$ 5.3
⑥ Token-by-token, single stage	U .249 .020 .002 .799 .053 .168	— .070				62.4 $\pm$ 5.6	.442 .023 .023 .483 .159 .213	— .144				40.6 $\pm$ 5.3

minimize the same multi-scale spectral loss $\mathcal{L}_{\text{MSS}}$ used for evaluation. A spectral loss only helps where its gradient is informative, so we first measured recoverability per processor and per parameter. For each processor we sampled a random target, rendered it, and optimized the parameters from a different initialization on the same loss. Most processors recover well under direct optimization. However, several processors included in this extension are difficult to optimize with the MSS loss alone. This is especially true for time- and frequency-misalignment effects, such as pitch shifting, delays, and modulation, where small shift errors can create poor local minima. These processors may require additional objectives or processor-specific optimization tricks for reliable parameter recovery. We do not introduce those tricks here; instead, we use this experiment as a first test of whether differentiable rendering can still provide a useful audio-domain training signal despite these limitations.

Let $\hat{G}$ be the decoded prototype and $\hat{P}$ the predicted processor and edge parameters. We render either the decoded prediction $\hat{y} = \hat{G}(x; \hat{P})$ or the oracle-structure prediction $\hat{y} = G(x; \hat{P})$, and minimize the spectral loss $\mathcal{L}_{\text{MSS}}(\hat{y}, y)$. In both cases, the graph topology is treated as discrete, so the audio gradient updates the reference encoder, the parameter pathway, and differentiable edge parameters, but not the structure decoder. The dry input x is unobserved at inference yet available during training, so the audio objective is applied only as a training-time refinement. We add it, weighted by λ , as a warm continuation from the converged supervised baseline. Some ablations keep the parameter loss, while others remove it so that the refinement is driven only by the render loss.

7.6.2 Refinement Results

Table 7.3 reports differentiable refinement variants on the singing voice validation set. The ① supervised baseline obtains an MSS of 0.0628, with node error rate 0.275, edge error rate 0.005, IOU 0.744, parameter loss 0.117, and MSS-oracle 0.0571. The ② supervised continuation reaches an MSS of 0.0631 after the same fine-tuning budget, so the improvement below is not explained by additional optimization steps alone.

The ③ decoded-graph refinement with both audio and parameter losses im-

Table 7.3: **Differentiable processing refinement on the singing voice task.**

Setting	Node error rate	Edge error rate	IOU	Parameter loss	MSS-oracle	MSS
① Supervised baseline	.275	.005	.744	.117	.0571	.0628
② Supervised continuation	.275	.006	.741	.116	.0569	.0631
③ Audio + parameter loss (decoded graph)	.277	.005	.745	.118	.0564	.0615
④ Audio loss only (oracle graph)	.269	.005	.745	.159	.0553	.0596
⑤ Audio loss only (decoded graph)	.273	.005	.759	.161	.0557	.0592

proves the audio-domain metrics, reducing MSS to 0.0615 and MSS-oracle to 0.0564, while leaving the graph metrics essentially unchanged (node error 0.277, edge error 0.005, IOU 0.745). The ④ oracle-graph refinement with audio loss only improves the audio match further, reaching MSS 0.0596 and MSS-oracle 0.0553. The matched ⑤ decoded-graph refinement reaches a similar MSS of 0.0592 and MSS-oracle of 0.0557, again with similar graph metrics (node error 0.273, edge error 0.005, IOU 0.759). The parameter loss rises from 0.117 to 0.161 in this setting, but this should not be read as a degradation of the primary objective: the refinement is optimized to match rendered audio, and many processor parameters are non-identifiable, inaudible, or only weakly audible under a given signal. Rather, the higher parameter loss confirms that audio-domain refinement can move away from the particular synthetic parameterization when direct parameter supervision is removed. The similar oracle- and decoded-graph results support the audio-domain refinement itself but do not support a strong claim that topology choice is the main lever. These results should be read together with the optimization caveat above: the differentiable extension includes processors that the MSS loss does not optimize reliably without additional tricks, so the experiment demonstrates a useful training signal rather than a complete solution for every processor parameter. Finally, these results keep differentiable edge parameters enabled, but they do not isolate their contribution; the role of differentiable edge parameters should therefore be treated as an implementation hypothesis rather than a demonstrated causal factor.

7.7 Conclusion

This chapter presented a blind estimation system for audio processing graphs. The results show that token-by-token generation is preferable to node-by-node alternatives and that separating prototype decoding from parameter estimation is beneficial. The differentiable processing extension further shows that refining the decoded graph with differentiable processors and an audio-domain objective can improve estimation over graph and parameter supervision alone.

Several limitations remain. The experiments use a controlled synthetic distribution, so broader processor vocabularies, more diverse source signals, and graph data closer to real production sessions are needed before the method can be read as a general-purpose blind estimator. The graph representation is also restricted to a single-edge DAG, which simplifies tokenization and autoregressive decoding but excludes feedback loops, repeated parallel connections, and other routing patterns that appear in real signal chains.

The model also inherits limitations from the chosen decoder. It uses a single canonical serialization order and a standard transformer with sinusoidal encodings; future graph decoders could make the generation less order-dependent, for example, with order-agnostic graph generation, masked-token decoding, diffusion-style graph decoders, or architectures that inject graph connectivity more explicitly. The results on unseen source distributions further show that the reference encoder must better separate processing-relevant cues from source-dependent content.

The differentiable processing experiment is also limited to the singing voice task. Extending this refinement to the drum mixing task, handling processors that are non-differentiable or difficult to optimize with spectral losses, and combining conventional differentiable processors with neural audio processors remain open directions. The same blind estimation framework could also be extended to broader graph-based applications, including automatic processing [31, 278, 279] and style transfer [57].

Chapter 8

Conclusions

8.1 Thesis Summary

This thesis studied inverse problems of audio systems: estimating the system that transformed an observed signal. In particular, system estimation was formulated as the problem of choosing a proxy that is expressive enough while keeping its description space interpretable and amenable to the estimation algorithms. Within this lens, two complementary approaches, differentiable processing and generative modeling, were explored across two system classes: linear time-invariant systems and compositional systems, with room acoustics and music applications.

For room acoustics, Chapter 2 showed that a range of classical artificial reverberators can be made differentiable and used for end-to-end estimation of filter parameters from reference signals, either RIRs or reverberant speech. Chapter 3 extended from estimating a single source–receiver response to reconstructing an entire acoustic field with a differentiable geometric acoustics simulator equipped with learnable surface material properties. For both chapters, various techniques, e.g., frequency-domain computation, were introduced to improve the speed of forward computation and make the optimization feasible. Chapter 4 introduced discrete representation learning combined with an autoregressive model for RIR estimation. It also introduced a hybrid extension that employed differentiable late reverberation modeling to improve the compression rate and reduce the inference cost.

For musical signal processing, the thesis focused on compositional audio systems. Chapter 5 first developed a common representation of heterogeneous differentiable audio processing graphs and an efficient batched algorithm for rendering them. Building on it, Chapter 6 introduced a graph reverse engineering algorithm: starting from a differentiable mixing console, the method optimized processor parameters and iteratively pruned inactive structure to obtain sparse, interpretable graphs from dry multitracks and target mixes. Chapter 7 addressed the more challenging blind graph estimation task; to tackle this, an autoregressive model on a tokenized graph was introduced. Further, another hybrid extension that refines the parameter estimation quality with differentiable audio processing graphs was explored.

Together, these chapters show that differentiable processing and generative modeling are complementary rather than competing approaches to audio system estimation, and that the two can be combined within a single system description.

8.2 Discussion

We conclude by revisiting some of the key framing of the thesis, along with its limitations, open questions, and future directions.

8.2.1 Expressivity and Model Choice

The choice of proxy model determines the upper bound of the system estimation quality. A more expressive model is not automatically more useful, however: its parameters should remain meaningful for the intended application, its computation should remain practical, and its parameterization should permit reliable optimization. These properties do not necessarily form a strict tradeoff, but they must be considered together, and how to balance them remains open.

In room acoustics, the thesis studied three model classes with different capabilities. The artificial reverberators expose compact and controllable parameters but cannot represent an arbitrary RIR. The learned discrete representation can describe free-form RIRs, but its coverage and resolution follow from the training data rather than from an acoustic model. The geometric acoustic model is

geometry-aware but only represents energy transport under the assumption of a high-frequency limit and cannot model phase-dependent phenomena such as diffraction and interference. While a differentiable wave-based method is, in principle, possible, it was not explored because of the high computational cost. The geometric model is not free of this cost either: the number of radiance variables and the cost of propagation grow with the spatial and directional resolution, so the level of detail it can describe is bounded as much by the available computation as by the model itself. At present, geometric acoustics [280, 160] and geometry-aware extensions of artificial reverberators [175, 164] remain the practical approaches, both for optimization and for auralization.

The same issue arises in musical signal processing. While the thesis explored a general graph form for musical signal processing, its expressivity is limited not only by the set of processors available for the graphs, but also by how each of them is implemented. Expanding that set also expands the space of graphs that must be searched or learned, so expressivity cannot be widened independently of estimation cost. Differentiable implementations of linear time-invariant processors are relatively more established, while nonlinear and time-varying ones remain the harder case and still call for processor-specific treatment [7, 281]. Distortion alone spans a large number of circuits and plugins, and writing a differentiable implementation for each of them is not realistic. A neural or gray-box processor can instead be used as a node within the same graph, keeping the routing and any meaningful high-level controls explicit while learning behavior that is difficult to reproduce with a conventional implementation. Prior gray-box studies show that the appropriate balance depends on the device, available data, and computational requirements [282, 283]. Hybrid models should therefore be validated against the real processors they are intended to represent.

8.2.2 Data Availability and Generalization

Data availability varies considerably even within room acoustics, depending on the task that one aims to solve. For monaural RIR estimation, one can use a combination of arbitrary RIR datasets as a training dataset. This ensemble, while still small by modern machine learning standards, has become sufficiently large

to train a neural generator. On the other hand, learning an acoustic field requires measurements of the same room at many source and receiver positions, and far fewer datasets provide such coverage.

Since RIRs require specific equipment and measurement processes that are difficult to crowdsource, it is unclear how much scale-up will be achieved in the near future. Because of this, recent few-shot acoustic field learning relies on synthetic data generated with room acoustics simulators [147], and adapts models trained on such large simulated corpora to real-world measurements [149, 284].

When the synthetic data is reliable, i.e., it exactly matches the real-world distribution or at least covers it, then employing maximally expressive methods, e.g., a generative model with large transformers, could achieve the best performance. However, if a gap remains between the synthetic and the real distributions, an expressive model can only reproduce what it was shown, and the prior imposed by a structured proxy becomes the more reliable source of the missing information.

Music production data is limited in a different way. Finished mixes are abundant, but the processor graphs and parameter settings that produced them are not distributed with the audio, so no dataset pairs a reference mix with a verified graph. The blind graph estimator was therefore trained on synthetically generated graphs and their rendered audio, and the pruning method offers one route to collecting such pairs from existing multitrack mixes without manual annotation.

Future work should therefore evaluate this gap explicitly and investigate how measured data, structured forward models, synthetic supervision, and neural components should be combined as the available data regime changes.

8.2.3 Objectives for Optimization and Evaluation

Both differentiable processing and generative modeling depend on optimization objectives, although the objectives play different roles. In differentiable processing, an audio-domain loss directly determines the gradients used to estimate the parameters of an explicit forward model. In generative modeling, token prediction, parameter regression, likelihood, and reconstruction objectives determine which distribution over system descriptions is learned. In both cases, the objective determines which errors and solutions the model prioritizes.

Objective design is particularly important for differentiable processing because the forward model, its parameterization, and the loss jointly determine the optimization landscape. Common waveform and spectrogram losses compare aligned samples, frames, or frequency bins. They may therefore respond strongly to a small temporal displacement and provide poorly informative or oscillatory gradients for parameters such as frequencies and delays. With a suitable continuous parameterization, a single sinusoidal frequency or delay can be recovered by gradient descent, but mixtures of sinusoids or delayed components remain substantially more difficult because their contributions interact and create competing assignments and local minima [260, 58, 92]. Revised multi-scale spectral objectives and spectral optimal-transport losses attempt to provide more useful gradients for models containing multiple spectral components [60, 59]. Their effectiveness nevertheless depends on the system being optimized, and empirical comparisons show that the relative performance of audio objectives changes with the underlying synthesis architecture [285]. Differentiable audio processing should therefore be understood as the joint design of processors, parameterizations, and objectives, rather than only the implementation of an existing processor in an automatic-differentiation framework.

A useful optimization landscape is only one requirement. The objective should also assign meaningful costs to perceptually relevant differences. Distances based on learned or pretrained audio representations offer one approach, while fixed multiscale or wavelet representations provide a distinct way to describe temporal and spectrotemporal structure without learning a perceptual model [286, 261, 287, 288]. A metric used only for evaluation does not need to provide useful gradients, but defining one that reflects perceived similarity and the practical usefulness of an estimated system remains difficult. Training objectives and evaluation metrics should therefore be selected and validated separately.

Part of the prior knowledge can also be moved from the forward model into the objective, penalizing violations of governing equations or constraints where the observations are sparse [289, 290].

8.2.4 Alternative Estimation Strategies

The methods developed in this thesis are not the only approaches to audio system estimation. Gradient-free inference-time optimization can operate directly on non-differentiable effect chains using a learned similarity objective, while reinforcement learning can tune black-box synthesizers using audio similarity as a reward [291, 292]. These methods broaden the range of processors and objectives that can be used, but their query cost can be high and their performance still depends on the chosen objective. A learned model can also propose an initial configuration that such a search then refines [293].

The generative side offers alternatives of the same kind. Both generators in this thesis factorized a sequence of discrete tokens autoregressively, but a distribution over a discrete description can also be learned by discrete diffusion [294, 295], which has a counterpart for graph generation [296]. Autoregressive factorization has also been extended to continuous frames, removing the resolution ceiling that a codec imposes instead of working around it with a smaller token budget [297, 298].

The two hybrid extensions of this thesis combined differentiable processing and generative modeling in two specific ways, and neither exhausts the possibilities. Blind restoration methods place a generative prior on the unprocessed source and keep a differentiable model of the processors, estimating the source and the system together under that prior [299, 300]. Which of these strategies is preferable, and under what conditions, remains unclear.

8.2.5 System-Aware Representation Learning

Any neural network used to solve an inverse problem forms an explicit or implicit representation of the observed system. The representation is explicit when the network produces RIR tokens, acoustic parameters, or a processing graph, and implicit when the relevant information remains in an intermediate embedding. This suggests treating audio system estimation as a representation-learning problem and using individual inverse problems as downstream tasks.

Most general-purpose audio representations are trained primarily for content-oriented tasks and are not designed specifically to preserve the transformation applied to a signal [301, 302]. Processing-aware representation learning has begun

to address this gap [264, 291, 303, 304, 305], but remains at an early stage. The same idea has developed independently in room acoustics, where embeddings are learned to describe the acoustic environment [306, 307, 308, 309, 284], and one recent method uses such an embedding as the objective for fitting a differentiable artificial reverberator [310]. Training examples can expose the representation to the same system applied to different source signals and to different systems applied to related source material. The aim is to preserve information about the system while reducing irrelevant variation in the content, although complete content invariance is inappropriate when a processor responds to the input itself.

8.2.6 Audio Post-Production Workflows and Deployment

All the system estimation tasks tackled in this thesis are tightly related to audio post-production applications, so the value of an estimation depends on how it will be used. Take room acoustics as an example: artificial reverberator parameters and surface material properties expose different forms of control, while a free-form RIR can be reused directly for convolution but requires some post-processing for semantic controls. Learned tokens and latent variables can improve coverage or reduce inference cost, but they are not automatically meaningful editing parameters.

The available computation also depends on the workflow. Offline room calibration or mix analysis can support iterative optimization and repeated rendering, whereas interactive production and real-time effects require lower latency and predictable memory use. A method may run inside a digital audio workstation or plugin, as a standalone application, or through a remote service with different computational resources. Inference time, rendering cost, and memory use are therefore as much a part of a method’s usefulness as its estimation quality.

Some of the recent post-production tools take a different route: rather than estimating a system description and rendering it, a generative signal model produces the processed audio directly. In this thesis, the estimated target was always a description of the system itself, and the methods differed only in how that description was predicted. Generating the audio may be sufficient when only the result matters, but structured system descriptions remain useful when practitioners need

to inspect, edit, reuse, automate, or audit the transformation.

Audio system estimation therefore requires two separate decisions: which description of the system to commit to, and how that description should be estimated. Differentiable processing and generative modeling widen the second choice without removing the first, and whichever combination is chosen, the estimated description should remain meaningful to whoever uses it.

References

- [1] Jari P Kaipio and Erkki Somersalo. *Statistical and computational inverse problems*. Springer, 2005.
- [2] Andrew M. Stuart. Inverse problems: A Bayesian perspective. *Acta Numerica*, 19:451–559, 2010.
- [3] Julius O. Smith. *Introduction to Digital Filters with Audio Applications*. W3K Publishing, 2007.
- [4] Julius O. Smith. *Mathematics of the Discrete Fourier Transform (DFT)*. W3K Publishing, 2007.
- [5] Vadim Zavalishin. *The Art of VA Filter Design*. Native Instruments, 2020. Revision 2.1.2.
- [6] Shahan Nercessian, Andy Sarroff, and Kurt James Werner. Lightweight and interpretable neural modeling of an audio distortion effect using hyperconditioned differentiable biquads. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 890–894, 2021.
- [7] Alistair Carson, Simon King, Cassia Valentini Botinhao, and Stefan Bilbao. Differentiable grey-box modelling of phaser effects using frame-based spectral processing. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, 2023.
- [8] Julius O. Smith. *Spectral Audio Signal Processing*. W3K Publishing, 2011.
- [9] Udo Zölzer, editor. *DAFX: Digital Audio Effects*. John Wiley & Sons, 2nd edition, 2011.

- [10] Dimitrios Giannoulis, Michael Massberg, and Joshua D Reiss. Digital dynamic range compressor design—a tutorial and analysis. *Journal of the Audio Engineering Society (JAES)*, 60(6):399–408, 2012.
- [11] S.A. Billings and S.Y. Fakhouri. Identification of systems containing linear dynamic and static nonlinear elements. *Automatica*, 18(1):15–26, 1982.
- [12] Martin Schetzen. Nonlinear system modeling based on the wiener theory. *Proceedings of the IEEE*, 69(12):1557–1573, 1981.
- [13] Christian J Steinmetz, Thomas Walther, and Joshua D Reiss. High-fidelity noise reduction with differentiable signal processing. In *Audio Engineering Society Convention 155*, 2023.
- [14] Christian J. Steinmetz, Jordi Pons, Santiago Pascual, and Joan Serrà. Automatic multitrack mixing with a differentiable mixing console of neural audio effects. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 71–75, 2021.
- [15] Brecht De Man, Joshua Reiss, and Ryan Stables. Ten years of automatic mixing. In *3rd Workshop on Intelligent Music Production*, 2017.
- [16] Marco A. Martínez Ramírez, Emmanouil Benetos, and Joshua D. Reiss. Deep learning for black-box modeling of audio effects. *Applied Sciences*, 10(2):638, January 2020.
- [17] Christian J Steinmetz and Joshua D Reiss. Efficient neural networks for real-time modeling of analog dynamic range compression. In *152nd Audio Engineering Society Convention (AES)*, 2022. Convention Paper 10596.
- [18] Lawrence E Kinsler, Austin R Frey, Alan B Coppens, and James V Sanders. *Fundamentals of acoustics*. John wiley & sons, 2000.
- [19] Thomas Funkhouser, Patrick Min, and Ingrid Carlbom. Real-time acoustic modeling for distributed virtual environments. In *26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 365–374, 1999.

- [20] Sebastia Vicenc Amengual Gari, Carl Schissler, and Philip Robinson. Perceptual comparison of efficient real-time geometrical acoustics engines in virtual reality. In *Audio Engineering Society International Conference on Audio for Games (AES)*. Audio Eng. Soc., 2024.
- [21] Lukas Aspöck, Sönke Pelzer, Frank Wefers, and Michael Vorländer. A real-time auralization plugin for architectural design and education. In *EAA Joint Symposium on Auralization and Ambisonics*, pages 156–161, 2014.
- [22] Vesa Valimäki, Julian D Parker, Lauri Savioja, Julius O Smith, and Jonathan S Abel. Fifty years of artificial reverberation. *IEEE Transactions on Audio, Speech, and Language Processing (TASLPRO)*, 20(5):1421–1448, 2012.
- [23] Vesa Välimäki, Julian Parker, Lauri Savioja, Julius O Smith, and Jonathan Abel. More than 50 years of artificial reverberation. In *Proceedings in 60th AES International Conferences on DREAMS (Dereverberation Reverberation Audio, Music, Speech)*, pages 1–12, 2016.
- [24] Jon Dattorro. Effect design, part 1: reverberator and other filters. *Journal of the Audio Engineering Society (JAES)*, 45(9):660–684, 1997.
- [25] Samuel Siltanen, Tapio Lokki, Sami Kiminki, and Lauri Savioja. The room acoustic rendering equation. *The Journal of the Acoustical Society of America (JASA)*, 122(3):1624–1635, 2007.
- [26] Lauri Savioja and U Peter Svensson. Overview of geometrical room acoustic modeling techniques. *The Journal of the Acoustical Society of America (JASA)*, 138(2):708–730, 2015.
- [27] Andrew Luo, Yilun Du, Michael Tarr, Josh Tenenbaum, Antonio Torralba, and Chuang Gan. Learning neural acoustic fields. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 3165–3177, 2022.
- [28] Mason Long Wang, Ryosuke Sawata, Samuel Clarke, Ruohan Gao, Shangzhe Wu, and Jiajun Wu. Hearing anything anywhere. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11790–11799, 2024.

- [29] Daniele Barchiesi and Joshua Reiss. Reverse engineering of a mix. *Journal of the Audio Engineering Society (JAES)*, 58(7/8):563–576, 2010.
- [30] Joseph T. Colonel and Joshua Reiss. Reverse engineering of a recording mix with differentiable digital signal processing. *The Journal of the Acoustical Society of America*, 150(1):608–619, 2021.
- [31] Marco A Martínez-Ramírez, Wei-Hsiang Liao, Giorgio Fabbro, Stefan Uhlich, Chihiro Nagashima, and Yuki Mitsufuji. Automatic music mixing with deep learning and out-of-domain data. In *23rd International Society for Music Information Retrieval Conference (ISMIR)*, pages 411–418, 2022.
- [32] Alex Wilson and Bruno Fazenda. 101 mixes: A statistical analysis of mix-variation in a dataset of multi-track music mixes. In *Audio Engineering Society Convention 139*. Audio Engineering Society, 2015.
- [33] Angeliki Mourgela, Elio Quinton, Spyridon Bissas, Joshua D Reiss, and David Ronan. Exploring trends in audio mixes and masters: Insights from a dataset analysis. In *157th Audio Engineering Society Convention (AES)*, 2024. Convention Paper 10186.
- [34] Simon Shelley and D.T. Murphy. Openair: An interactive auralization web resource and database. *129th Audio Engineering Society Convention 2010*, 2:1270–1278, 01 2010.
- [35] J. Eaton, N. D. Gaubitch, A. H. Moore, and P. A. Naylor. Estimation of room acoustic parameters: The ace challenge. *IEEE/ACM Transactions on Audio, Speech, and Language Processing (TASLP)*, 24(10):1681–1693, 2016.
- [36] L. Rabiner, B. Gold, and C. McGonegal. An approach to the approximation problem for nonrecursive digital filters. *IEEE Transactions on Audio and Electroacoustics*, 18(2):83–106, 1970.
- [37] Shahan Nercessian. Neural parametric equalizer matching using differentiable biquads. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, pages 265–272, 2020.

- [38] Boris Kuznetsov, Julian D Parker, and Fabián Esqueda. Differentiable iir filters for machine learning applications. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, pages 297–303, 2020.
- [39] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.
- [40] Bernard Widrow and Marcian E. Hoff, Jr. Adaptive switching circuits. In *IRE WESCON Convention Record*, volume 4, pages 96–104, 1960.
- [41] C. Richard Johnson, Jr. Adaptive IIR filtering: Current results and open issues. *IEEE Transactions on Information Theory*, 30(2):237–250, 1984.
- [42] John J. Shynk. Adaptive IIR filtering. *IEEE ASSP Magazine*, 6(2):4–21, 1989.
- [43] David A. Johns, W. Martin Snelgrove, and Adel S. Sedra. Adaptive recursive state-space filters using a gradient-based algorithm. *IEEE Transactions on Circuits and Systems*, 37(6):673–684, 1990.
- [44] F.X.Y. Gao and W.M. Snelgrove. An adaptive backpropagation cascade IIR filter. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 39(9):606–610, 1992.
- [45] A. D. Back and A. C. Tsoi. FIR and IIR synapses, a new neural network architecture for time series modeling. *Neural Computation*, 3(3):375–385, 1991.
- [46] P. Campolucci, A. Uncini, and F. Piazza. Fast adaptive IIR-MLP neural networks for signal processing applications. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 6, pages 3529–3532 vol. 6, 1996.
- [47] Jesse Engel, Lamtharn (Hanoi) Hantrakul, Chenjie Gu, and Adam Roberts. DDSP: differentiable digital signal processing. In *International Conference on Learning Representations (ICLR)*, 2020.

- [48] Xavier Serra and J. Smith. Spectral modeling synthesis: A sound analysis/synthesis based on a deterministic plus stochastic decomposition. *Computer Music Journal*, 14:12–24, 1990.
- [49] Ben Hayes, Jordie Shier, György Fazekas, Andrew McPherson, and Charalampos Saitis. A review of differentiable digital signal processing for music and speech synthesis. *Frontiers in Signal Processing*, 3:1284100, 2024.
- [50] Chin-Yun Yu, Christopher Mitcheltree, Alistair Carson, Stefan Bilbao, Joshua D Reiss, and György Fazekas. Differentiable all-pole filters for time-varying audio systems. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, pages 345–352, 2024.
- [51] Mario Lezcano-Casado and David Martínez-Rubio. Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In *International Conference on Machine Learning (ICML)*, pages 3794–3803, 2019.
- [52] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-softmax. In *International Conference on Learning Representations (ICLR)*, 2017.
- [53] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations (ICLR)*, 2017.
- [54] Sungho Lee, Marco A Martínez-Ramírez, Wei-Hsiang Liao, Stefan Uhlich, Giorgio Fabbro, Kyogu Lee, and Yuki Mitsufuji. Searching for music mixing graphs: a pruning approach. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, 2024.
- [55] James C Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 37(3):332–341, 1992.
- [56] Marco A Martínez-Ramírez, Oliver Wang, Paris Smaragdis, and Nicholas J Bryan. Differentiable signal processing with black-box audio effects. In *IEEE*

International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2021.

- [57] Christian J Steinmetz, Nicholas J Bryan, and Joshua D Reiss. Style transfer of audio effects with differentiable signal processing. *Journal of the Audio Engineering Society (JAES)*, 70(9):708–721, 2022.
- [58] Ben Hayes, Charalampos Saitis, and György Fazekas. Sinusoidal frequency estimation by gradient descent. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2023.
- [59] Bernardo Torres, Geoffroy Peeters, and Gaël Richard. Unsupervised harmonic parameter estimation using differentiable dsp and spectral optimal transport. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1176–1180. IEEE, 2024.
- [60] Simon Schwär and Meinard Müller. Multi-scale spectral loss revisited. *IEEE Signal Processing Letters*, 30:1712–1716, 2023.
- [61] Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations (ICLR)*, 2014.
- [62] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning (ICML)*, pages 1530–1538. PMLR, 2015.
- [63] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. In *International Conference on Learning Representations (ICLR)*, 2017.
- [64] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*, volume 27, 2014.
- [65] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics.

- In *International Conference on Machine Learning (ICML)*, pages 2256–2265. PMLR, 2015.
- [66] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 6840–6851, 2020.
 - [67] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations (ICLR)*, 2021.
 - [68] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations (ICLR)*, 2023.
 - [69] Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International Conference on Machine Learning (ICML)*, pages 1747–1756. PMLR, 2016.
 - [70] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio. In *ISCA Speech Synthesis Workshop (SSW)*, page 125, 2016.
 - [71] Manfred R. Schroeder and Benjamin F. Logan. ‘colorless’ artificial reverberation. *Journal of the Audio Engineering Society (JAES)*, 9(3):192–197, July 1961.
 - [72] Jean-Marc Jot and Antoine Chaigne. Digital delay networks for designing artificial reverberators. In *90th Audio Engineering Society Convention (AES)*. Audio Engineering Society, 1991.
 - [73] Vesa Välimäki, Bo Holm-Rasmussen, Benoit Alary, and Heidi-Maria Lehtonen. Late reverberation synthesis using filtered velvet noise. *Applied Sciences*, 7(5):483, 2017.

- [74] J-M. Jot. An analysis/synthesis approach to real-time artificial reverberation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 2, pages 221–224, 1992.
- [75] Jay Coggin and Will Pirkle. Automatic design of feedback delay network reverb parameters for impulse response matching. In *141st Audio Engineering Society International Convention 2016, AES 2016*. Audio Engineering Society, 2016.
- [76] Nils Peters, Jaeyoung Choi, and Howard Lei. Matching artificial reverb settings to unknown room recordings: a recommendation system for reverb plugins. In *133rd Audio Engineering Society Convention (AES)*, October 2012.
- [77] Andy Sarroff and Roth Michaels. Blind arbitrary reverb matching. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, 2020.
- [78] James T Kajiya. The rendering equation. In *13th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 143–150, 1986.
- [79] Jont B Allen and David A Berkley. Image method for efficiently simulating small-room acoustics. *The Journal of the Acoustical Society of America (JASA)*, 65(4):943–950, 1979.
- [80] Kun Su, Mingfei Chen, and Eli Shlizerman. Inras: Implicit neural representation for audio scenes. *Advances in Neural Information Processing Systems*, 35:8144–8158, 2022.
- [81] Sungho Lee, Matteo Scerbo, Seungu Han, Min Jun Choi, Kyogu Lee, and Enzo De Sena. Differentiable acoustic radiance transfer. *IEEE Transactions on Audio, Speech, and Language Processing (TASLPRO)*, 2026.
- [82] Christian J. Steinmetz, Vamsi Krishna Ithapu, and Paul Calamia. Filtered noise shaping for time domain room impulse response estimation from reverberant speech. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*., pages 221–225, 2021.

- [83] Sungho Lee, Hyeong-Seok Choi, and Kyogu Lee. Differentiable artificial reverberation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing (TASLP)*, 30:2541–2556, 2022.
- [84] Anton Ratnarajah, Ishwarya Ananthabhotla, Vamsi Krishna Ithapu, Pablo Hoffmann, Dinesh Manocha, and Paul Calamia. Towards improved room impulse response estimation for speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.
- [85] Neil Zeghidour, Alejandro Luebs, Ahmed Omran, Jan Skoglund, and Marco Tagliasacchi. Soundstream: An end-to-end neural audio codec. *IEEE/ACM Transactions on Audio, Speech, and Language Processing (TASLP)*, 30:495–507, 2021.
- [86] Zálán Borsos, Raphaël Marinier, Damien Vincent, Eugene Kharitonov, Olivier Pietquin, Matt Sharifi, Dominik Roblek, Olivier Teboul, David Grangier, Marco Tagliasacchi, and Neil Zeghidour. AudioLM: A language modeling approach to audio generation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing (TASLP)*, 31:2523–2533, 2023.
- [87] Sanyuan Chen, Chengyi Wang, Yu Wu, Ziqiang Zhang, Long Zhou, Shujie Liu, Zhuo Chen, Yanqing Liu, Huaming Wang, Jinyu Li, Lei He, Sheng Zhao, and Furu Wei. Neural codec language models are zero-shot text to speech synthesizers. *IEEE Transactions on Audio, Speech and Language Processing (TASLPRO)*, 33:705–718, 2025.
- [88] Sungho Lee, Hyeong-Seok Choi, and Kyogu Lee. Yet another generative model for room impulse response estimation. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2023.
- [89] Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11523–11532, 2022.
- [90] Noy Uzrad, Oren Barkan, Almog Elharar, Shlomi Shvartzman, Moshe

- Laufer, Lior Wolf, and Noam Koenigstein. Diffmoog: a differentiable modular synthesizer for sound matching. *arXiv preprint arXiv:2401.12570*, 2024.
- [91] Sunggho Lee, Marco A Martínez-Ramírez, Wei-Hsiang Liao, Stefan Uhlich, Giorgio Fabbro, Kyogu Lee, and Yuki Mitsufuji. GRAFX: an open-source library for audio processing graphs in PyTorch. In *Proceedings of the International Conference on Digital Audio Effects (DAFx, demo)*, 2024.
- [92] Sunggho Lee, Marco A Martínez-Ramírez, Wei-Hsiang Liao, Stefan Uhlich, Giorgio Fabbro, Kyogu Lee, and Yuki Mitsufuji. Reverse engineering of music mixing graphs with differentiable processors and iterative pruning. *Journal of the Audio Engineering Society (JAES)*, 73(6):344–365, 2025.
- [93] Joseph T Colonel and Joshua Reiss. Reverse engineering a nonlinear mix of a multitrack recording. *Journal of the Audio Engineering Society (JAES)*, 71(9):586–595, 2023.
- [94] Giovanna Castellano, Anna Maria Fanelli, and Marcello Pelillo. An iterative pruning algorithm for feedforward neural networks. *IEEE transactions on Neural networks*, 8(3):519–531, 1997.
- [95] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018.
- [96] Jinwoo Kim, Dat Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. Pure transformers are powerful graph learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 14582–14595, 2022.
- [97] Christopher Mitcheltree, Christian J Steinmetz, Marco Comunità, and Joshua D Reiss. Modulation extraction for LFO-driven audio effects. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, 2023.
- [98] William G. Gardner. A real-time multichannel room simulator. *Journal of the Acoustical Society of America*, 92:2395–2395, 1992.

- [99] E. De Sena, H. Hacıhabiboğlu, Z. Cvetković, and J. O. Smith. Efficient synthesis of room acoustics via scattering delay networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing (TASLP)*, 23(9):1478–1492, 2015.
- [100] Natalie Agus, Hans Anderson, Jer-Ming Chen, Simon Lui, and Dorien Herremans. Minimally simple binaural room modeling using a single feedback delay network. *Journal of the Audio Engineering Society (JAES)*, 66(10):791–807, October 2018.
- [101] John Stautner and Miller Puckette. Designing multi-channel reverberators. *Computer Music Journal*, 6(1):52–65, 1982.
- [102] Michael Gerzon. Synthetic reverberation, part I. *Studio Sound*, 13:632–635, 1971.
- [103] International Organization for Standardization. Acoustics—measurement of room acoustic parameters—part 2: Reverberation time in ordinary rooms. International Standard ISO 3382-2:2008, International Organization for Standardization, Geneva, Switzerland, 2008.
- [104] Rama Ratnam, Douglas L. Jones, Bruce C. Wheeler, William D. O’Brien, Charissa R. Lansing, and Albert S. Feng. Blind estimation of reverberation time. *The Journal of the Acoustical Society of America*, 114(5):2877–2892, 2003.
- [105] Jimi YC Wen, Emanuël AP Habets, and Patrick A Naylor. Blind estimation of reverberation time based on the distribution of signal decay rates. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 329–332, 2008.
- [106] H. Gamper and I. J. Tashev. Blind reverberation time estimation using a convolutional neural network. In *16th International Workshop on Acoustic Signal Enhancement (IWAENC)*, pages 136–140, 2018.
- [107] S. Li, R. Schlieper, and J. Peissig. A hybrid method for blind estimation of frequency dependent reverberation time using speech signals. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 211–215, 2019.

- [108] Sebastian Schlecht and Emanuël Habets. Accurate reverberation time control in feedback delay networks. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, September 2017.
- [109] Karolina Prawda, Vesa Välimäki, and Sebastian Schlecht. Improved reverberation time control for feedback delay networks. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, September 2019.
- [110] Emmanouil T. Chourdakakis and Joshua D. Reiss. A machine-learning approach to application of intelligent artificial reverberation. *Journal of the Audio Engineering Society (JAES)*, 65(1/2):56–65, January 2017.
- [111] Alexander Richard, Dejan Markovic, Israel D Gebru, Steven Krenn, Gladstone Butler, Fernando de la Torre, and Yaser Sheikh. Neural synthesis of binaural speech from mono audio. In *International Conference on Learning Representations (ICLR)*, 2021.
- [112] Aaron Wishnick. Time-varying filters for musical applications. In *Proceedings of the 17th International Conference on Digital Audio Effects (DAFx)*, 2014.
- [113] Vesa Välimäki, Heidi-Maria Lehtonen, and Marko Takanen. A perceptual study on velvet noise and its variants at different pulse densities. *IEEE Transactions on Audio, Speech, and Language Processing (TASLPRO)*, 21(7):1481–1488, 2013.
- [114] Hanna Järveläinen and Matti Karjalainen. Reverberation modeling using velvet noise. In *30th Audio Engineering Society International Conference: Intelligent Audio Environments (AES)*, March 2007.
- [115] D. Rocchesso and J. O. Smith. Circulant and elliptic feedback delay networks for artificial reverberation. *IEEE Transactions on Speech and Audio Processing*, 5(1):51–63, 1997.
- [116] Sebastian J. Schlecht and Emanuël A. P. Habets. Time-varying feedback matrices in feedback delay networks and their application in artificial reverberation. *The Journal of the Acoustical Society of America*, 138(3):1389–1398, 2015.

- [117] Sebastian Schlecht. Fdntb: The feedback delay network toolbox. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, pages 211–218. DAFx, 2020.
- [118] Peter Svensson and Ulf R. Kristiansen. Computational modelling and simulation of acoutic spaces. In *22nd Audio Engineering Society International Conference: Virtual, Synthetic, and Entertainment Audio (AES)*, June 2002.
- [119] Robin Scheibler, Eric Bezzam, and Ivan Dokmanić. Pyroomacoustics: A python package for audio room simulation and array process. algorithms. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 351–355. IEEE, 2018.
- [120] Guillaume Defrance, Laurent Daudet, and Jean-Dominique Polack. Finding the onset of a room impulse response: straightforward? *The Journal of the Acoustical Society of America*, 124(4):EL248–EL254, 2008.
- [121] Nicholas J. Bryan. Impulse response data augmentation and deep neural networks for blind room acoustic parameter estimation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5, 2020.
- [122] Junichi Yamagishi, Christophe Veaux, and Kirsten MacDonald. CSTR VCTK corpus: English multi-speaker corpus for CSTR voice cloning toolkit (version 0.92), 2019.
- [123] Constant Hak, Remy Wenmaekers, and L. Luxemburg. Measuring room impulse responses: Impact of the decay range on derived room acoustic parameters. *Acta Acustica united with Acustica*, 98, 11 2012.
- [124] Z. Meng, F. Zhao, and M. He. The just noticeable difference of noise length and reverberation perception. In *International Symposium on Communications and Information Technologies*, pages 418–421, 2006.
- [125] Matthew G Blevins, Adam T Buck, Z. Ellen Peng, and Lily M Wang. Quantifying the just noticeable difference of reverberation time with band-limited

noise centered around 1000 Hz using a transformed up-down adaptive method. In *International Symposium on Room Acoustics (ISRA)*, 2013.

- [126] J.S. Bradley, R. Reich, and S.G. Norcross. A just noticeable difference in c50 for speech. *Applied Acoustics*, 58(2):99 – 108, 1999.
- [127] Francesco Martellotta. The just noticeable difference of center time and clarity index in large reverberant spaces. *The Journal of the Acoustical Society of America*, 128:654–63, 08 2010.
- [128] Erik Larsen, Nandini Iyer, Charissa Lansing, and Albert Feng. On the minimum audible difference in direct-to-reverberant energy ratio. *The Journal of the Acoustical Society of America*, 124:450–61, 07 2008.
- [129] International Telecommunication Union. Method for the subjective assessment of intermediate quality level of audio systems. Recommendation ITU-R BS.1534-3, International Telecommunication Union, Geneva, Switzerland, 2015.
- [130] Michael Schoeffler, Sarah Bartoschek, Fabian-Robert Stöter, Marlene Roess, Susanne Westphal, Bernd Edler, and Jürgen Herre. webMUSHRA—a comprehensive framework for web-based listening tests. *Journal of Open Research Software*, 6(1):8, 2018.
- [131] Juho Liski, Balazs Bank, Julius Smith, and Vesa Välimäki. Converting series biquad filters into delayed parallel form: Application to graphic equalizers. *IEEE Transactions on Signal Processing*, PP:1–1, 05 2019.
- [132] Natsuki Ueno, Shoichi Koyama, and Hiroshi Saruwatari. Kernel ridge regression with constraint of helmholtz equation for sound field interpolation. In *International Workshop on Acoustic Signal Enhancement (IWAENC)*, pages 1–440. IEEE, 2018.
- [133] Federico Miotello, Luca Comanducci, Mirco Pezzoli, Alberto Bernardini, Fabio Antonacci, and Augusto Sarti. Reconstruction of sound field through diffusion models. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1476–1480. IEEE, 2024.

- [134] Stefano Damiano, Federico Borra, Alberto Bernardini, Fabio Antonacci, and Augusto Sarti. A compressive sensing approach for the reconstruction of the soundfield produced by directive sources in reverberant rooms. *IEEE/ACM Transactions on Audio, Speech, and Language Processing (TASLP)*, 32:2667–2679, 2024.
- [135] Juliano GC Ribeiro, Shoichi Koyama, Ryosuke Horiuchi, and Hiroshi Saruwatari. Sound field estimation based on physics-constrained kernel interpolation adapted to environment. *IEEE/ACM Transactions on Audio, Speech, and Language Processing (TASLP)*, 2024.
- [136] Niccolo Antonello, Enzo De Sena, Marc Moonen, Patrick A Naylor, and Toon Van Waterschoot. Room impulse response interpolation using a sparse spatio-temporal representation of the sound field. *IEEE/ACM Transactions on Audio, Speech, and Language Processing (TASLP)*, 25(10):1929–1941, 2017.
- [137] Jesper Brunnström, Martin Bo Møller, Jan Østergaard, Shoichi Koyama, Toon van Waterschoot, and Marc Moonen. Time-domain sound field estimation using kernel ridge regression. *IEEE Transactions on Audio, Speech and Language Processing (TASLPRO)*, 34:1243–1258, 2026.
- [138] Samuel A Verburg, Efren Fernandez-Grande, and Peter Gerstoft. Differentiable physics for sound field reconstruction. *The Journal of the Acoustical Society of America (JASA)*, 158(5):4059–4069, 2025.
- [139] Antonio Figueroa-Duran and Efren Fernandez-Grande. Reconstruction of reverberant sound fields over large spatial domains. *The Journal of the Acoustical Society of America (JASA)*, 157(1):180–190, 2025.
- [140] Sagnik Majumder, Changan Chen, Ziad Al-Halah, and Kristen Grauman. Few-shot audio-visual learning of environment acoustics. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 2522–2536, 2022.
- [141] Yuhang He, Anoop Cherian, Gordon Wichern, and Andrew Markham. Deep neural room acoustics primitive. In *International Conference on Machine Learning (ICML)*, 2024.

- [142] Zitong Lan, Chenhao Zheng, Zhiwei Zheng, and Mingmin Zhao. Acoustic volume rendering for neural impulse response fields. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 44600–44623, 2024.
- [143] Changan Chen, Alexander Richard, Roman Shapovalov, Vamsi Krishna Ithapu, Natalia Neverova, Kristen Grauman, and Andrea Vedaldi. Novel-view acoustic synthesis. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6409–6419, 2023.
- [144] Mingfei Chen and Eli Shlizerman. AV-Cloud: Spatial audio rendering through audio-visual cloud splatting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.
- [145] Huiyu Gao, Jiahao Ma, David Ahmedt-Aristizabal, Chuong Nguyen, and Miaomiao Liu. SOAF: Scene occlusion-aware neural acoustic field. In *IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 4097–4107, 2026.
- [146] Ricardo Falcon-Perez, Ruohan Gao, Gregor Mueckl, Sebastia V Amengual Gari, and Ishwarya Ananthabhotla. Scene-wide acoustic parameter estimation. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2025.
- [147] Changan Chen, Unnat Jain, Carl Schissler, Sebastia Vicenc Amengual Gari, Ziad Al-Halah, Vamsi Krishna Ithapu, Philip Robinson, and Kristen Grauman. Soundspaces: Audio-visual navigation in 3d environments. In *ECCV*, 2020.
- [148] Ziyang Chen, Israel D Gebru, Christian Richardt, Anurag Kumar, William Laney, Andrew Owens, and Alexander Richard. Real acoustic fields: An audio-visual room acoustics dataset and benchmark. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21886–21896, 2024.
- [149] Xiulong Liu, Anurag Kumar, Paul Calamia, Sebastià V. Amengual, Calvin Murdock, Ishwarya Ananthabhotla, Philip Robinson, Eli Shlizerman,

- Vamsi Krishna Ithapu, and Ruohan Gao. Hearing anywhere in any environment. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5732–5741, 2025.
- [150] Susan Liang, Chao Huang, Yapeng Tian, Anurag Kumar, and Chenliang Xu. AV-NeRF: Learning neural fields for real-world audio-visual scene synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 37472–37490, 2023.
- [151] Amandine Brunetto, Sascha Hornauer, and Fabien Moutarde. NeRAF: 3d scene infused neural radiance and acoustic fields. In *International Conference on Learning Representations (ICLR)*, 2025.
- [152] Swapnil Bhosale, Haosen Yang, Diptesh Kanojia, Jiankang Deng, and Xiaotian Zhu. AV-GS: Learning material and geometry aware priors for novel view acoustic synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 28920–28937, 2024.
- [153] Anton Ratnarajah, Zhenyu Tang, Rohith Aralikatti, and Dinesh Manocha. Mesh2ir: Neural acoustic impulse response generator for complex 3d scenes. In *ACM International Conference on Multimedia (ACM MM)*, pages 924–933, 2022.
- [154] Zitong Lan, Yiduo Hao, and Mingmin Zhao. Resounding acoustic fields with reciprocity. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.
- [155] Niccoló Antonello, Toon van Waterschoot, Marc Moonen, and Patrick A Naylor. Identification of surface acoustic impedances in a reverberant room using the FDTD method. In *International Workshop on Acoustic Signal Enhancement (IWAENC)*, pages 114–118. IEEE, 2014.
- [156] Bowen Zhi, Alisha Sharma, Dmitry N Zotkin, and Ramani Duraiswami. A differentiable image source model for room acoustics optimization. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2023.

- [157] Carl Schissler, Christian Loftin, and Dinesh Manocha. Acoustic classification and optimization for multi-modal rendering of real-world scenes. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 24(3):1246–1259, 2018.
- [158] Dingzeyu Li, Timothy R Langlois, and Changxi Zheng. Scene-aware audio for 360 videos. *ACM Transactions on Graphics (TOG)*, 37(4):1–12, 2018.
- [159] Zhenyu Tang, Nicholas J Bryan, Dingzeyu Li, Timothy R Langlois, and Dinesh Manocha. Scene-aware audio rendering via deep acoustic analysis. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 26(5):1991–2001, 2020.
- [160] Derong Jin and Ruohan Gao. Differentiable room acoustic rendering with multi-view vision priors. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.
- [161] Susan Liang, Chao Huang, Yunlong Tang, Zeliang Zhang, and Chenliang Xu. p-AVAS: Can physics-integrated audio-visual modeling boost neural acoustic synthesis? In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 13942–13951, 2025.
- [162] Hequn Bai, Gael Richard, and Laurent Daudet. Modeling early reflections of room impulse responses using a radiance transfer method. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 1–4, 2013.
- [163] Hequn Bai, Gael Richard, and Laurent Daudet. Geometric-based reverberator using acoustic rendering networks. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2015.
- [164] Matteo Scerbo, Lauri Savioja, and Enzo De Sena. Room acoustic rendering networks with control of scattering and early reflections. *IEEE/ACM Transactions on Audio, Speech, and Language Processing (TASLP)*, 32:3745–3758, 2024.

- [165] Matteo Scerbo, Sebastian J. Schlecht, Randall Ali, Lauri Savioja, and Enzo De Sena. Modeling nonuniform energy decay through the modal decomposition of acoustic radiance transfer (MoD-ART). *IEEE Transactions on Audio, Speech and Language Processing (TASLPRO)*, 33:3363–3376, 2025.
- [166] Samuel Siltanen, Tapio Lokki, and Lauri Savioja. Frequency domain acoustic radiance transfer for real-time auralization. *Acta Acustica united with Acustica*, 95(1):106–117, 2009.
- [167] Samuel Siltanen, Tapio Lokki, and Lauri Savioja. Efficient acoustic radiance transfer method with time-dependent reflections. *Proceedings of Meetings on Acoustics (POMA)*, 2011.
- [168] Lauri Savioja. Room acoustics modeling with interactive visualizations. <https://interactiveacoustics.info/>, 2016.
- [169] João Fatela, Anne Heimes, and Michael Vorländer. Efficient acoustic radiosity modelling for precise bi-directional reflectance distribution patterns. In *Proceedings of DAS/DAGA*, pages 1564–1567, 2025.
- [170] Fred E Nicodemus. Directional reflectance and emissivity of an opaque surface. *App. optics*, 4(7):767–775, 1965.
- [171] Frederick O Bartell, Eustace L Dereniak, and William L Wolfe. The theory and measurement of bidirectional reflectance distribution function (BRDF) and bidirectional transmittance distribution function (btddf). In *Radiation scattering in optical systems*, volume 257, pages 154–160. SPIE, 1981.
- [172] K Heinrich Kuttruff. Auralization of impulse responses modeled on the basis of ray-tracing results. *Journal of the Audio Engineering Society (JAES)*, 41(11):876–880, 1993.
- [173] Gloria Dal Santo, Benoit Alary, Karolina Prawda, Sebastian Schlecht, and Vesa Välimäki. Rir2fdn: An improved room impulse response analysis and synthesis. In *International Conference on Digital Audio Effects (DAFx)*, pages 230–237. University of Surrey, 2024.

- [174] Gloria Dal Santo, Karolina Prawda, Sebastian Schlecht, and Vesa Välimäki. Differentiable feedback delay network for colorless reverberation. In *International Conference on Digital Audio Effects (DAFx)*, pages 244–251. Aalborg University, 2023.
- [175] Alessandro Ilic Mezza, Riccardo Giampiccolo, Enzo De Sena, and Alberto Bernardini. Data-driven room acoustic modeling via differentiable feedback delay networks with learnable delay lines. *EURASIP Journal on Audio, Speech, and Music Processing*, 2024(1):51, 2024.
- [176] Alessandro Ilic Mezza, Riccardo Giampiccolo, Enzo De Sena, and Alberto Bernardini. Differentiable scattering delay networks for artificial reverberation. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, pages 202–207, 2025.
- [177] Julius O Smith. *Mathematics of the discrete Fourier transform (DFT): with audio applications*. Julius Smith, 2007.
- [178] Timo I Laakso, Vesa Valimäki, Matti Karjalainen, and Unto K Laine. Splitting the unit delay [fir/all pass filters design]. *IEEE Signal Processing Magazine (SPM)*, 13(1):30–60, 1996.
- [179] Bruce Walter, Stephen R Marschner, Hongsong Li, and Kenneth E Torrance. Microfacet models for refraction through rough surfaces. *Rendering techniques*, 2007:18th, 2007.
- [180] Heinrich Kuttruff. *Room acoustics*. Crc Press, 2016.
- [181] Gloria Dal Santo, Gian Marco De Bortoli, Karolina Prawda, Sebastian J Schlecht, and Vesa Välimäki. FLAMO: An open-source library for frequency-domain differentiable audio processing. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2025.
- [182] Thomas McKenzie, Sebastian J. Schlecht, and Ville Pulkki. Acoustic analysis and dataset of transitions between coupled rooms. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 481–485, 2021.

- [183] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.
- [184] Miles Macklin. Warp: A high-performance python framework for gpu simulation and graphics. <https://github.com/nvidia/warp>, March 2022. NVIDIA GPU Technology Conf. (GTC).
- [185] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.
- [186] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM (CACM)*, 65(1):99–106, 2021.
- [187] Samuel Siltanen, Tapio Lokki, Lauri Savioja, and Claus Lynge Christensen. Geometry reduction in room acoustics modeling. *Acta Acustica united with Acustica*, 94(3):410–418, 2008.
- [188] Chunxiao Cao, Zhong Ren, Carl Schissler, Dinesh Manocha, and Kun Zhou. Interactive sound propagation with bidirectional path tracing. *ACM Transactions on Graphics (TOG)*, 35(6):1–11, 2016.
- [189] Jan Kautz and Michael D McCool. Interactive rendering with arbitrary BRDFs using separable approximations. In *Eurographics Workshop on Rendering (EGWR)*, pages 247–260. Springer, 1999.
- [190] Samuel Siltanen and Tapio Lokki. Diffraction modeling in acoustic radiance transfer method. *The Journal of the Acoustical Society of America (JASA)*, 123(5):3759, 2008.

- [191] Michael Monks, Byong Mok Oh, and Julie Dorsey. Audiooptimization: goal-based acoustic design. *IEEE Computer Graphics and Applications (CG&A)*, 20(3):76–90, 2000.
- [192] Saeed Hadadan and Matthias Zwicker. Differentiable neural radiosity. *arXiv preprint arXiv:2201.13190*, 2022.
- [193] Saeed Hadadan, Geng Lin, Jan Novák, Fabrice Rousselle, and Matthias Zwicker. Inverse global illumination using a neural radiometric prior. In *ACM SIGGRAPH Conference Proceedings*, pages 1–11, 2023.
- [194] James Traer and Josh H. McDermott. Statistics of natural reverberation enable perceptual separation of sound and space. *Proceedings of the National Academy of Sciences (PNAS)*, 113(48):E7856–E7865, 2016.
- [195] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [196] Sebastian Braun and Ivan Tashev. A consolidated view of loss functions for supervised deep learning-based speech enhancement. In *44th International Conference on Telecommunications and Signal Processing (TSP)*, pages 72–76, 2021.
- [197] Aaron Van Den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems (NIPS)*, volume 30, 2017.
- [198] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12873–12883, 2021.
- [199] Biing-Hwang Juang and A Gray. Multiple stage vector quantization for speech coding. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 7, pages 597–600, 1982.
- [200] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer

normalization in the transformer architecture. In *International Conference on Machine Learning*, pages 10524–10533. PMLR, 2020.

- [201] Daniel Griffin and Jae Lim. Signal estimation from modified short-time fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing (TASSP)*, 32(2):236–243, 1984.
- [202] Guochang Zhang, Libiao Yu, Chunliang Wang, and Jianqiang Wei. Multi-scale temporal frequency convolutional network with axial attention for speech enhancement. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 9122–9126, 2022.
- [203] Marco Jeub, Magnus Schafer, and Peter Vary. A binaural room impulse response database for the evaluation of dereverberation algorithms. In *ICDSP*, 2009.
- [204] Igor Szöke, Miroslav Skácel, Ladislav Mošner, Jakub Paliesek, and Jan Černocký. Building and evaluation of a real room impulse response dataset. *IEEE Journal of Selected Topics in Signal Processing*, 13(4):863–876, 2019.
- [205] Rebecca Stewart and Mark Sandler. Database of omnidirectional and b-format room impulse responses. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2010.
- [206] Sebastia Vicenç Amengual Gari, Banu Sahin, Dusty Eddy, and Malte Kob. Open database of spatial room impulse responses at detmold university of music. In *149th Audio Engineering Society Convention (AES)*, 2020.
- [207] Masahiro Yasuda, Yasunori Ohishi, and Shoichiro Saito. Echo-aware adaptation of sound event localization and detection in unknown environments. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 226–230, 2022.
- [208] Gavin Kearney, Helena Daffern, Patrick Cairns, Anthony Hunt, Ben Lee, Jacob Cooper, Panos Tsagkarakis, Tomasz Rudzki, and Daniel Johnston. Measuring the acoustical properties of the bbc maida vale recording studios for virtual reality. *Acoustics*, 4(3):783–799, 2022.

- [209] T. Dietzen, R. Ali, M. Taseska, and T. van Waterschoot. MYriad: A multi-array room acoustic database. *EURASIP Journal on Audio, Speech, and Music Processing*, 2023, article no. 17:1–14, Apr. 2023.
- [210] Satoshi Nakamura, Kazuo Hiyane, Futoshi Asano, and Takashi Endo. Sound scene data collection in real acoustical environments. *Journal of the Acoustical Society of Japan (E)*, 20(3):225–231, 1999.
- [211] Juan Carlos Franco Hernández, Bogdan Bacila, Tim Brookes, and Enzo De Sena. A multi-angle, multi-distance dataset of microphone impulse responses. *Journal of the Audio Engineering Society (JAES)*, 70(10):882–893, 2022.
- [212] Gautham J. Mysore. Can we automatically transform speech recorded on common consumer devices in real-world environments into professional production quality speech?—a dataset, insights, and challenges. *IEEE Signal Processing Letters (SPL)*, 22(8):1006–1010, 2015.
- [213] Zhiyan Duan, Haotian Fang, Bo Li, Khe Chai Sim, and Ye Wang. The nus sung and spoken lyrics corpus: A quantitative comparison of singing and speech. In *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1–9. IEEE, 2013.
- [214] Gregor Pirker, Michael Wohlmayr, Stefan Petrik, and Franz Pernkopf. A pitch tracking corpus with evaluation on multipitch tracking scenario. In *12th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2011.
- [215] Ryosuke Sonobe, Shinnosuke Takamichi, and Hiroshi Saruwatari. Jsut corpus: free large-scale japanese speech corpus for end-to-end speech synthesis. *arXiv preprint arXiv:1711.00354*, 2017.
- [216] Rongjie Huang, Feiyang Chen, Yi Ren, Jinglin Liu, Chenye Cui, and Zhou Zhao. Multi-singer: Fast multi-singer singing voice vocoder with a large-scale corpus. In *ACM International Conference on Multimedia (ACM MM)*, pages 3945–3954, 2021.

- [217] Bidisha Sharma, Xiaoxue Gao, Karthika Vijayan, Xiaohai Tian, and Haizhou Li. Nhss: A speech and singing parallel database. *Speech Communication*, 133:9–22, 2021.
- [218] Itsuki Ogawa and Masanori Morise. Tohoku kiritan singing database: A singing database for statistical parametric singing synthesis using japanese pop songs. *Acoustical Science and Technology*, 42(3):140–145, 2021.
- [219] Clément Le Moine and Nicolas Obin. Att-hack: An expressive speech database with social attitudes. In *Speech Prosody*, pages 744–748, 2020.
- [220] Peter Kabal. Tsp speech database. *McGill University, Database Version*, 1(0):09–02, 2002.
- [221] Pascal Puchtler, Johannes Wirth, and René Peinl. Hui-audio-corpus-german: A high quality tts dataset. In *44th German Conference on Artificial Intelligence (KI)*, pages 204–216. Springer, 2021.
- [222] Keshan Sodimana, Knot Pipatsrisawat, Linne Ha, Martin Jansche, Oddur Kjartansson, Pasindu De Silva, and Supheakmongkol Sarin. A Step-by-Step Process for Building TTS Voices Using Open Source Data and Framework for Bangla, Javanese, Khmer, Nepali, Sinhala, and Sundanese. In *6th Workshop on Spoken Language Technologies for Under-Resourced Languages (SLTU)*, pages 66–70, Gurugram, India, August 2018.
- [223] Fei He, Shan-Hui Cathy Chu, Oddur Kjartansson, Clara Rivera, Anna Katanova, Alexander Gutkin, Isin Demirsahin, Cibu Johny, Martin Jansche, Supheakmongkol Sarin, and Knot Pipatsrisawat. Open-source multi-speaker speech corpora for building Gujarati, Kannada, Malayalam, Marathi, Tamil and Telugu speech synthesis systems. In *Proceedings of the Twelfth Language Resources and Evaluation Conference (LREC)*, pages 6494–6503, 2020.
- [224] Crowdsourced high-quality Nigerian English speech data set. OpenSLR resource 70, <https://www.openslr.org/70/>, 2020.

- [225] Jenthe Thienpondt and Kris Demuynck. ECAPA2: A hybrid neural network architecture and training strategy for robust speaker embeddings. In *IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2023.
- [226] Jiaqi Su, Zeyu Jin, and Adam Finkelstein. Acoustic matching by embedding impulse responses. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 426–430, 2020.
- [227] Junghyun Koo, Seungryeol Paik, and Kyogu Lee. Reverb conversion of mixed vocal tracks using an end-to-end convolutional deep neural network. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 81–85, 2021.
- [228] Sunggho Lee, Jaehyun Park, Seungryeol Paik, and Kyogu Lee. Blind estimation of audio processing graph. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2023.
- [229] Aric A Hagberg, Daniel A Schult, and Pieter J Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy)*, pages 11–15, 2008.
- [230] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with PyTorch Geometric. In *Workshop on Representation Learning on Graphs and Manifolds at the International Conference on Learning Representations (ICLR)*, 2019.
- [231] Pedro D Pestana and Joshua D Reiss. Intelligent audio production strategies informed by best practices. In *Audio Engineering Society 53rd International Conference on Semantic Audio*, 2014.
- [232] Flavio Everardo. Towards an automated multitrack mixing tool using answer set programming. In *14th Sound and Music Computing Conference*, 2017.
- [233] David Ronan, Hatice Gunes, and Joshua D Reiss. Analysis of the subgrouping practices of professional mix engineers. In *Audio Engineering Society Convention*, volume 142, 2017.

- [234] Rachel M Bittner, Justin Salamon, Mike Tierney, Matthias Mauch, Chris Cannam, and Juan Pablo Bello. MedleyDB: A multitrack dataset for annotation-intensive mir research. In *15th International Society for Music Information Retrieval Conference (ISMIR)*, volume 14, pages 155–160, 2014.
- [235] Rachel M Bittner, Julia Wilkins, Hanna Yip, and Juan P Bello. MedleyDB 2.0: New data and a system for sustainable data collection. In *17th International Society for Music Information Retrieval Conference (ISMIR), Late-Breaking/Demo Session*, page 36, 2016.
- [236] Mike Senior. *Mixing Secrets for the Small Studio*. Routledge, 2nd edition, 2018.
- [237] Alex Wilson and Bruno Fazenda. Variation in multitrack mixes: analysis of low-level audio signal features. *Journal of the Audio Engineering Society (JAES)*, 64(7/8):466–473, 2016.
- [238] Enrique Perez-Gonzalez and Joshua Reiss. Automatic gain and fader control for live mixing. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 1–4, 2009.
- [239] Brecht De Man and Joshua D Reiss. A knowledge-engineered autonomous mixing system. In *Audio Engineering Society Convention 135*, 2013.
- [240] Yang He and Lingao Xiao. Structured pruning for deep convolutional neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 46(5):2900–2919, 2024.
- [241] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.
- [242] Zhen Ye, Wei Xue, Xu Tan, Qifeng Liu, and Yike Guo. NAS-FM: Neural architecture search for tunable and interpretable sound synthesis based on frequency modulation. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 5869–5877, 2023.

- [243] Franco Caspe, Andrew McPherson, and Mark Sandler. DDX7: Differentiable FM synthesis of musical instrument sounds. In *23rd International Society for Music Information Retrieval Conference (ISMIR)*, 2022.
- [244] Christopher Mitcheltree and Hideki Koike. SerumRNN: Step by step audio VST effect programming. In *Artificial Intelligence in Music, Sound, Art and Design*, pages 218–234. Springer International Publishing, 2021.
- [245] Jinyue Guo and Brian McFee. Automatic recognition of cascaded guitar effects. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, 2023.
- [246] Osamu Take, Kento Watanabe, Takayuki Nakatsuka, Tian Cheng, Tomoyasu Nakano, Masataka Goto, Shinnosuke Takamichi, and Hiroshi Saruwatari. Audio effect chain estimation and dry signal recovery from multi-effect-processed musical signals. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, 2024.
- [247] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research (JMLR)*, 20(55):1–21, 2019.
- [248] Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.
- [249] Daniel C Elton, Zois Boukouvalas, Mark D Fuge, and Peter W Chung. Deep learning for molecular design—a review of the state of the art. *Molecular Systems Design & Engineering*, 4(4):828–849, 2019.
- [250] Daniel Y Fu, Hermann Kumbong, Eric Nguyen, and Christopher Ré. FlashFFTConv: Efficient convolutions for long sequences with tensor cores. *International Conference on Learning Representations (ICLR)*, 2023.
- [251] Ryuichi Yamamoto, Eunwoo Song, and Jae-Min Kim. Parallel wavegan: A fast waveform generation model based on generative adversarial networks

- with multi-resolution spectrogram. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 6199–6203, 2020.
- [252] Alec Wright and Vesa Välimäki. Perceptual loss function for neural modeling of audio systems. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 251–255, 2020.
- [253] Christian J Steinmetz and Joshua D Reiss. auraloss: Audio-focused loss functions in PyTorch. In *Digital Music Research Network One-Day Workshop (DMRN+15)*, 2020.
- [254] Soumya Vanka, Christian Steinmetz, Jean-Baptiste Rolland, Joshua Reiss, and György Fazekas. Diff-mst: Differentiable mixing style transfer. In *25th International Society for Music Information Retrieval Conference (ISMIR)*, San Francisco, USA, 2024. Int. Society for Music Information Retrieval (ISMIR).
- [255] Kevin Kilgour, Mauricio Zuluaga, Dominik Roblek, and Matthew Sharifi. Frèchet audio distance: A reference-free metric for evaluating music enhancement algorithms. In *Interspeech*, pages 2350–2354, 2019.
- [256] Jonathan Le Roux, Scott Wisdom, Hakan Erdogan, and John R Hershey. Sdr-half-baked or well done? In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 626–630. IEEE, 2019.
- [257] BD Man, Brett Leonard, Richard King, and Joshua D Reiss. An analysis and evaluation of audio features for multitrack music mixtures. In *15th International Society for Music Information Retrieval Conference (ISMIR)*, 2014.
- [258] Joseph T Colonel, Marco Comunità, and Joshua Reiss. Reverse engineering memoryless distortion effects with differentiable waveshapers. In *Audio Engineering Society Convention 153*, 2022.
- [259] David Ronan, Hatice Gunes, David Moffat, and Joshua D Reiss. Automatic subgrouping of multitrack audio. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, 2015.

- [260] Joseph Turian and Max Henry. I’m sorry for your loss: Spectrally-based audio distances are bad at pitch. In *“I Can’t Believe It’s Not Better!” Conference on Neural Information Processing Systems (NeurIPS) workshop*, 2020.
- [261] Cyrus Vahidi, Han Han, Changhong Wang, Mathieu Lagrange, György Fazekas, and Vincent Lostanlen. Mesostructures: Beyond spectrogram loss in differentiable time–frequency analysis. *Journal of the Audio Engineering Society (JAES)*, 2023.
- [262] Damian Koszewski, Thomas Görne, Grazina Korvel, and Bozena Kostek. Automatic music signal mixing system based on one-dimensional wave-u-net autoencoders. *EURASIP Journal on Audio, Speech, and Music Processing*, 2023(1):1–17, 2023.
- [263] Christian J. Steinmetz, Soumya Sai Vanka, Marco A Martínez-Ramírez, and Gary Bromham. *Deep Learning for Automatic Mixing*. ISMIR, December 2022.
- [264] Junghyun Koo, Marco A Martínez-Ramírez, Wei-Hsiang Liao, Stefan Uhlich, Kyogu Lee, and Yuki Mitsufuji. Music mixing style transfer: A contrastive learning approach to disentangle audio effects. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE, 2023.
- [265] Shawn Hershey, Sourish Chaudhuri, Daniel P. W. Ellis, Jort F. Gemmeke, Aren Jansen, R. Channing Moore, Manoj Plakal, Devin Platt, Rif A. Saurous, Bryan Seybold, Malcolm Slaney, Ron J. Weiss, and Kevin Wilson. Cnn architectures for large-scale audio classification. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 131–135. IEEE, 2017.
- [266] Julius O Smith. *Physical Audio Signal Processing for virtual musical instruments and digital audio effects*. W3K publishing, 2010.
- [267] Felix Eichas. *System Identification of Nonlinear Audio Circuits*. PhD thesis, Helmut Schmidt University, 2019.
- [268] Maho Nakata and Tomomi Shimazaki. Pubchemqc project: a large-scale first-

principles electronic structure database for data-driven chemistry. *Journal of chemical information and modeling*, 57(6):1300–1308, 2017.

- [269] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 1263–1272, 2017.
- [270] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1251–1258, 2017.
- [271] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.
- [272] Faezeh Faez, Yassaman Ommi, Mahdieh Soleymani Baghshah, and Hamid R Rabiee. Deep graph generators: A survey. *IEEE Access*, 9:106675–106702, 2021.
- [273] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. *Advances in neural information processing systems*, 32, 2019.
- [274] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 28877–28888. Curran Associates, Inc., 2021.
- [275] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations (ICLR)*, 2021.
- [276] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

- [277] Jon Gillick, Adam Roberts, Jesse Engel, Douglas Eck, and David Bamman. Learning to groove with inverse sequence transformations. In *International Conference on Machine Learning (ICML)*, 2019.
- [278] Jung Hyun Koo, Seungryeol Paik, and Kyogu Lee. End-to-end music remastering system using self-supervised and adversarial training. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4608–4612, 2022.
- [279] Spyridon Stasis, Nicholas Jillings, Sean Enderby, and Ryan Stables. Audio processing chain recommendation. In *Proceedings of the 20th International Conference on Digital Audio Effects (DAFx)*, 2017.
- [280] Ugo Finnendahl, Markus Worchel, Tobias Jüterbock, Daniel Wujecki, Fabian Brinkmann, Stefan Weinzierl, and Marc Alexa. Differentiable geometric acoustic path tracing using time-resolved path replay backpropagation. *ACM Transactions on Graphics*, 44(4):1–17, 2025.
- [281] Christopher Mitcheltree, Hao Hao Tan, and Joshua D. Reiss. Modulation discovery with differentiable digital signal processing. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 1–5, 2025.
- [282] Štěpán Miklánek, Alec Wright, Vesa Välimäki, and Jiří Schimmel. Neural grey-box guitar amplifier modelling with limited data. In *26th International Conference on Digital Audio Effects (DAFx)*, 2023.
- [283] Marco Comunità, Christian J. Steinmetz, and Joshua D. Reiss. Differentiable black-box and gray-box modeling of nonlinear audio effects. *Frontiers in Signal Processing*, 5, 2025.
- [284] Amandine Brunetto. Few-shot acoustic synthesis with multimodal flow matching. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15773–15783, 2026.
- [285] Amir Salimi, Abram Hindle, and Osmar R Zaiane. Evaluating sound simi-

- larity metrics for differentiable, iterative sound-matching. *IEEE Transactions on Audio, Speech, and Language Processing (TASLPRO)*, 33:4982–4994, 2025.
- [286] Pranay Manocha, Adam Finkelstein, Richard Zhang, Nicholas J Bryan, Gautham J Mysore, and Zeyu Jin. A differentiable perceptual audio metric learned from just noticeable differences. In *Interspeech 2020*, pages 2852–2856, 2020.
- [287] Han Han, Vincent Lostanlen, and Mathieu Lagrange. Learning to solve inverse problems for perceptual sound matching. *IEEE/ACM Transactions on Audio, Speech, and Language Processing (TASLP)*, 32:2605–2615, 2024.
- [288] Christopher Mitcheltree, Vincent Lostanlen, Emmanouil Benetos, and Mathieu Lagrange. Scaprl: Scattering transform with random paths for machine learning. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026.
- [289] Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.
- [290] Xenofon Karakostas, Diego Caviedes-Nozal, Antoine Richard, and Efrén Fernández-Grande. Room impulse response reconstruction with physics-informed deep learning. *The Journal of the Acoustical Society of America (JASA)*, 155(2):1048–1059, 2024.
- [291] Christian J Steinmetz, Shubhr Singh, Marco Comunità, Ilias Ibnyahya, Shanxin Yuan, Emmanouil Benetos, and Joshua D Reiss. ST-ITO: Controlling audio effects for style transfer with inference-time optimization. In *The 25th International Society for Music Information Retrieval Conference (ISMIR)*, 2024.
- [292] Wonchul Shin and Kyogu Lee. Synthrl: Cross-domain synthesizer sound matching via reinforcement learning. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 10162–10170, 2025.

- [293] Youichi Okita and Haruhiro Katayose. Audio effect estimation with DNN-based prediction and search algorithm. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 15952–15956, 2026.
- [294] Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 17981–17993, 2021.
- [295] Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *International Conference on Machine Learning (ICML)*, pages 32819–32848, 2024.
- [296] Clément Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. DiGress: Discrete denoising diffusion for graph generation. In *International Conference on Learning Representations (ICLR)*, 2023.
- [297] Tianhong Li, Yonglong Tian, He Li, Mingyang Deng, and Kaiming He. Autoregressive image generation without vector quantization. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, 2024.
- [298] Simon Rouard, Manu Orsini, Axel Roebel, Neil Zeghidour, and Alexandre Défossez. Continuous audio language models. In *International Conference on Learning Representations (ICLR)*, 2026.
- [299] Jean-Marie Lemerrier, Eloi Moliner, Simon Welker, Vesa Välimäki, and Timo Gerkmann. Unsupervised blind joint dereverberation and room acoustics estimation with diffusion models. *IEEE Transactions on Audio, Speech, and Language Processing (TASLPRO)*, 33:2244–2258, 2025.
- [300] Eloi Moliner, Michal Švento, Alec Wright, Lauri Juvela, Pavel Rajmic, and Vesa Välimäki. Unsupervised estimation of nonlinear audio effects: Comparing diffusion-based and adversarial approaches. In *28th International Conference on Digital Audio Effects (DAFx)*, pages 366–373, 2025.

- [301] Benjamin Elizalde, Soham Deshmukh, Mahmoud Al Ismail, and Huaming Wang. CLAP: Learning audio concepts from natural language supervision. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.
- [302] Yizhi Li, Ruibin Yuan, Ge Zhang, Yinghao MA, Xingran Chen, Hanzhi Yin, Chenghao Xiao, Chenghua Lin, Anton Ragni, Emmanouil Benetos, Norbert Gyenge, Roger Dannenberg, Ruibo Liu, Wenhua Chen, Gus Xia, Yemin Shi, Wenhao Huang, zili wang, Yike Guo, and Jie Fu. MERT: Acoustic music understanding model with large-scale self-supervised training. In *The International Conference on Learning Representations (ICLR)*, 2024.
- [303] Yen-Tung Yeh, Junghyun Koo, Marco Martínez-Ramírez, Wei-Hsiang Liao, Yi-Hsuan Yang, and Yuki Mitsufuji. Fx-Encoder++: Extracting instrument-wise audio effects representations from mixtures. In *The 26th International Society for Music Information Retrieval Conference (ISMIR)*, 2025.
- [304] Aogu Wada, Tomohiko Nakamura, and Hiroshi Saruwatari. Hyperbolic embeddings for order-aware classification of audio effect chains. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, pages 396–402, 2025.
- [305] Sungho Lee, Marco A. Martínez-Ramírez, Junghyun Koo, Wei-Hsiang Liao, Kyogu Lee, and Yuki Mitsufuji. Exploring the design space of representation learning for audio transformations. In *International Society for Music Information Retrieval Conference (ISMIR)*, 2026.
- [306] Philipp Götz, Cagdas Tuna, Andreas Walther, and Emanuël A. P. Habets. Contrastive representation learning for acoustic parameter estimation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.
- [307] Jacob Bitterman, Daniel Levi, Hilel Hagai Diamandi, Sharon Gannot, and Tal Rosenwein. RevRIR: Joint reverberant speech and room impulse response embedding using contrastive learning with application to room shape classification. In *Interspeech 2024*, pages 3280–3284, 2024.

- [308] Philipp Götz, Cagdas Tuna, Andreas Brendel, Andreas Walther, and Emanuel A. P. Habets. Blind acoustic parameter estimation through task-agnostic embeddings using latent approximations. In *International Workshop on Acoustic Signal Enhancement (IWAENC)*, pages 289–293, 2024.
- [309] Philipp Götz, Cagdas Tuna, Andreas Brendel, Andreas Walther, and Emanuel A. P. Habets. Multi-stage representation learning for blind room-acoustic parameter estimation with uncertainty quantification. *The Journal of the Acoustical Society of America (JASA)*, 159(1):247–259, 2026.
- [310] Philipp Götz, Gloria Dal Santo, Sebastian J. Schlecht, Vesa Välimäki, and Emanuel A. P. Habets. Matching reverberant speech through learned acoustic embeddings and feedback delay networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 14577–14581, 2026.

초 록

오디오 신호는 생성된 그대로 청취자에게 전달되는 경우가 드물며, 물리적 환경, 전자 장치 및 계산 과정이 가하는 변환을 거치게 된다. 우리는 이러한 변환을 가하는 객체 또는 구조를 시스템이라 부른다. 관측된 오디오로부터 이러한 시스템을 복원하는 작업, 즉 역문제는 가상 현실에서의 실내 음향 재현이나 음악 믹싱의 역설계와 같은 다양한 오디오 응용에서 등장한다. 본 학위논문은 이러한 역문제를 해결하기 위한 두 가지 상호 보완적인 패러다임에 기반한 방법들을 제시한다. 하나는 고전적 오디오 시스템에 대한 경사 기반 최적화를 가능하게 하는 미분 가능 신호 처리이며, 다른 하나는 역문제에 내재된 모호성을 다루기 위해 관측값이 주어졌을 때 가능한 시스템 표현의 조건부 분포를 학습하는 생성 모델링이다. 본 논문은 두 가지 영역을 다룬다. 하나는 시스템이 선형 시불변으로 잘 근사되는 실내 음향이고, 다른 하나는 시스템이 다양한 처리기로 구성된 합성 그래프로 표현되는 음악 신호 처리이다. 두 영역의 대표적인 시스템을 미분 가능하게 구현하고, 생성 모델링을 적용하며, 두 접근법을 함께 사용하는 혼합형 방법을 탐구함으로써, 도메인 지식과 데이터 기반 방법을 결합하여 학습 기반 접근법의 확장성을 유지하면서도 해석 가능하고 구조화된 해를 도출할 수 있음을 보인다.

주요어 : 역문제, 미분 가능한 신호처리, 생성 모델링, 오디오 포스트프로덕션, 실내 음향, 오디오 이펙트, 뮤직 프로덕션.

학 번 : 2020-22847

감사의 글

6년간의 박사과정을 지나 학위논문을 제출하기까지 많은 분들께 신세를 졌습니다.

먼저 박사과정 내내 아낌없이 지원해 주신 이교구 교수님께 감사드립니다. 아울러 학위논문을 완성하기 위한 조언을 아끼지 않으신 곽노준 교수님, 이원종 교수님, 남주한 교수님, 그리고 정다샘 교수님께 감사드립니다. 학창 시절 음악을 위한 공학의 세계를 처음으로 소개해 주신 여운승 교수님께도 감사드립니다.

연구실에서도 오랫동안 신세를 졌습니다. 처음 연구실에 자리를 잡도록 도와주신 형석이형, 주현이형, 재준이형, 진우형, 승렬이형, 그리고 정현이형에게 감사드립니다. 또 연구실에서 함께 부대끼며 연구하고 지낸 유리누나, 지원이형, 종호형, 경수형, 윤기님, 해선, 재현, 승우, 민준, 하준, 그리고 하윤에게도 고마운 마음을 전합니다.

I would like to express my sincere gratitude to Marco Martínez Ramírez, who supported me throughout my internship at Sony AI. I am also grateful to Sanjeel Parekh for his guidance during my internship at Meta, and for his patience while I explored and struggled in unfamiliar territory. My thanks also go to Matteo Scerbo and Professor Enzo De Sena for their support during our joint work.

학계와 산업계 밖에도 감사를 드리고 싶은 분들이 계십니다. 학창 시절을 함께해 준 명준, 세진, 계진, 지원, 승원, 그리고 핀림에게 고마움을 전합니다.

また、大学院に入学する前に東京で温かく迎えてくださったOTOOTOの木村ご夫妻、中村としまるさん、秋山徹次さんにも、心より感謝申し上げます。

마지막으로, 오랜 시간 기다려 주시고 변함없이 믿어 주신 부모님께 감사드립니다.