META-LEARNING FOR ADAPTIVE FILTERING

BY

JONAH CASEBEER

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2023

Urbana, Illinois

Abstract

Adaptive filtering algorithms form a cornerstone of intelligent signal processing infrastructure and play a vital role in a wide array of applications, including telecommunications, biomedical sensing, and seismology. Adaptive filters typically operate via specialized, online, iterative optimization methods such as least-mean squares or recursive least squares and aim to process signals in unknown or nonstationary environments. Such algorithms, however, can be slow and laborious to develop, require domain expertise to create, and necessitate mathematical insight for improvement.

In this thesis, we present a different approach to the design of adaptive filters. Our core contribution is a comprehensive framework that leverages the power of meta-learning and deep learning techniques to learn adaptive signal processing algorithms directly from data. By formulating adaptive filter development as a meta-learning problem, we train iterative update rules for adaptive filters using various forms of supervision and training. This approach enables us to overcome the limitations of traditional design methodologies and opens up new possibilities for developing highly efficient and effective adaptive filters.

To validate our framework, we focus on audio applications and systematically develop a family of meta-learned adaptive filters for key tasks such as system identification, inverse modeling, prediction, and informed interference cancellation. Through extensive evaluations of diverse audio applications, including acoustic echo cancellation, blind equalization, multichannel dereverberation, beamforming, telephony, and keyword spotting, we demonstrate the strong performance of our approach compared to existing methods.

Our findings indicate that our meta-learned adaptive filters not only function in realtime but also consistently excel across different tasks. We achieve remarkable performance gains by employing a single general-purpose configuration, highlighting the versatility and effectiveness of our framework. Moreover, our work pushes the boundaries of meta-learning and adaptive filter literature, leading to a new conceptual framework that has the potential to change how we approach, solve, and construct adaptive filter pipelines.

To my beloved grandfather,
who shaped my journey.

Acknowledgments

I am immensely grateful to all those who have contributed to the completion of this thesis, and I would like to extend my heartfelt thanks and deepest appreciation to each and every one of them. While I may have been the one typing this up, I certainly did not have to read or develop it alone.

For that, I am indebted to my committee members: Andrew Singer, Deepak Vasisht, Elahe Soltanaghai, Nicholas Bryan, and Paris Smaragdis. I am particularly grateful to my advisor, mentor, and dear friend, Paris. Paris took me under his wing as an undergraduate student and I haven't looked back since. Thank you, Paris, I have truly had a wonderful time as your student and without a doubt would do it again.

I owe a special thanks to Dr. Nick Bryan, who exemplified the essence of being a scientist and helped San Francisco feel like home. I also owe a great deal to Dr. Umut Isik, who instilled in me the art of scientific thinking. Our discussions on science in the era of deep learning transformed how I approach problems and were a true catalyst for this thesis. I want to thank Ivan Dokmanić, who for some crazy reason trusted an unsupervised freshman with all his lab equipment, got me hooked on signal processing research with this enthusiasm, and was the first to show me that math could be beautiful. Finally, I want to thank my high school electronics teacher David Bell, for letting me take whatever parts I wanted home and pointing me towards this field.

Behind every Ph.D. lies a lab, and I am incredibly thankful for my labmates: Cem, Shrikant, Efthymios, Zhepei, Krishna, and Dimitrios. I want to extend a special thank you to Shrikant, who mentored me during my undergraduate studies and prepared me for the challenges of the research world. I am also grateful to our junior lab members, who entrusted me with their mentorship. Zach and Adam, thank you for allowing me to learn and grow as an advisor, even during moments when "the blind leading the blind" might have been an apt description. I want to express my gratitude to Junkai, who taught me far more than he may realize. I also want to thank Jamshed, who kept me company through some of the most frustrating and exciting moments of graduate school.

During my time at the University of Illinois Urbana-Champaign, I have been fortunate to blur the lines between friendship and mentorship. I want to thank the UWAF crew: Kabir, Aaron, Ophir, Michael, and Austin for standing by my side through the literal barren tundras of this journey and putting up with my shenanigans. A special acknowledgment goes to Michael and Kabir, who generously shared their relentless curiosity and demonstrated to me what true mastery looks like.

I am truly grateful to my parents, who provided unwavering support and allowed me to embrace challenges, and to my siblings Mara, Linnae, and Mason, who continuously challenged me. Finally, to my loving wife, Sara, words cannot express the depth of my gratitude. Your presence, time, and encouragement have meant the world to me throughout this journey.

Thank you all for being part of this chapter in my life.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AEC	Acoustic Echo Cancellation. viii, x, 3, 16, 18, 22, 55, 60, 61, 65, 73–76
AF	Adaptive Filter. 1–5, 7, 18–20, 24, 33, 42, 47, 53, 54, 57, 59, 60, 71
BF	Beamforming. 3
DNN	Deep Neural Network. viii, 2–4, 16, 22, 32, 35, 60, 73, 75
EQ	Equalization. 3
FD	Frequency Domain. 6, 7, 10, 20, 60
FDAF	Frequency Domain Adaptive Filter. 8, 20
ISE	Instantaneous Squared Error. 8, 9, 12, 16, 19, 20, 24, 25, 28, 59
KF	Kalman Filter. 1, 14–17, 23, 60, 73
LMS	Least Mean Squares. 1, 12, 14, 15, 17, 23, 33, 34
Meta-AF	Meta-Learned Adaptive Filter. viii, x, 2, 18, 22, 23, 28–35, 37–39, 41, 44, 54, 56–59, 65, 73–75, 98–102
MSE	Mean Squared Error. 8, 9, 16, 20, 25, 28, 29
NLMS	Normalized Least Mean Squares. 1, 12–17, 23, 73
OLA	Overlap-Add. 9–11, 16, 20, 26
OLS	Overlap-Save. 9–11, 16, 26, 28, 55, 61, 71
RLS	Recursive Least Squares. 1, 13–17, 23, 73
RMSProp	Root Mean Square Propagation. 12–15, 17
SYS-ID	System Identification. 3
WPE	Weighted Prediction Error. 3
WSE	Weighted Squared Error. 9, 13, 16, 20

Chapter 1: Introduction

1.1 MOTIVATION

Our lives are filled with an array of devices that constantly transmit and receive signals, from acoustic signals played out by smart speakers to radio-frequency signals emitted by Wi-Fi routers. With this vast amount of data, however, comes noise - unwanted elements that contaminate every signal we send and receive. Real-world noise, such as background noise in a crowded room or echoes bouncing off the walls of a conference hall, poses a critical challenge for most devices. To address this issue, Adaptive Filters (AFs) have emerged as crucial components of modern infrastructure, enabling signal processing in adverse environments. The applications of AFs touch every signal processing domain including audio processing, telecommunications, biomedical sensing, astrophysics and cosmology, seismology, and more. Audio applications, in particular, are of exceptional importance and play a vital role in the devices and services we use daily, such as video/voice calls, automatic speech recognizers, hearing aids, and noise-canceling headphones. The language and tooling provided by AFs offer a versatile approach to tackling these tasks.

AFs are the synergistic combination of online optimization and clever modeling techniques. Typically, they rely on a coarse model of the task, with finer details and nuances controlled by an update rule that customizes the AF on-the-fly for the specific scenario at hand. This delicate balance between generic modeling and bespoke modeling necessitates the use of sophisticated optimization algorithms. The coarse AF atoms typically fall into four general configurations: system identification, inverse modeling, prediction, and interference cancellation. These configurations can be mixed and matched to describe a wide range of signal processing problems.

The design of AF update rules are typically guided by online optimization or adaptation principles such as Least Mean Squares (LMS) [1], Normalized Least Mean Squares (NLMS), Recursive Least Squares (RLS) [2], or the Kalman Filter (KF) [3, 4, 5, 6]. These approaches provide powerful mathematical frameworks for AF tasks. However, developing real-world capable AFs is a challenging and laborious task that requires significant domain expertise in both signal processing and the target domain (e.g., voice telephony). When an existing AF is deployed in a new environment or applied to a new task, it may need to be redesigned from scratch, requiring significant engineering hours, manual tuning, and deep mathematical insight. These harsh design barriers lead to the infamous feedback on conference calls, the challenge of using smart assistants in cars, and hit-or-miss hearing-aid quality. However, these are mostly software challenges, meaning that we can design better algorithms today and deploy better solutions tomorrow.

In this thesis, we aim to simplify the design and development of AF algorithms through a data-driven approach. Our goal is to invent general development systems that can produce high-performance AF systems for almost any AF task and use case without significant manual intervention. Additionally, we want our approach to be modular, and a drop-in replacement within existing AF systems. We call this general approach a Meta-Learned Adaptive Filter (Meta-AF) and benchmark it on a suite of audio tasks critical for telephony and smart devices. We frame this challenge as a meta-learning or end-to-end learning setup, enabling us to describe both a single isolated AF and a suite of AFs and other potentially Deep Neural Network (DNN) modules in one unified framework. This unified framework enables conceptual advances by connecting previously disparate approaches and provides practical benefits by providing simpler and modular engineering abstractions. We demonstrate this by developing end-to-end AF systems inside our framework and show that we can draw on lessons from meta-learning, optimizer design, adaptive filter theory, and digital signal processing (DSP) to build performant systems. In parallel, we discuss various forms of supervision and showcase trade-offs of unsupervised/self-supervised, and fully supervised training schemes.

The culmination of this thesis is a novel design perspective for all AF problems and an accompanying code-base, `meta-af`. We use `meta-af` for all of our development and evaluation. Our `meta-af` software enables easy translation of ideas useful for one AF domain to another and rapid development of novel techniques. In particular, we will demonstrate how using `meta-af`, we can scale from the design of a simple toy system identification task to a full telephony capture system with echo cancellation, noise suppression, multi-channel processing, and neural network post-processors. Our `meta-af` package provides both offline training code and online inference code, which is real-time capable on a single consumer-grade CPU core. Finally, to evaluate and benchmark all discussed algorithms, we built and maintained a suite of AF tasks called the `AF-Gym`. Through our experiments in the `AF-Gym`, on selected tasks, we achieve or approach state-of-the-art performance. We hope that our tools presented here can lead to better-performing systems and entice researchers and engineers alike.

1.2 PRIOR WORKS AND LITERATURE

AF tasks are typically classified into one of four categories: system identification, inverse modeling, prediction, and interference cancellation [5]. Each of these categories has numerous AF applications, and advances in one category or application can often be applied to many others. In the audio domain, Acoustic Echo Cancellation (AEC), the task of enabling full-duplex, can be formulated as a single- or multi-channel System Identification (SYS-ID) problem and has been extensively studied [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]. Equalization (EQ), the task of undoing the effect of some unknown system, can be formulated as an inverse modeling problem, has been explored in single- and multi-channel formats, and is particularly useful for sound zone reproduction and active noise control [19, 20, 21, 22, 23, 24]. Dereverberation, the task of removing echo from a recording, can be formulated as a Weighted Prediction Error (WPE) problem and has been extensively studied in this context [25, 26, 27, 28, 29, 30, 31]. Finally, multi-microphone enhancement or Beamforming (BF), the task of isolating a signal of interest using spatial information, can be formulated as an informed interference cancellation task and also has a breadth of associated approaches [32, 33, 34, 35, 36, 37, 38] and unmixing algorithms [39, 40, 41, 42, 43, 44]. These various audio tasks are both useful as standalone modules and as pre-processing steps for larger systems. In terms of training, most AF approaches are unsupervised in that they only perform learning at inference time and do not require oracle signals to construct the algorithms themselves.

When considering AFs in the context of the data-driven DNN revolution, we observe two main points. First, AFs continue to be widely used in practice but that research centers on hybrid fully-supervised approaches, which combine neural networks with standard AF algorithms. Second, the underlying AF algorithms and tools for developing new AFs have undergone little change. Hybrid approaches, however, have proven very successful. For example, in AEC, neural networks can be trained for residual echo suppression, noise suppression, reference estimation, and more [45, 46, 47, 48, 49, 50, 51, 52, 53]. Similarly, neural networks paired with AFs for active noise control tasks have shown impressive results [54, 55, 56, 57]. For dereverberation, DNNs have proven useful for both online and offline approaches, by directly estimating statistics of the dereverberated speech [58, 59, 60, 61, 62]. This pattern has repeated itself for beamforming applications, where DNNs have led to many performance improvements [63, 64, 65, 66]. In many of these works, DNNs are trained to predict ratio masks, or directly enhance/separate the desired signal. Essentially, they act as a distinct module within a larger pipeline that also uses AFs.

In developing larger AF pipelines, using end-to-end or joint training can significantly enhance performance by modeling the full processing pipeline instead of each individual component. Joint optimization techniques have led to advancements both with and without DNN modules and are useful for signal-level objectives [67, 68, 69, 70, 71, 72] as well as downstream objectives like speech recognition [73, 74]. This holds true for DNN modules as well [75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85]. Nearly all joint and hybrid training approaches are fully supervised. However, some of them sacrifice modularity for joint optimization.

In contrast, there are a few works that more closely integrate neural networks and AFs, to use DNNs for optimal control of AFs. These methods include both model-based [78, 82, 86, 87, 88, 89, 90] and model-free approaches [91, 92, 93, 94]. The former uses DNNs to estimate signal statistics, step-sizes, or augment portions of existing signal models, while the latter aims to learn AF control from scratch. These two classes of approaches differ from hybrid methods in that they use neural networks to update or control AFs directly, aiming to improve the performance of AFs themselves. Such improvements can be used in isolation or together with complementary hybrid or end-to-end approaches [78, 94]. The training and corresponding data requirements for these approaches are much more varied.

The idea of controlling AFs via neural networks is broadly related to several fields of signal processing and machine learning, including optimal control, optimization, automatic machine learning, reinforcement learning, and meta-learning. Relevant works in these areas involve automatic selection of step sizes [7, 95, 96], direct control of model weights [97, 98, 99], rapid fine-tuning [100], and meta-learning optimization rules [101]. There are many other flavors of meta-learning, but learned optimizers are most relevant [102, 103, 104, 105, 106, 107], and present the idea of using one neural network as a function that optimizes another function. However, such studies focus on creating learned optimizers for training neural networks in an offline setting, where the latter network is the final product, and the learned optimizer is otherwise discarded (or used to train additional networks).

We take inspiration from these works but make numerous advances specific to AFs. In particular, past learned optimizers [101] are element-wise, offline, real-valued, only a function of the gradient, and are trained to optimize general-purpose neural networks. In contrast, we design *online* AF optimizers that use multiple input signals, are complex-valued, adapt any kind of filter, and integrate numerous domain-specific insights to reduce complexity and improve performance. Moreover, we deploy learned optimizers to solve AF tasks as the end goal and do not use them to train downstream neural networks. This enables the development of modular end-to-end systems, which significantly improve performance. Compared to prior hybrid AF works, we replace the entire update with a neural network and provide a single comprehensive framework for all AF tasks. Furthermore, we will illustrate how our optimization techniques can adapt to varying data volumes and quality, as well as varying levels of computational resources.

1.3 CONTRIBUTIONS AND OUTCOMES

The key conceptual contribution of the thesis is a new design paradigm for AF systems. The software artifacts are contained in the open source and pip installable python package, `meta-af`, hosted on GitHub. It contains implementations of all algorithms discussed in both the background and main sections as well as the `AF-Gym`, which we also refer to as the `Zoo`.

1.3.1 Publications

The academic outcomes include workshop, conference, and journal papers:

- Auto-DSP: Learning to Optimize Acoustic Echo Cancellers [92]
- Meta-AF: Meta-Learning for Adaptive Filters [93]
- Meta-Learning for Adaptive Filters with Higher-Order Frequency Dependencies [94]
- Meta-AF Echo Cancellation for Improved Keyword Spotting [108]
- Supervised Multi-Step Adaptive Filters [109]
- NICE-Beam: Neural Integrated Covariance Estimators for Time-Varying Beamformers [78]

Other publications produced during the course of this thesis but not described here include the following references: [79, 91, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121].

1.3.2 Code

The use of software frameworks that automate autodiff has had a significant impact on deep learning, driving progress in various application areas. This thesis includes a similar tool tailored for adaptive filtering. The goal was to simplify and automate key tasks in adaptive filtering, taking advantage of the widespread availability and GPU compatibility of deep learning frameworks. This includes functions such as buffering, overlap-save computations, overlap-add processes, and real-time filtering operations. The open-source and software outcomes include:

- The `meta-af` package `https://github.com/adobe-research/MetaAF`
- Early `meta-af` work `https://github.com/jmcasebeer/autodsp`

1.3.3 Patents

The invention outcomes include submitted patents on:

- Meta-AF: Meta-Learning for Adaptive Filters
- NICE-Beam: Neural Integrated Covariance Estimators for Time-Varying Beamformers

1.4 NOTATION & CONVENTIONS

We provide an overview of the symbols and operators we use in Table 1.1. We denote scalars via lower-case symbols, column vectors via bold, lower-case symbols, and matrices via bold upper-case symbols. We use bracket indexing $[\tau]$ to denote time-varying signals and an underline to denote the time-domain counterpart of a Frequency Domain (FD) symbol. We index FD rows via the discrete frequency bin subscript, k, and columns via either the microphone subscript, m, or the frame buffer subscript, b. A single element can be indexed via the use of stacked subscripts kmb. Occasionally we will drop parenthesis or indices for brevity, but only when it does not confuse the operation.

Symbols	Description
$x \in \mathbb{R}$	A real-valued scalar
$x \in \mathbb{C}$	A complex-valued scalar
$\underline{x} \in \mathbb{R}$	A real-valued time-domain scalar
$\underline{\mathbf{x}} \in \mathbb{R}^N$	A time-domain N-dimensional column vector
$\mathbf{x} \in \mathbb{C}^N$	A complex-valued N-dimensional column vector
$\mathbf{X} \in \mathbb{C}^{M \times N}$	A complex-valued $M \times N$ matrix
$\mathbf{x}[\tau]$	A time-varying column vector
$\mathbf{X}[\tau]$	A time-varying matrix
$\mathbf{w}[\tau]$	FD AF linear-filter
$\mathbf{u}[\tau]$	FD AF input
$\mathbf{d}[\tau]$	FD AF target or desired response
$\mathbf{y}[\tau]$	FD AF estimated response
$\mathbf{d}[\tau] - \mathbf{y}[\tau]$	FD AF error
$\mathbf{s}[\tau]$	FD AF true desired signal
$\mathbf{I}_N$	An $N \times N$ identity matrix
$\mathbf{0}_{N \times R}$	$N \times R$ matrix of zeros
$\mathbf{1}_{N \times R}$	$N \times R$ matrix of ones
$\mathbf{F}_N$	N-point discrete Fourier transform (DFT) matrix
Operators	**Description**
$*$	Convolution
$.^{\top}$	Transpose
$.^{*}$	Complex conjugate
$.^{\mathsf{H}}$	Hermitian transpose
$\mathrm{diag}(\cdot)$	Vector to a diagonal matrix
$\mathrm{cat}(\cdots)$	Column vector concatenation (vertical stack)
E	Expected value
$\frac{(\cdot)}{(\cdot)}$	Element-wise division
$\|\| \cdot \|\|$	Euclidean norm
$\| \cdot \|$	Element-wise magnitude of complex value
$\angle$	Phase of a complex-value
$\ln$	Natural logarithm
$^{-1}$	Scalar or matrix inverse

Table 1.1: Symbols and operators for adaptive filtering that are used throughout this work.

Chapter 2: Background on Adaptive Filtering

Adaptive filters are a crucial component of this thesis, consisting of two main elements. The first component is the filter itself, which performs the signal processing and is typically parametric. The second component is the adaptation or optimization rule, which governs the adjustment of the filter parameters. We define a complete Adaptive Filter (AF) as an algorithm or optimizer that solves a specific objective function over time,

$$\hat{\boldsymbol{\theta}}[\tau] = \arg\min_{\boldsymbol{\theta}[\tau]} \mathcal{L}(\cdots)[\tau], \tag{2.1}$$

where $\hat{\boldsymbol{\theta}}$ represents the estimated time-varying filter parameters. Usually, this optimization is accomplished using an additive update rule of the form

$$\boldsymbol{\theta}[\tau+1] = \boldsymbol{\theta}[\tau] + \boldsymbol{\Delta}[\tau], \tag{2.2}$$

where $\boldsymbol{\Delta}[\tau]$ denotes the update applied to the filter parameters at time τ. In this thesis, we typically focus on linear Frequency Domain Adaptive Filter (FDAF), where the constituent filter parameters are represented by $\boldsymbol{\theta}[\tau] = \{\mathbf{w}[\tau]\}$, without loss of generality. The choice of the objective function $\mathcal{L}$ and the method for applying and updating the filter parameters $\hat{\boldsymbol{\theta}}[\tau]$ have a significant impact on the performance and applicability of the adaptive filter. Different configurations of these parameters can yield a wide range of applications, from linear acoustic echo cancellers to direction of arrival estimators. To ensure generality, we provide explanations that apply to any filter kind of filtering task.

Most often, the optimization objective $\mathcal{L}$ is computed with respect to the filter output $\mathbf{y}_{\mathrm{m}}$ and some reference $\mathbf{d}_{\mathrm{m}}$, at the reference receiver m. Some common settings includes the Mean Squared Error (MSE),

$$\mathcal{L}_{MSE}[\tau] = E[\|\mathbf{e}_{\mathrm{m}}[\tau]\|^2] = E[\|\mathbf{d}_{\mathrm{m}}[\tau] - \mathbf{y}_{\mathrm{m}}[\tau]\|^2], \tag{2.3}$$

which is widely used and aims to minimize the expected squared error. The MSE provides a measure of overall estimation accuracy and is quite popular when some statistical properties of the task are known ahead of time.

A second common objective is the Instantaneous Squared Error (ISE),

$$\mathcal{L}_{ISE}[\tau] = \|\mathbf{e}_{\mathrm{m}}[\tau]\|^2 = \|\mathbf{d}_{\mathrm{m}}[\tau] - \mathbf{y}_{\mathrm{m}}[\tau]\|^2, \tag{2.4}$$

which is a time-varying version of the MSE. The ISE is useful when real-time adaptation is a concern and signal statistics are unknown or not stationary.

A third objective is the Weighted Squared Error (WSE),

$$\mathcal{L}_{WSE}[\tau] = \sum_{n=0}^{\tau} \gamma^{\tau-n} \|\mathbf{d}_{\mathrm{m}}[n] - \mathbf{y}_{\mathrm{m}}[n]\|^2, \tag{2.5}$$

where γ is a tunable forgetting factor. The WSE acts as a middle-ground between the ISE and MSE since γ can be selected to prioritize recent information while still considering past knowledge.

These are just a few examples of loss functions for adaptive filters. The choice of loss depends on numerous criteria like the application, desired performance, prior knowledge, and more. Typically, these loss functions can only rely on knowledge that is available at inference time. This means that any oracle knowledge, such as clean signal exemplars or ground truth information, needs to be incorporated manually into the adaptation process. Though, oracle information like clean signal exemplars could be used to choose between ISE, MSE, and WSE type losses.

2.1 ONLINE FREQUENCY-DOMAIN FILTERING

Filtering is the fundamental operation performed by an adaptive filter to generate its output. In this thesis, our focus is on frequency domain filtering, which is widely used and the default approach for implementing adaptive filters in audio. We employ streaming frequency domain filtering using either the Overlap-Save (OLS) or Overlap-Add (OLA) convolution techniques.

The OLS method involves block processing by dividing the input signal into overlapping windows and recombining complete non-overlapping components. We use $\underline{u}_{\mathrm{m}}[\mathrm{t}] \in \mathbb{R}$ to represent the time-domain sample recorded by microphone m at discrete time t. We collect N such samples from microphone m to form the τ^{th} time-domain frame,

$$\underline{\mathbf{u}}_{\mathrm{m}}[\tau] = [\underline{u}_{\mathrm{m}}[\tau R - N + 1], \cdots, \underline{u}_{\mathrm{m}}[\tau R]] \in \mathbb{R}^N, \tag{2.6}$$

where τ is the frame index, N is the window length in samples, R is the hop size in samples, and $O = N - R$ is the overlap between frames in samples. Finally, we gather samples from all M microphones to form a multi-channel time-domain signal,

$$\underline{\mathbf{U}}[\tau] = [\underline{\mathbf{u}}_1[\tau], \cdots, \underline{\mathbf{u}}_M[\tau]] \in \mathbb{R}^{N \times M}. \tag{2.7}$$

We compute the corresponding frequency-domain representation,

$$\mathbf{U}[\tau] = \mathbf{F}_N \underline{\mathbf{U}}[\tau] \in \mathbb{C}^{K \times M}, \tag{2.8}$$

where K is the number of frequency bins, set to $K = N$ for this work. We select the m^th^ channel from a multi-channel frequency-domain representation using $\mathbf{u}_{\mathrm{m}}[\tau] \in \mathbb{C}^K$. We define the FD filter $\mathbf{w}_{\mathrm{m}}[\tau] \in \mathbb{C}^K$ and the frequency and time output for the m$^{\text{th}}$ channel as:

$$\begin{aligned} \mathbf{y}_{\mathrm{m}}[\tau] &= \operatorname{diag}(\mathbf{u}_{\mathrm{m}}[\tau]) \mathbf{Z}_w \mathbf{w}^*_{\mathrm{m}}[\tau] \in \mathbb{C}^K & (2.9)\\ \underline{\mathbf{y}}_{\mathrm{m}}[\tau] &= \mathbf{Z}_y \mathbf{y}_{\mathrm{m}}[\tau] \in \mathbb{R}^R, & (2.10) \end{aligned}$$

with the use of two anti-aliasing matrices,

$$\begin{aligned} \mathbf{Z}_w &= \mathbf{F}_K \mathbf{T}^\top_{K/2} \mathbf{T}_{K/2} \mathbf{F}_K^{-1} \in \mathbb{C}^{K \times K}, & (2.11)\\ \mathbf{Z}_y &= \bar{\mathbf{T}}_R \mathbf{F}_K^{-1} \in \mathbb{C}^{R \times K}, & (2.12) \end{aligned}$$

and two trimming matrices,

$$\begin{aligned} \mathbf{T}_{K/2} &= = [\mathbf{I}_{K/2}, \mathbf{0}_{K/2 \times K/2}] \in \mathbb{R}^{R \times K}, & (2.13)\\ \bar{\mathbf{T}}_R &= [\mathbf{0}_{R \times O}, \mathbf{I}_R] \in \mathbb{R}^{R \times K}, & (2.14) \end{aligned}$$

which trim the last $K/2$ samples from a vector and the first O samples respectively.

The counterpart to OLA is OLA filtering, which computes the frequency output, time output, and buffer $\underline{\mathbf{b}}_{\mathrm{m}}[\tau]$ as

$$\begin{aligned} \mathbf{y}_{\mathrm{m}}[\tau] &= \operatorname{diag}(\mathbf{u}_{\mathrm{m}}[\tau]) \mathbf{w}^*_{\mathrm{m}}[\tau] \in \mathbb{C}^K & (2.15)\\ \underline{\mathbf{y}}_{\mathrm{m}}[\tau] &= \mathbf{T}_R \mathbf{F}_K^{-1} \mathbf{y}_{\mathrm{m}}[\tau] + \bar{\mathbf{T}}_R \underline{\mathbf{b}}_{\mathrm{m}}[\tau - 1] \in \mathbb{R}^R & (2.16)\\ \underline{\mathbf{b}}_{\mathrm{m}}[\tau] &= \mathbf{F}_K^{-1} \mathbf{y}_{\mathrm{m}}[\tau] + \mathbf{T}_R^\top \bar{\mathbf{T}}_R \underline{\mathbf{b}}_{\mathrm{m}}[\tau - 1] \in \mathbb{R}^K. & (2.17) \end{aligned}$$

Here,

$$\mathbf{T}_R = [\mathbf{I}_R, \mathbf{0}_{R \times O}] \in \mathbb{R}^{R \times K}, \tag{2.18}$$

which trims the last O samples from a vector.

Typically, the forward and inverse discrete Fourier transforms are combined with analysis and synthesis windows, and zero-padding may also be employed. We typically use Hann windows [122], which have nice constant-overlap add properties.

For multi-channel multi-block input, single-channel output FD filters, the OLS/OLA equations described above are applied per channel, and anti-aliasing is applied per block. The per-frequency (anti-aliased) filter is $\mathbf{w}_{\mathrm{k}}[\tau] \in \mathbb{C}^{BM}$ where B represents the number of buffered frames and M represents the number of channels stacked together. The filter input is similarly stacked denoted as $\mathbf{u}_{\mathrm{k}}[\tau] \in \mathbb{C}^{BM}$, which requires straightforward modifications to equations (2.9) and (2.15). Picking between OLS and OLA is not always straightforward. If speed is your top priority, choose OLS. However, if your processing might introduce undesirable artifacts or vary rapidly over time, opt for OLA.

Methods like OLA and OLA are valuable for efficient real-time processing in both the frequency and time domains. Frequency domain filtering, in particular, offers several advantages that make it a popular choice in adaptive filtering applications. One key advantage of frequency domain filtering is its improved convergence properties compared to time domain filtering, for natural signals like human speech. By operating in the frequency domain, adaptive algorithms can exploit the inherent properties of the frequency representation of signals. More custom representations are also possible, such as those based on task-specific filter-banks [123, 124], portnoff [125] windows, learnable filter-banks [110, 111, 118, 126], and more. Frequency domain and custom domains often lead to faster convergence and more efficient adaptation, allowing adaptive filters to quickly adapt to changing signal conditions and provide better performance. This adaptation relies on sophisticated optimization algorithms, which we describe next.

2.2 ONLINE ADAPTATION

Given a filtering objective and filtering mechanics, we can develop filter adaptation rules. When developing adaptive filter adaptation rules for adaptive filters, it is common to leverage OLS or OLA filtering and solve the objective defined in (2.1) through optimization methods applied independently to each frequency bin. So, we modify (2.2) to be

$$\mathbf{w}_{\mathrm{k}}[\tau+1] = \mathbf{w}_{\mathrm{k}}[\tau] + \boldsymbol{\Delta}_{\mathrm{k}}[\tau], \tag{2.19}$$

where the update $\boldsymbol{\Delta}_{\mathrm{k}}[\tau]$ is per frequency k. We focus on three common conventional AF optimizers in this form as well as a machine learning optimizer to ground the development of our method to familiar, fundamental algorithms. When considering gradient-based optimization methods, we calculate the partial derivative of the objective function with respect to the conjugate transpose of the filter weights, $\mathbf{w}_{\mathrm{k}}[\tau]^{\mathsf{H}}$.

2.2.1 Least Mean Square

The Least Mean Squares (LMS) is a stochastic gradient descent method that uses the ISE loss and gradient. The LMS update is

$$\boldsymbol{\Delta}_{\mathrm{k}}[\tau] = -\lambda \boldsymbol{\nabla}_{\mathrm{k}}[\tau], \tag{2.20}$$

where λ is the step-size and $\boldsymbol{\nabla}_{\mathrm{k}}[\tau]$ is the gradient of the ISE. Selecting a single λ is challenging since a larger λ enables faster adaptation, but typically comes at the cost of lower peak performance. Several schemes have been proposed [5] to change λ across time and adaptively trade off convergence vs peak performance. Note, LMS is stateless and only a function of the gradient.

2.2.2 Normalized Least Mean Square

The Normalized Least Mean Squares (NLMS) algorithm is a variation of the LMS algorithm that incorporates a running normalizer based on the input power. This normalization step helps improve the convergence and stability of the adaptive filter. The NLMS update is

$$\mathbf{v}_{\mathrm{k}}[\tau] = \gamma \mathbf{v}_{\mathrm{k}}[\tau - 1] + (1-\gamma)\|\mathbf{u}_{\mathrm{k}}[\tau]\|^2 \tag{2.21}$$

$$\boldsymbol{\Delta}_{\mathrm{k}}[\tau] = -\lambda \frac{\boldsymbol{\nabla}_{\mathrm{k}}[\tau]}{\mathbf{v}_{\mathrm{k}}[\tau]}, \tag{2.22}$$

where the division is element-wise, λ is the step-size and γ is a forgetting factor. The NLMS algorithm combines the benefits of the LMS algorithm with the normalization of the adaptation step based on the input power, resulting in improved performance and less sensitivity to the specific value of λ. There is an alternative interpretation that corresponds to running LMS with a constraint on the norm of the update [5].

2.2.3 Root Mean Squared Propagation

The Root Mean Square Propagation (RMSProp) optimizer [127] modifies NLMS by replacing $\mathbf{v}_{\mathrm{k}}[\tau]$ with a gradient-based per-element running normalizer, $\boldsymbol{\nu}[\tau]$ using forget factor

γ as,

$$\boldsymbol{\nu}_{\mathrm{k}}[\tau] = \gamma \boldsymbol{\nu}_{\mathrm{k}}[\tau-1] + (1-\gamma)\|\boldsymbol{\nabla}_{\mathrm{k}}[\tau]\|^2 \tag{2.23}$$

$$\boldsymbol{\Delta}_{\mathrm{k}}[\tau] = -\lambda \frac{\boldsymbol{\nabla}_{\mathrm{k}}[\tau]}{\sqrt{\boldsymbol{\nu}_{\mathrm{k}}[\tau]}}, \tag{2.24}$$

The value, $1/\sqrt{\boldsymbol{\nu}_{\mathrm{k}}[\tau]}$ supplements the fixed step-size λ and acts as an adaptive learning rate, $\lambda/\sqrt{\boldsymbol{\nu}_{\mathrm{k}}[\tau]}$. RMSProp has similar properties to NLMS for linear filters.

2.2.4 Recursive Least Squares

The aim of the Recursive Least Squares (RLS) algorithm is to exactly solve the AF loss, most commonly the WSE error. This is accomplished by expanding the weighted least squares error into a function of the weighted empirical signal covariance matrix,

$$\boldsymbol{\Phi}_{\mathrm{k}}[\tau] = \sum_{\tau} \gamma^{N-\tau} \mathbf{u}_{\mathrm{k}}[\tau]\mathbf{u}_{\mathrm{k}}[\tau]^{\mathsf{H}}, \tag{2.25}$$

where the summation time-indices are application dependent (e.g. causal vs. non-causal implementations), and the (weighted) empirical cross-correlation vector

$$\mathbf{z}_{\mathrm{k}}[\tau] = \sum_{\tau} \gamma^{N-\tau} \mathbf{u}_{\mathrm{k}}[\tau]\mathbf{d}_{\mathrm{k}}[\tau]^{\mathsf{H}}, \tag{2.26}$$

is used to compute the exact solution to the resulting normal equations,

$$\boldsymbol{\Phi}_{\mathrm{k}}[\tau]\mathbf{w}_{\mathrm{k}}[\tau] = \mathbf{z}_{\mathrm{k}}[\tau]. \tag{2.27}$$

Running estimates of $\boldsymbol{\Phi}_{\mathrm{k}}[\tau]$ and $\mathbf{z}_{\mathrm{k}}[\tau]$ are also commonly used. However, instead of performing repeated matrix inversion, the matrix inversion lemma is used. Thus, RLS can be implemented using a time-varying precision (inverse covariance) matrix $\mathbf{P}_{\mathrm{k}}[\tau]$ and the Kalman-gain $\boldsymbol{\kappa}_{\mathrm{k}}[\tau]$,

$$\boldsymbol{\kappa}_{\mathrm{k}}[\tau] = \frac{\mathbf{P}_{\mathrm{k}}[\tau-1]\mathbf{u}_{\mathrm{k}}[\tau]}{\gamma + \mathbf{u}_{\mathrm{k}}[\tau]^{\mathsf{H}}\mathbf{P}_{\mathrm{k}}[\tau-1]\mathbf{u}_{\mathrm{k}}[\tau]} \tag{2.28}$$

$$\mathbf{P}_{\mathrm{k}}[\tau] = \frac{\mathbf{P}_{\mathrm{k}}[\tau-1] - \boldsymbol{\kappa}_{\mathrm{k}}[\tau]\mathbf{u}_{\mathrm{k}}[\tau]^{\mathsf{H}}\mathbf{P}_{\mathrm{k}}[\tau-1]}{\gamma} \tag{2.29}$$

$$\boldsymbol{\Delta}_{\mathrm{k}}[\tau] = \boldsymbol{\kappa}_{\mathrm{k}}[\tau](d_{\mathrm{km}}[\tau] - y_{\mathrm{km}}[\tau])^{*}, \tag{2.30}$$

where γ is a forgetting factor, $d_{\mathrm{km}}[\tau]$ and $y_{\mathrm{km}}[\tau]$ are the desired and estimated signal at frequency k and reference microphone m, and the initialization of $\mathbf{P}_{\mathrm{k}}[\tau]$ is critical and commonly based on the input signal-to-noise ratio.

Unlike LMS, NLMS, and RMSProp, RLS leverages higher-order or inter-parameter information. In the case of multi-block and/or multi-channel filters, there are multiple ways to formulate RLS, some of which differ from time-domain RLS. Common approaches include diagonalized RLS (D-RLS) and block diagonalized RLS (BD-RLS) as well as QR decomposition techniques [2, 128, 129] and differ in what terms of the covariance (precision) matrix are modeled. D-RLS makes an uncorrelated assumption and optimizes each $\mathrm{k}, \mathrm{m}, \mathrm{b}$ filter tap separately, forming K diagonal $\mathrm{BM} \times \mathrm{BM}$ precision matrices. BD-RLS couples across frames and channels by forming K separate $\mathrm{BM} \times \mathrm{BM}$ precision matrices. In the case of single block/channel filters, D-RLS, BD-RLS, and NLMS can be reduced to the same algorithm with different parameterizations. Intuitively, the denominator in NLMS corresponds to applying a 1×1 matrix inverse, which is like a fully diagonal RLS (assuming zero-mean). This inversion step is actually the crux of RLS, and can make RLS implementations particularly prone to numerical issues.

2.2.5 Kalman Filtering

The Kalman Filter (KF) is a recursive algorithm that turns these optimization tasks into state estimation problems. Intuitively, algorithms like LMS leave information on the table since they perform an update at time τ but wait to use that update until time $\tau + 1$. The KF attempts to leverage this extra information at τ by updating the filter and then seeing how well that update worked. The KF also comes with a complete noise and variance model, which makes it much more sophisticated and enables many other beautiful connections to both RLS and LMS [130].

At its core, the KF consists of two steps: the prediction step and the update step. The prediction step uses the system model to predict the current state (filter), and the update step incorporates the measurement to refine the state estimate. Here, we use $m|n$ to represent the estimate at time m using information up to time n. Denote the per-frequency state vector as $\mathbf{x}_{\mathrm{k}}[\tau]$, the system model as $\mathbf{F}_{\mathrm{k}}[\tau]$, the measurement matrix as $\mathbf{H}_{\mathrm{k}}[\tau]$, the system noise as $\mathbf{n}_{\mathrm{k}}[\tau]$, the process covariance matrix as $\mathbf{Q}_{\mathrm{k}}[\tau]$, and the measurement noise as $\mathbf{v}_{\mathrm{k}}[\tau]$.

The prediction step predicts the state and its covariance as follows:

$$\hat{\mathbf{x}}_{\mathrm{k}}[\tau|\tau-1] = \mathbf{F}_{\mathrm{k}}[\tau|\tau-1]\hat{\mathbf{x}}_{\mathrm{k}}[\tau-1|\tau-1] + \mathbf{n}_{\mathrm{k}}[\tau] \tag{2.31}$$

$$\mathbf{R}_{\mathrm{k}}[\tau|\tau-1] = \mathbf{F}_{\mathrm{k}}[\tau|\tau-1]\mathbf{R}_{\mathrm{k}}[\tau-1|\tau-1]\mathbf{F}_{\mathrm{k}}[\tau|\tau-1]^{\mathsf{H}} + \mathbf{Q}_{\mathrm{k}}[\tau] \tag{2.32}$$

In these equations, $\hat{\mathbf{x}}_{\mathrm{k}}[\tau|\tau-1]$ is the predicted state estimate given measurements up to time $\tau-1$, $\hat{\mathbf{x}}_{\mathrm{k}}[\tau-1|\tau-1]$ is the previous state estimate, $\mathbf{R}_{\mathrm{k}}[\tau|\tau-1]$ is the predicted state covariance, and $\mathbf{R}_{\mathrm{k}}[\tau-1|\tau-1]$ is the previous state covariance.

$$\mathbf{K}_{\mathrm{k}}[\tau] = \mathbf{R}_{\mathrm{k}}[\tau|\tau-1]\mathbf{H}_{\mathrm{k}}[\tau]^{\mathsf{H}}\left(\mathbf{H}_{\mathrm{k}}[\tau]\mathbf{R}_{\mathrm{k}}[\tau|\tau-1]\mathbf{H}_{\mathrm{k}}[\tau]^{\mathsf{H}} + \mathbf{N}_{\mathrm{k}}[\tau]\right)^{-1} \tag{2.33}$$

$$\hat{\mathbf{x}}_{\mathrm{k}}[\tau|\tau] = \hat{\mathbf{x}}_{\mathrm{k}}[\tau|\tau-1] + \mathbf{K}_{\mathrm{k}}[\tau](\mathbf{d}_{\mathrm{k}}[\tau] - \mathbf{H}_{\mathrm{k}}[\tau]\hat{\mathbf{x}}_{\mathrm{k}}[\tau|\tau-1]) \tag{2.34}$$

$$\mathbf{R}_{\mathrm{k}}[\tau|\tau] = (\mathbf{I} - \mathbf{K}_{\mathrm{k}}[\tau]\mathbf{H}_{\mathrm{k}}[\tau])\mathbf{R}_{\mathrm{k}}[\tau|\tau-1] \tag{2.35}$$

In these equations, $\mathbf{K}_{\mathrm{k}}[\tau]$ is the Kalman gain, $\mathbf{N}\mathrm{k}[\tau]$ is the measurement noise covariance, $\mathbf{d}_{\mathrm{k}}[\tau]$ is the measurement vector, $\hat{\mathbf{x}}_{\mathrm{k}}[\tau|\tau]$ is the updated state estimate, and $\mathbf{R}_{\mathrm{k}}[\tau|\tau]$ is the updated state covariance. This formulation is incredibly generic and requires that the user select all appropriate parameters. The Kalman filter has numerous applications, variations, and task-specific implementations. Similar to RLS, there can also be numerical and higher-order variations (D-KF, BD-KF, etc).

For example, the KF is a popular choice for echo cancellation where a diagonal variant (D-KF) [11] with particular system and measurement settings is quite popular and a common comparison point [87]. It is also possible to run LMS/NLMS/RMSProp/RLS with an update step, or the KF without an update step. In order to decouple these characteristics, we will often match all algorithms in terms of whether they perform an update step or not. When applicable, we will describe how many predict (P) or update (U) steps an algorithm takes via a suffix (e.g. D-KF-PU).

2.2.6 Adaptation Overview

When comparing conventional optimizers, their performance can generally be ranked as LMS, NLMS/RMSProp, RLS, and KF, with computational complexity following the reverse order. We display the big-$\mathcal{O}$ of these setups and their approximate complexity costs in Table 3.2. However, these complexity costs and inference expenses are not the end of the story. Conventional algorithms often exhibit sensitivity to tuning, nonstationarities, nonlinearities, and other factors that necessitate careful engineering to address issues and ensure stability. In the case of large systems, the computational cost of the KF can become prohibitive, requiring simplifications. For multi-block BD-RLS filters, poor partition conditioning can result in degraded performance and potential stability problems compared to alternatives such as NLMS [131].

In our study, we base our comparison on these five fundamental optimizers. For each task, we conduct an exhaustive grid-search tuning of hyperparameters, using held-out validation sets. This tuning process involves adjusting parameters, as demonstrated in Table 3.1 under the update rules column. When applicable, we also employ task-specific optimizers to comprehensively explore the performance of different algorithms and gain insights into state-of-the-art approaches.

For the AEC task, we included comparisons with the open-source double-talk robust Speex algorithm [12], a weighted-RLS (wRLS) algorithm [132], WebRTC-AEC3 [133], and the Neural-Kalman-filter [134]. It is worth noting that the Speex and wRLS algorithms were used as linear adaptive filters (with non-linear post-processors) by the first-place [53] and second-place [132] winners, respectively, in the ICASSP 2021 Acoustic Echo Cancellation Challenge [135]. The Neural-Kalman-filter (NKF) is a DNN based module and the best performing AEC at the time of this work. Since our work primarily focuses on linear adaptive filters, which can be combined with non-linear post-processors, we believe that D-KF, wRLS, Speex, WebRTC-AEC3, and NKF serve as strong baseline methods. In cases where AEC is combined with downstream tasks, we will conduct tuning across the specific task and provide pure deep learning baselines where necessary. Regarding equalization, we adhere to our five fundamental optimizers. On the other hand, for dereverberation, we compared our approach against NARA-WPE [30], which is a highly effective normalized BD-RLS-based optimizer [27, 30] and comparable to the original NTT implementation [136]. For beamforming, our comparisons encompass fundamental optimizers used in a generalized sidelobe canceller setup, the minimum-variance distortionless response beamformer, and mask-based beamformers [63, 64].

2.2.7 Conclusion

Adaptive filtering and optimization techniques play a crucial role in signal processing applications, allowing us to improve signal quality, and extract valuable information from noisy or distorted signals. These techniques are particularly useful when we need to estimate an unknown system or remove unwanted components from a signal. In the context of adaptive filtering, the goal is to iteratively update the filter coefficients of methods like OLS or OLA, which are commonly used, based on different loss functions such as ISE, MSE, or WSE. This updating process is carried out using optimization rules provided by algorithms like LMS, RMSProp, NLMS, RLS, and KF. These optimizers serve as the engines driving the adaptation of the filter coefficients.

Each of these optimization algorithms has its own strengths and weaknesses in terms of performance and computational complexity. For example, LMS and NLMS are relatively simple and efficient stochastic gradient descent methods that can provide satisfactory results in many cases. RLS, on the other hand, aims to solve the weighted least squares error exactly, making it suitable for scenarios where high precision is required. The KF leverages a state-space model and combines a priori knowledge with measurements to estimate the filter coefficients, making it well-suited for systems with known dynamics.

It's important to consider the specific requirements of each application and the characteristics of the input signals when selecting an appropriate optimization algorithm. Factors such as convergence speed, stability, robustness to nonstationarities and nonlinearities, and computational complexity should be taken into account. By leveraging the power of adaptive filtering and optimization techniques, we can enhance the performance of signal processing systems and enable a wide range of applications in fields such as audio processing, communications, control systems, and more.

Chapter 3: Meta-Adaptive Filtering

In this section, we describe the core conceptual contribution of this thesis: how to formulate the development of AF systems as a meta-learning problem. The control rules for all subsequently examined adaptive filter tasks are learned via some form of supervision (un-/self-/fully- supervised) and data. We call this general approach Meta-AF. With this approach, it is possible to automatically develop processing pipelines that surpass expert-designed systems for a variety of tasks with no manual intervention that can scale across available data and compute.

Meta-learning, in general, is a subfield of machine learning that aims to develop algorithms and models capable of learning to learn. The core idea is to design systems that can acquire new knowledge and adapt their behavior based on related prior experiences, allowing them to learn more efficiently and effectively in new tasks or domains. The intuitive connection with adaptive filtering is that AFs solve a streaming learning task, such as fitting a transfer function, and that knowledge of other related learning tasks, like fitting other transfer functions, can be leveraged to produce more effective algorithms. Here, we focus on meta-learning across instances where the filtering task is the same (e.g. AEC) but the optimal filtering parameters are different (e.g. AEC on different devices).

The two fundamental concepts in meta-learning are the notions of an optimizee, which refers to the entity being optimized or learned, and the optimizer, which refers to the entity controlling the optimizee. In the context of adaptive filter theory, the optimizee is the composition of AF loss $\mathcal{L}(\cdots)$ and associated filtering function, $h_{\boldsymbol{\theta}[\tau]}(\cdot)$,

$$\mathcal{L}(h_{\boldsymbol{\theta}[\tau]}(\cdot), \cdots) \tag{3.1}$$

The loss $\mathcal{L}$ is typically minimized by controlling the time-varying parameters of the filter, $\boldsymbol{\theta}[\tau]$, using an optimization scheme. By leveraging meta-learning techniques, the optimizer can learn to adapt optimizee parameters $\boldsymbol{\theta}[\tau]$ in a way that improves performance across a range of input signals and environments. Ultimately, this leads to improved filter convergence properties, higher output quality, and less manual tuning.

3.1 FORMULATION AS META-LEARNING

We approach the design of AF algorithms as a meta-learning problem, where we train neural networks to control AFs using data, resulting in meta-learned adaptive filters. This is a departure from conventional AFs that are manually designed by human engineers. To accomplish this, we introduce a learned optimizer, denoted as $g_{\boldsymbol{\phi}}(\cdot)$, which is a neural network with one or more input signals and parameterized by weights $\boldsymbol{\phi}$. Its purpose is to optimize an AF loss, referred to as the *optimizee*, denoted as $\mathcal{L}(h_{\boldsymbol{\theta}[\tau]}(\cdot), \cdots)$, comprising a loss function $\mathcal{L}$ and an adaptive filter $h_{\boldsymbol{\theta}}$. Typically, the optimizer will use an additive update scheme:

$$\boldsymbol{\theta}[\tau+1] = \boldsymbol{\theta}[\tau] + g_{\boldsymbol{\phi}}(\cdot), \tag{3.2}$$

where $\boldsymbol{\theta}[\tau]$ represents the parameters of the filtering function $h_{\boldsymbol{\theta}[\tau]}$ at time τ. The inner or AF learning objective is to find optimal filter parameters, $\boldsymbol{\theta}[\tau]$ whereas the outer or meta-learning goal is to find optimal AF optimizer parameters, $\boldsymbol{\phi}$. We find optimal optimizer parameters with respect to some filter, filter loss, and a given dataset $\mathcal{D}$. The optimizer is then *optimal*, or custom learned, for that particular combination of filter, loss, and dataset. This two-level objective is formulated as a meta-objective,

$$\hat{\boldsymbol{\phi}} = \arg\min_{\boldsymbol{\phi}} E_{\mathcal{D}}[\ \mathcal{L}_M(\ g_{\boldsymbol{\phi}}, \mathcal{L}(h_{\boldsymbol{\theta}}(\cdot), \cdots)\ )\ ], \tag{3.3}$$

where $\mathcal{L}_M$ is a functional defining the meta-loss (or optimizer loss). It depends on $g_{\boldsymbol{\phi}}$, the AF loss $\mathcal{L}$ with one or more inputs, and the filtering function $h_{\boldsymbol{\theta}[\tau]}$ that itself has one or more inputs and parameters $\boldsymbol{\theta}[\tau]$. In essence, solving (3.3) enables us to learn a network $g_{\boldsymbol{\phi}}(\cdot)$ that can effectively optimize the AF loss $\mathcal{L}$ through repeated additive updates. Once learned, this optimizer can be deployed and used to optimize filtering operations on signals similar to those encountered in $\mathcal{D}$. This general formulation lets us tackle AF design with a single tool, instead of needing to develop new approaches for each new AF task. What is particularly useful is that the training procedure can be determined automatically by specifying the filter and dataset, thereby removing the need for any manual intervention beyond picking the filter structure.

3.2 FILTERING AS THE OPTIMIZEE

The optimizee, or the composition of the AF loss $\mathcal{L}$ and filter $h_{\boldsymbol{\theta}[\tau]}$ is optimized via (3.2) and is typically a function of the current filter parameters, filter output, and some reference signal. Let's take time-domain system identification with ISE as an example. Call the vector which holds a buffer of the previous system inputs $\mathbf{u}[\tau]$, the scalar system output $d[\tau]$, and the filter scalar output,

$$y[\tau] = h_{\boldsymbol{\theta}[\tau]}(\mathbf{u}[\tau]) = \mathbf{w}[\tau]^\top \mathbf{u}[\tau], \tag{3.4}$$

where $\boldsymbol{\theta}[\tau] = \mathbf{w}[\tau]$. The loss,

$$\mathcal{L}(h_{\boldsymbol{\theta}[\tau]}(\mathbf{u}[\tau]), d[\tau]) = \|d[\tau] - h_{\boldsymbol{\theta}[\tau]}(\mathbf{u}[\tau])\|^2 = \|d[\tau] - y[\tau]\|^2 = \|e[\tau]\|^2, \tag{3.5}$$

is now a function of all those quantities. In general, the filter can be any reasonable differentiable filtering operator such as a time-domain FIR filter, lattice FIR filter, non-linear filter, FD filters, multi-delayed block FD filter [4], etc. Similarly, the AF loss can be any reasonable differentiable loss such as the ISE, MSE, WSE, a regularized loss, negative log-likelihood, mutual information, etc.

For our work, we focus on single- and multi-channel multi-frame linear block FDAFs $h_{\boldsymbol{\theta}}$ applied via OLA or OLA with parameters $\boldsymbol{\theta}[\tau] = \{\mathbf{w}[\tau] \in \mathbb{C}^{K \times B \times M}\}$ with B buffered frames and M channels per frequency $\mathrm{k} \in K$. In cases where the AF produces multiple outputs, like multi-speaker and multi-microphone echo cancellation we can run multiple instances of $h_{\boldsymbol{\theta}}$ in parallel.

We will typically set the AF loss $\mathcal{L}[\tau]$ to be the ISE via (2.4) with gradient computed with respect to the per-frequency weight $\mathbf{w}_{\mathrm{k}}[\tau]^{\mathsf{H}}$ as:

$$\boldsymbol{\nabla}_{\mathrm{k}}[\tau] = \mathbf{u}_{\mathrm{k}}[\tau](y_{\mathrm{km}}[\tau] - d_{\mathrm{km}}[\tau])^*. \tag{3.6}$$

The AF loss is an input feature for the optimizer, with the true objective being the meta-objective. Interestingly, improved or more relevant AF losses enhance performance on the meta-objective, as the optimizer leverages them to address the meta-optimization problem, ultimately allowing for more efficient solutions to the global optimization challenge. The key difference is that the AF loss can be computed at inference time since it does not depend on oracle or non-causal information, whereas the meta-loss can only be computed at training time.

3.3 ADAPTATION AS THE OPTIMIZER

Our optimizer $g_{\boldsymbol{\phi}}$ is inspired by conventional AF optimizers, as described in the background, section 2.2, but updated to have a neural network form. Specifically, our focus is on developing a generalized, stochastic variant of conventional optimizers that can process parameters independently or coupled across frequency, channels, and buffer frame dimensions of an arbitrary optimizee. Intuitively, coupling more parameters increases the modeling demands on $g_{\boldsymbol{\phi}}$. However, this also enables amortization, as each instance of the optimizer can now control multiple optimization problems (parameters) simultaneously. To achieve this, we share the weights $\boldsymbol{\phi}$ across all simultaneous optimization problems, where for each simultaneous problem we maintain a separate optimizer state, $\boldsymbol{\psi}[\tau]$, and construct separate optimizer inputs, $\boldsymbol{\xi}[\tau]$. A key advantage of this framing is that learned optimizers can scale at both training and test time to optimizees with larger or smaller numbers of parameters as long as the optimizee architecture is the same.

To illustrate this concept, we can start with the simplest yet most flexible meta-learned optimizer, which operates on each parameter of the optimizee separately. A reasonable set of inputs to the optimizer are the vector:

$$\boldsymbol{\xi}_{\mathrm{kbm}}[\tau] = [\boldsymbol{\nabla}_{\mathrm{kbm}}[\tau], \mathbf{u}_{\mathrm{kbm}}[\tau], \mathbf{d}_{\mathrm{kbm}}[\tau], \mathbf{y}_{\mathrm{kbm}}[\tau], \mathbf{e}_{\mathrm{kbm}}[\tau]] \in \mathbb{C}^5, \tag{3.7}$$

where $\boldsymbol{\nabla}_{\mathrm{kbm}}[\tau]$ represents the gradient of the optimizee with respect to $\boldsymbol{\theta}_{\mathrm{kbm}}$, and $\mathbf{e}_{\mathrm{kbm}}[\tau] = \mathbf{d}_{\mathrm{kbm}}[\tau] - \mathbf{y}_{\mathrm{kbm}}[\tau]$. Such an optimizer produces two outputs: the update scalar-valued $\boldsymbol{\Delta}_{\mathrm{kbm}}[\tau] \in \mathbb{C}$ and the internal state $\boldsymbol{\psi}_{\mathrm{kbm}}[\tau+1] \in \mathbb{H}$, where H is the state size. In the case of frequency-independent processing, we have:

$$\begin{aligned} (\boldsymbol{\Delta}_{\mathrm{kbm}}[\tau], \boldsymbol{\psi}_{\mathrm{kbm}}[\tau+1]) \quad &= g_{\boldsymbol{\phi}}(\boldsymbol{\xi}_{\mathrm{kbm}}[\tau], \boldsymbol{\psi}_{\mathrm{kbm}}[\tau]) && (3.8)\\ \boldsymbol{\theta}_{\mathrm{kbm}}[\tau+1] \quad &= \boldsymbol{\theta}_{\mathrm{kbm}}[\tau] + \boldsymbol{\Delta}_{\mathrm{kbm}}[\tau]. && (3.9) \end{aligned}$$

Although fully independent processing is conceptually simple, it is rarely the most effective approach. The optimizer is effectively blind to how other parameters are being updated, and while the updates may implicitly incorporate some of this knowledge due to training, it cannot explicitly access it. This leads to drawbacks in terms of complexity and performance. Fortunately, it is easy to sidestep these issues. A highly effective approach is the frame-channel coupled optimization setup. In this scheme, each frequency is treated as a separate problem, but all values at each frequency, such as delayed frames and channels, are processed and updated jointly. This requires minimal modification of the previous update formulas, using the following inputs:

$$\boldsymbol{\xi}_{\mathrm{k}}[\tau] = \mathrm{cat}(\boldsymbol{\nabla}_{\mathrm{k}}[\tau], \mathbf{u}_{\mathrm{k}}[\tau], \mathbf{d}_{\mathrm{k}}[\tau], \mathbf{y}_{\mathrm{k}}[\tau], \mathbf{e}_{\mathrm{k}}[\tau]) \in \mathbb{C}^{5\mathrm{BM}}, \tag{3.10}$$

and vector valued outputs $\boldsymbol{\Delta}_{\mathrm{kbm}}[\tau] \in \mathbb{C}^{\mathrm{BM}}$, and $\boldsymbol{\psi}_{\mathrm{kbm}}[\tau+1] \in \mathbb{H}$. Notice that the number of optimizer calls has decreased by a factor of BM whereas the inputs and outputs of each

Optimizer	Inputs	State	Params	$\mathbf{\Delta}_{\mathrm{k}}[\tau]$
LMS	$\mathbf{\nabla}_{\mathrm{k}}[\tau]$	$\emptyset$	λ	(2.20)
NLMS	$\mathbf{\nabla}_{\mathrm{k}}[\tau], \mathbf{u}_{\mathrm{k}}[\tau]$	$\mathbf{v}_{\mathrm{k}}[\tau]$	λ, γ	(2.22)
RMSProp	$\mathbf{\nabla}_{\mathrm{k}}[\tau]$	$\boldsymbol{\nu}_{\mathrm{k}}[\tau]$	λ, γ	(2.24)
BD-RLS	$\mathbf{u}_{\mathrm{k}}[\tau], \mathbf{d}_{\mathrm{k}}[\tau], \mathbf{y}_{\mathrm{k}}[\tau]$	$\mathbf{P}_{\mathrm{k}}[\tau]$	γ	(2.30)
KF	$\mathbf{u}_{\mathrm{k}}[\tau], \mathbf{d}_{\mathrm{k}}[\tau], \mathbf{y}_{\mathrm{k}}, \mathbf{w}_{\mathrm{k}}[\tau]$	$\mathbf{R}_{\mathrm{k}}[\tau], \mathbf{x}_{\mathrm{k}}[\tau]$	[2]	(2.32), (2.35)
Meta-AF	$\boldsymbol{\xi}_{\mathrm{k}}[\tau]$	$\boldsymbol{\psi}_{\mathrm{k}}[\tau]$	$\boldsymbol{\phi}$	(3.2)

Table 3.1: Comparison of baseline optimizers in terms of inputs, state, and tunable parameters. Leveraging different input quantities leads to different kinds of state, which dictates memory requirements, and different parameters, which dictates tuning workload. This table compares these quantities.[2] The KF has a whole host of potential parameterizations due to its flexibility. These range from learning observation and system matrices to hand-picking them ahead of time. We will describe the choices we make depending on the task.

optimizer call have increased by a factor of BM when using the following update equations:

$$(\mathbf{\Delta}_{\mathrm{k}}[\tau], \boldsymbol{\psi}_{\mathrm{k}}[\tau+1]) = g_{\boldsymbol{\phi}}(\boldsymbol{\xi}_{\mathrm{k}}[\tau], \boldsymbol{\psi}_{\mathrm{k}}[\tau]) \tag{3.11}$$

$$\boldsymbol{\theta}_{\mathrm{k}}[\tau+1] = \boldsymbol{\theta}_{\mathrm{k}}[\tau] + \mathbf{\Delta}_{\mathrm{k}}[\tau]. \tag{3.12}$$

These modifications result in a significant net complexity reduction and improve performance. We explore more sophisticated coupling schemes in section 4.3.

The actual architecture of the optimizer has a significant impact on performance. Our most standard setup uses a small DNN composed of a linear layer, nonlinearity, and two Gated Recurrent Unit (GRU) layers with hidden size $H = 32$, followed by two additional linear layers with nonlinearities, where all layers are complex-valued. We will discuss other options for the optimizer in later sections since its configuration as a neural module gives us lots of design flexibility. For a comparison of optimizer inputs, state, parameters, and gradients, please see Table 3.1. In Table 3.1 we display one possible variation of a Meta-AF model, which makes several modeling assumptions that we describe later. The feature requirements shown here are meant to give a general sense to a reader.

An interesting feature of Meta-AF models is that they are highly configurable. If a practitioner identifies a signal or feature that may be useful for the optimization procedure, they can simply collate that feature to those already in $\boldsymbol{\psi}_{\mathrm{k}}[\tau]$. An example of this could include anything from accelerometer information for head-mounted devices to nonlinear reference signals in the case of AEC. For a general sense of optimizer complexity see Table 3.2. Where we again show the configuration of a particular model.

A point of interest here is that the computational complexity of Meta-AF is highly flex-

Optimizer	Big-$\mathcal{O}$	$\approx \mathbb{C}$MACS
LMS	$\mathcal{O}(\mathrm{KMB})$	KMB
NLMS	$\mathcal{O}(\mathrm{KMB})$	5KMB
RMSProp	$\mathcal{O}(\mathrm{KMB})$	6KMB
BD-RLS	$\mathcal{O}(\mathrm{K(MB)^2})$	$\mathrm{K(4(MB)^2 + 5MB)}$
KF	$\mathcal{O}(\mathrm{K(MB)^2})$	[1]
Meta-AF	$\mathcal{O}(\mathrm{K(H^2 + MBH)})$	$\mathrm{K(12H^2 + (21 + 10MB)H)}$

Table 3.2: Comparison of baseline optimizers in terms of asymptotic and practical complexity. All optimizers are configured to operate independently per frequency and are thus linear in complexity with respect to K. However, they have different modeling assumptions across microphones and frames (K, B). The diagonal nature of LMS/NLMS/RMSProp leads to good complexity scaling however, they can not leverage the same relationships that RLS/ KF can.[1] The compute of a KF varies wildly depending on the task and configuration. It typically costs at least as much as two RLS updates.

ible. In a later section, we will discuss how to drastically reduce and increase the compute requirements. As expected, models that require more compute are typically able to achieve higher peak performance.

In summary, the Meta-AF design differs from LMS-, NLMS-, and RMSProp-like optimizers, which either have no state (e.g., LMS) or minimal state dynamics (e.g., NLMS, RMSProp). Furthermore, we also distinguish our approach from other learned optimizers [101] by incorporating training and architecture insights that enable better performance on streaming problems. We discuss these advances in a later section and also ablate their performance characteristics. The theoretical analysis and exploration of these neural network-based optimizers in adaptive filter theory hold great potential for advancing our understanding of optimization in signal processing and machine learning domains [137]. For example questions about how to learn optimizers with guarantees, how to relate learned optimizers to conventional ones, and how to quantify performance gains are all active areas of research [138, 139].

3.4 META-LEARNING AN OPTIMIZER

To learn an optimizer, we need to define the meta-objective and establish a training procedure. We will explore two general classes of meta losses, denoted as $\mathcal{L}_M(\cdot)$, which allow us to quantify the performance of optimizer parameters $\boldsymbol{\phi}$. Then, we will discuss a general training procedure to learn the optimizer parameters, $\boldsymbol{\phi}$.

3.4.1 Frame Independent Losses

First, we introduce the *frame independent* loss,

$$\mathcal{L}_M = \ln \frac{1}{L} \sum_{\tau}^{\tau+L} \mathcal{L}(\cdots), \tag{3.13}$$

which sums the AF loss across time. This loss computes the average loss across time by summing the AF loss over a time horizon of L frames. The logarithm is applied to reduce the dynamic range, which has been found to empirically improve learning. The frame-independent loss assumes that optimizer performance can be decomposed into a series of frame-wise losses. The idea of accumulating independent batch-wise losses is commonly employed in deep-learning-based meta-learning[101]. For example, in the case of a frequency-domain ISE loss,

$$\mathcal{L}_M = \ln \frac{1}{L} \sum_{\tau}^{\tau+L} E[||\mathbf{d}_{\mathrm{m}}[\tau] - \mathbf{y}_{\mathrm{m}}[\tau]||^2], \tag{3.14}$$

where $\mathbf{d}_{\mathrm{m}}[\tau]$ and $\mathbf{y}_{\mathrm{m}}[\tau]$ are the desired and estimated FD signal vectors of the reference channel m (e.g. $\mathrm{m} = 0$). A natural class of frame-independent losses are computed directly on the filter parameters since by definition these can be decomposed across updates. Frame independent losses align the AF loss with the meta-loss, making them straightforward to implement and valuable for debugging and development purposes.

3.4.2 Frame Accumulated Losses

Not all useful loss metrics, however, can be decomposed in a per-frame fashion. We, therefore, propose a more general class of losses called *frame-accumulated* losses, which model the entire filtering procedure and can capture different information than a per-frame loss. Generically,

$$\mathcal{L}_M = \ln \mathcal{L}(\bar{\mathbf{y}}, \cdots) \tag{3.15}$$

where this loss is computed by accumulating all filtered outputs across some time span. Frame-accumulated losses can leverage different signals or information compared to frame-independent losses. Though, it is possible to configure the frame-accumulated loss to be similar to a frame-independent loss. For instance, instead of using frame-wise ISE, we can use the MSE,

$$
\begin{aligned}
\mathcal{L}_M &= \ln E[||\bar{\mathbf{d}}_{\mathrm{m}}[\tau] - \bar{\mathbf{y}}_{\mathrm{m}}[\tau]||^2] &(3.16)\\
\bar{\mathbf{y}}_{\mathrm{m}}[\tau] &= \mathrm{cat}(\underline{\mathbf{y}}_{\mathrm{m}}[\tau], \underline{\mathbf{y}}_{\mathrm{m}}[\tau+1], \cdots, \underline{\mathbf{y}}_{\mathrm{m}}[\tau+L-1]), &(3.17)\\
\bar{\mathbf{d}}_{\mathrm{m}}[\tau] &= \mathrm{cat}(\underline{\mathbf{d}}_{\mathrm{m}}[\tau], \underline{\mathbf{d}}_{\mathrm{m}}[\tau+1], \cdots, \underline{\mathbf{d}}_{\mathrm{m}}[\tau+L-1]) &(3.18)
\end{aligned}
$$

where $\underline{\mathbf{d}}_{\mathrm{m}}[\tau]$ and $\underline{\mathbf{y}}_{\mathrm{m}}[\tau]$ are the time-domain desired and estimated responses of reference channel m and $\bar{\mathbf{d}}_{\mathrm{m}}[\tau] \in R^{RL}$ and $\bar{\mathbf{y}}_{\mathrm{m}}[\tau] \in R^{RL}$.

To compute this loss for a given optimizer $g_{\boldsymbol{\phi}}$, we run the update rule in Equation 3.2 for a time horizon of L frames, concatenate the sequence of time-domain outputs and target signals to form longer signals, compute the time-domain MSE loss, and finally take the logarithm. This setup resembles the frame independent ISE, but takes into account additional filter dynamics, and frame borders, and typically results in different optimizer dynamics.

While both styles of losses use the same time horizon, the frame accumulated loss allows us to model more general objectives, since the loss does not need to decompose across frames. For example, it can model boundaries between adjacent updates and implicitly learn updates that are STFT consistent [140], and complex downstream processing like off-the-shelf speech classifiers [108]. To the best of our knowledge, this frame accumulated loss is novel for adaptive filters and can be easily extended to other types of information that cannot be factored across frames. In section 5, we explore more sophisticated loss functions, including large neural network-based loss functions, which do not admit a simple frame-wise decomposition. Then, in section 6.1.4, we compare performance on a system identification task in Fig. 6.3.

3.4.3 Self-Supervised and Fully Supervised Losses

Both frame and accumulated losses can be configured to leverage different information at training times. We conceptually decompose the kind of information leveraged into mixture information, which is available in the wild, and oracle information, which requires some sort of collection or preparation procedure. For example, a raw mixture recorded at a microphone is mixture information whereas the isolated speech content from that mixture is oracle information.

We call losses that only rely on mixture information, whether frame-wise or accumulated, self-supervised. This emphasizes the fact that they are only supervised by their own structure and natural filter characteristics. Such losses are very versatile since they can be trained on real-world data and customized to any kind of environment with minimal data cleaning or collection efforts. On the other hand, we call losses which rely on oracle information, fully-supervised. These kinds of losses can often outperform their self-supervised counterparts. This improved performance comes at zero additional inference complexity since the loss function does not impact inference time complexity. However, they rely on a costly and time-consuming data collection or cleaning procedure, since they require external information. In noiseless or particularly friendly environments, self and fully supervised losses may be the same.

In the later sections of this chapter, we examine one such scenario, where the noiseless nature of the task leads the self-supervised loss to effectively be fully-supervised. We explore these concepts in more detail in section 5.

3.4.4 Meta Training Procedure

To train the optimizer $g_{\boldsymbol{\phi}}$ from data, we employ standard deep learning tools, such as JAX and Haiku [141, 142], which provide automatic differentiation for training and inference. We use truncated backpropagation through time (TBPTT) [143] with the Adam optimizer [144], referred to as our meta optimizer, to solve the meta-objective in equation 3.3.

In Algorithm 3.1, we present a specific case of our training algorithm, which involves a frame-channel coupled optimizer, a frame accumulated loss, and a batch size of one. In this algorithm, Stft is an OLA or OLS processor, Grad returns the gradient of the first argument with respect to the second, Sample randomly samples signals from a dataset $\mathcal{D}$, and NextL grabs the next L time buffers from a longer signal. In practice, we use batching.

The training algorithm in Algorithm 3.1 iterates over batches of signals. In each batch, a random sample of signals $\underline{\mathbf{d}}_{\mathrm{m}}, \underline{\mathbf{y}}_{\mathrm{m}}$ is drawn from the dataset $\mathcal{D}$. The hidden state $\boldsymbol{\psi}$ is initialized to zeros, and the next L time buffers are extracted from the input signals using NextL. The optimizer $g_{\boldsymbol{\phi}}$ is then applied to optimize the filtered output $\hat{\mathbf{y}}_{\mathrm{m}}$ and update the hidden state $\boldsymbol{\psi}$. The meta-objective $\mathcal{L}_M$ is evaluated based on the filtered output and the target output, and the gradient of the meta-objective with respect to the optimizer parameters $\boldsymbol{\phi}$ is computed using automatic differentiation. Finally, the optimizer parameters are updated using the meta-optimizer. The process is repeated until convergence, and the learned optimizer parameters $\boldsymbol{\phi}$ are returned as the output of the algorithm.

In this training algorithm, we have chosen specific setups for the optimizee, optimizer, and meta-objective to provide a comprehensive example. However, it is important to note that in later chapters, we will discuss the flexibility and possible variations in these components. The general procedure outlined in this algorithm remains the same for all future sections of this paper, regardless of the specific filtering task selected. In section 6.1.4, we compare

Algorithm 3.1 Meta-training algorithm for an STFT-based optimizee, frame-channel coupled optimizer, and a frame-accumulated meta-objective.

```
function FORWARD($g_{\boldsymbol{\phi}}, \boldsymbol{\psi}, h_{\boldsymbol{\theta}}, \underline{\mathbf{U}}, \underline{\mathbf{d}}_{\mathrm{m}}$)
    for $\tau \leftarrow 0$ to $L$ do                                          ▷ Unroll
        $\mathbf{U}[\tau], \mathbf{d}[\tau] \leftarrow$ STFT($\underline{\mathbf{U}}[\tau], \underline{\mathbf{d}}_{\mathrm{m}}[\tau]$)          ▷ Forward STFT
        $\mathbf{y}_{\mathrm{m}}[\tau] \leftarrow h_{\boldsymbol{\theta}}(\mathbf{U}[\tau])$                          ▷ Save filter output
        $\underline{\mathbf{y}}_{\mathrm{m}}[\tau] \leftarrow \mathrm{STFT}^{-1}(\mathbf{y}_{\mathrm{m}}[\tau])$                  ▷ Inverse STFT
        $\mathcal{L} \leftarrow ||\mathbf{d}_{\mathrm{m}}[\tau] - \mathbf{y}_{\mathrm{m}}[\tau]||^2$                    ▷ AF frame loss
        $\boldsymbol{\nabla}[\tau] \leftarrow$ GRAD($\mathcal{L}, \boldsymbol{\theta}$)                           ▷ Filter gradient
        for k $\leftarrow 0$ to $K$ do                                   ▷ Apply update per freq
            $\boldsymbol{\xi}_{\mathrm{k}}[\tau] \leftarrow [\boldsymbol{\nabla}_{\mathrm{k}}[\tau], \mathbf{u}_{\mathrm{k}}[\tau], \mathbf{d}_{\mathrm{k}}[\tau], \mathbf{y}_{\mathrm{k}}[\tau], \mathbf{e}_{\mathrm{k}}[\tau]]$
            $(\boldsymbol{\Delta}_{\mathrm{k}}[\tau], \boldsymbol{\psi}_{\mathrm{k}}[\tau+1]) \leftarrow g_{\boldsymbol{\phi}}(\boldsymbol{\xi}_{\mathrm{k}}[\tau], \boldsymbol{\psi}_{\mathrm{k}}[\tau])$
            $\boldsymbol{\theta}_{\mathrm{k}}[\tau+1] \leftarrow \boldsymbol{\theta}_{\mathrm{k}}[\tau] + \boldsymbol{\Delta}_{\mathrm{k}}[\tau]$
    $\bar{\mathbf{y}} \leftarrow$ CAT($\underline{\mathbf{y}}[\tau], \forall\tau$)                    ▷ Concatenate accumulated frames
    return $\bar{\mathbf{y}}, \boldsymbol{\psi}, h_{\boldsymbol{\theta}}$
function TRAIN($\mathcal{D}$)
    $\boldsymbol{\phi} \leftarrow$ [145] init
    while $\boldsymbol{\phi}$ not CONVERGED do                                  ▷ Train loop
        $\underline{\mathbf{U}}, \underline{\mathbf{d}}_{\mathrm{m}} \leftarrow$ SAMPLE($\mathcal{D}$)                          ▷ Sample signals
        $\boldsymbol{\theta}, \boldsymbol{\psi} \leftarrow \mathbf{0}, \mathbf{0}$                          ▷ Init filter and optimizer state
        for $n \leftarrow 0$ to end do                              ▷ Loop across long signal
            $\bar{\mathbf{U}}, \bar{\mathbf{d}}_{\mathrm{m}} \leftarrow$ NEXTL($\underline{\mathbf{U}}, \underline{\mathbf{d}}_{\mathrm{m}}$)                    ▷ Get next $L$ frames
            $\bar{\mathbf{y}}, \boldsymbol{\psi}, h_{\boldsymbol{\theta}} \leftarrow$ FORWARD($g_{\boldsymbol{\phi}}, \boldsymbol{\psi}, h_{\boldsymbol{\theta}}, \bar{\mathbf{U}}, \bar{\mathbf{d}}_{\mathrm{m}}$)
            $\mathcal{L}_M \leftarrow$ via (3.15)                                ▷ Meta loss
            $\boldsymbol{\nabla} \leftarrow$ GRAD($\mathcal{L}_M, \boldsymbol{\phi}$)                       ▷ Optimizer gradient
            $\boldsymbol{\phi}[n+1] \leftarrow$ METAOPT($\boldsymbol{\phi}[n], \boldsymbol{\nabla}$)                       ▷ Update opt
    return $\hat{\boldsymbol{\phi}}$                                                 ▷ Return best $\boldsymbol{\phi}$
```

performance on a system identification task of various training settings and display results in Fig. 6.3.

Improving meta-learning training methods is an active area of research, with significant work being done on better initialization, training techniques, data considerations, and various other aspects of learned optimizer development. This research is being conducted by both the machine learning [107] and signal processing [89, 90] communities. During the development of this training algorithm, we observed an interesting interplay between these two fields. One notable parallel is the treatment of values with a temporal component. Many machine learning works focus on batch-wise processing, where independence across batches is a reasonable assumption. However, in the context of adaptive filtering, this assumption is inadequate. In fact, the signal from the last frame can serve as a good prediction for the signal in the current frame!

3.5 DEMONSTRATION

In this section, we train a Meta-AF and demonstrate its behavior by generating a synthetic system identification task. We then manipulate several parameters to observe the effects on Meta-AF performance. For a more in-depth analysis of real-world tasks, see Chapter 6.

3.5.1 Demonstration Task Setup

To begin, we create a dataset consisting of input-output pairs. Each pair is generated as follows: we sample new random signals, resulting in 2048 pairs with a length of $N = 1024$ and a filter size of $R = 32$. Specifically,

$$\underline{\mathbf{u}} = \mathcal{N}(\mathbf{0}^N, \mathbf{I}^N) \tag{3.19}$$

$$\underline{\mathbf{w}} = \mathcal{N}(\mathbf{0}^R, \mathbf{I}^R)/R \tag{3.20}$$

$$\underline{\mathbf{d}} = \underline{\mathbf{u}} * \underline{\mathbf{w}}. \tag{3.21}$$

We will use a standard full block OLS filter with a block size of $K = 64$ and a hop of $R = 32$. We train the Meta-AF model using the MSE and equip it with an input linear layer, a single GRU layer, an output linear layer, and a state size of 16. We perform validation on a separate dataset of signals with 128 unseen input/output pairs. For training, we use Adam with a learning rate of 10^{-4}, the frame-independent ISE meta-loss $\mathcal{L}_m$, and set the unroll to L in our training algorithm 3.1. The batch size is set to 16 meaning our training dataset is composed of 128 unique batches which we randomly sample from for 64 epochs. This procedure takes a couple of minutes using just one consumer-grade GPU. We reuse this same procedure in all subsequent “Demonstration” sections.

3.5.2 Training Demonstration

Using the setup from section 3.5.1, we observe the training and validation plots depicted in the top portion of Fig. 3.1. It is evident that the model effectively minimizes the training loss (top left plot) and does not merely memorize the signals from the training set, as shown by its improving performance on the held-out validation (top right plot) set. Notably, the training loss is computed across L updates, while the validation loss is computed across the full signal span of 32 updates. This is a positive outcome, indicating that the model generalizes well to signals longer than those used for training. Furthermore, since the model's performance did not plateau, it suggests that further training could lead to even better

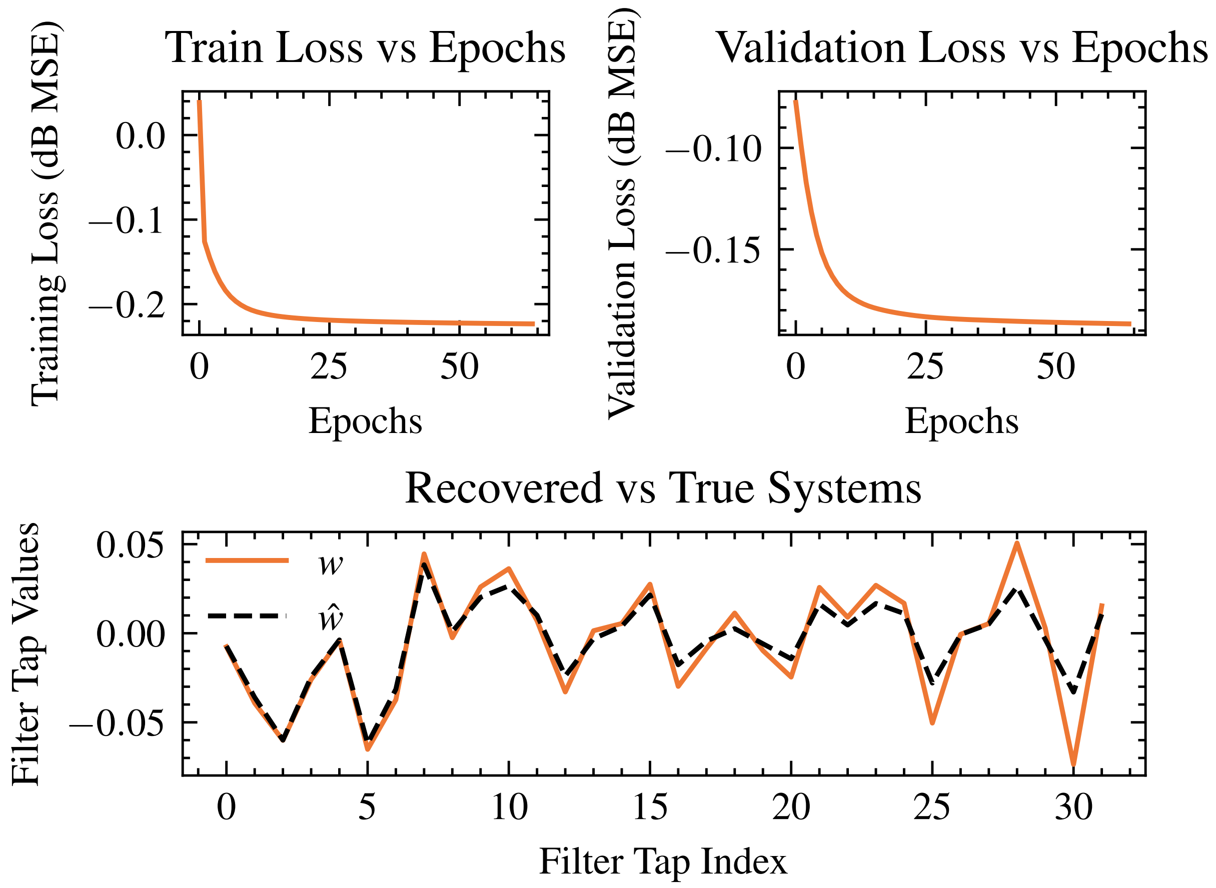


Figure 3.1: Here, we demo Meta-AF on a toy noiseless system identification task. The top two plots track training and validation performance and the bottom plot shows filter parameters recovered on a particular sample in the test set.

results. In summary, our model has learned an online optimization rule tailored to this specific task. To gain insights into the recovered filter parameters of the Meta-AF, we examine the bottom portion of Fig. 3.1, which displays both the recovered time-domain system parameters ($\hat{\mathbf{w}}$) and the true system parameters ($\mathbf{w}$). Encouragingly, we observe that the final recovered filter parameters closely resemble the true parameters. While expected, this is promising since we trained for the systems to have matched outputs and did not explicitly ask for the systems to match.

3.5.3 Meta-Loss Demonstration

Given promising results from our simplest setup, we now compare the accumulated loss with the independent loss. Here, we re-run the same experiment but simply switch to a frame-accumulated time-domain MSE loss. Our convergence results, which track system

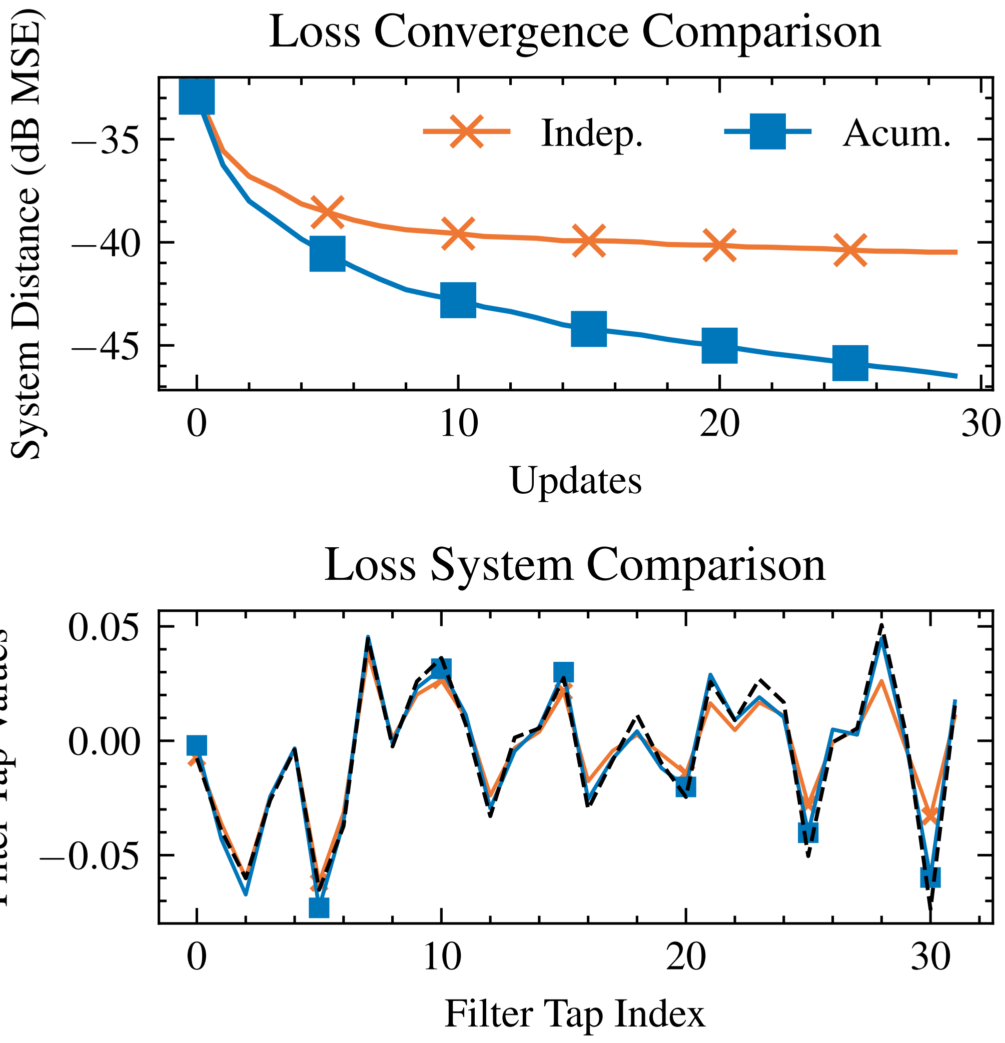


Figure 3.2: Here, we demo Meta-AF on a toy noiseless system identification task and compare accumulated to independent losses. We find the accumulated loss leads to better convergence and peak performance as well as more accurately recovered systems.

distance from the oracle solution are shown via the dB mean squared error of the system distance. We find that accumulated systems typically converge faster and to much better solutions as shown in the top section Fig. 3.2 via a convergence plot and the bottom section of Fig. 3.2 via recovered systems.

3.5.4 Optimizer Scaling Demonstration

Another significant aspect of Meta-AF systems is their scalability. If we desire these models to handle a wide range of signal scenarios or learn more implicit rules from the data,

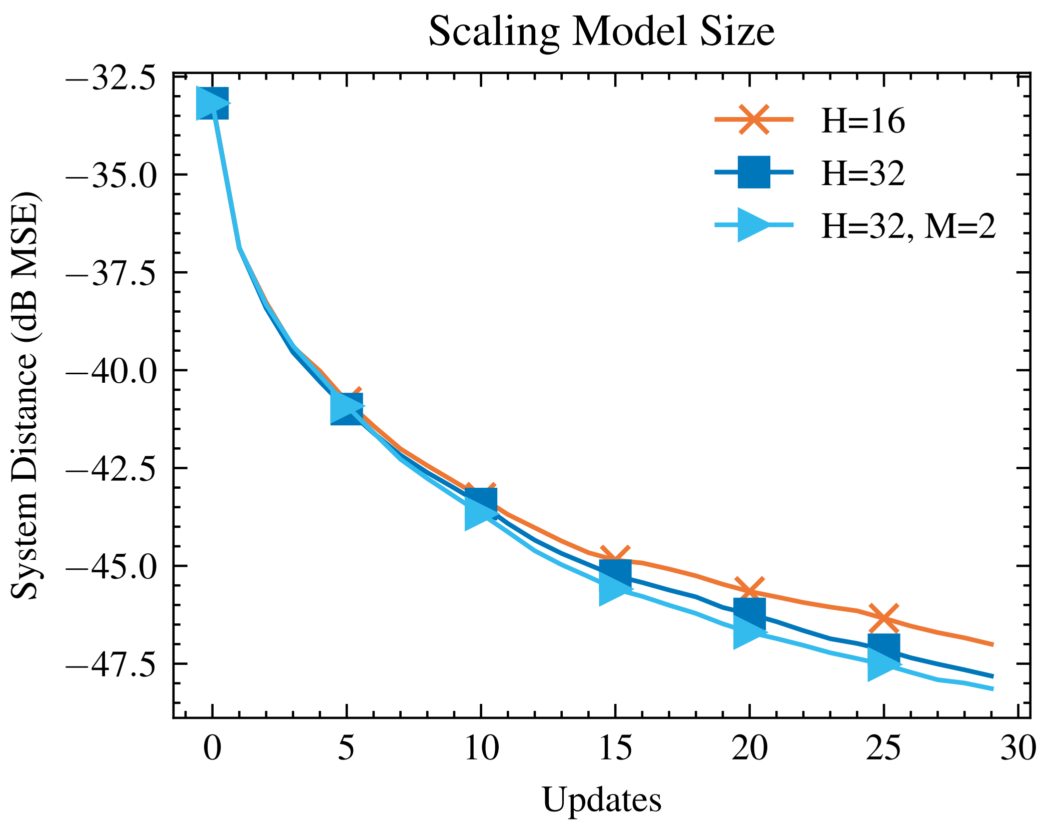


Figure 3.3: Here, we demo training Meta-AF with more parameters and show that this is a simple way to improve performance without modifying the filtering setup.

we can achieve that by simply increasing the size of the optimizer deep neural network, g_ϕ. To illustrate this effect, we present Figure 3.3, where we double the hidden state size from $H = 16$ to $H = 32$, and then to $H = 32$ with two layers effectively quadrupling and then again doubling model size from ≈ 2000 parameters to ≈ 8000 and then a final ≈ 16000 parameters. Training these larger models also only takes a couple of minutes on the GPU. Remarkably, this scaled-up model exhibits improved performance without requiring any modifications to the actual filter.

3.5.5 Optimizer Generalization Demonstration

One notable strength of Meta-AF systems is their ability to acquire and learn favorable traits that are implicitly defined by the data. However, this also poses a challenge when it comes to enforcing known and desired traits. For instance, achieving a balance between convergence speed and peak performance can be problematic. To explore this further, we can apply the same training setup as described above but use longer signals for testing.

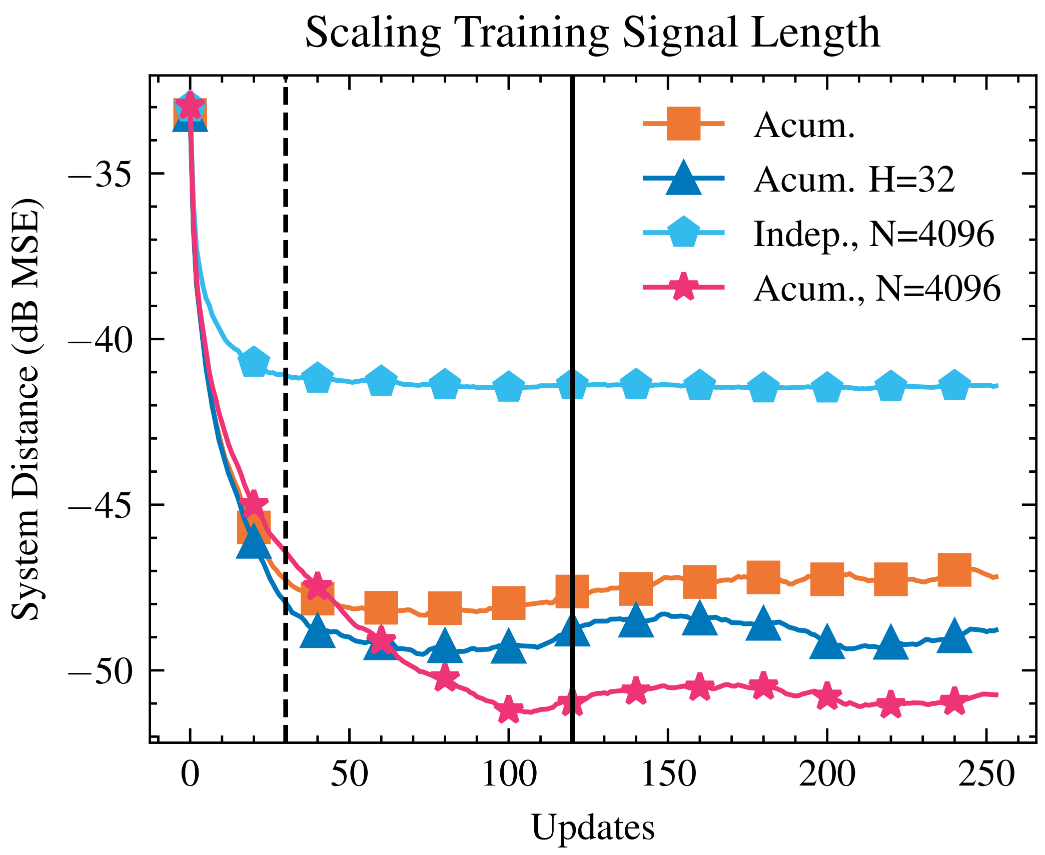


Figure 3.4: Here, we demo training Meta-AF on data that only differs in terms of signal length. This leads to different convergence behaviors. The black dashed line represents the longest signal the $N = 1024$ model saw in training, and the black solid line represents the longest signal the $N = 4096$ model saw in training.

To investigate the effect of longer training signals on the Meta-AF model, we conduct an experiment illustrated in Fig. 3.4. In this experiment, we use the same training scheme as described in 3.5.1, but instead of setting the training signal lengths to $N = 1024$, we increase them to $N = 4096$. We also cut the number of training examples in $\frac{1}{4}$ so the amount of training data is the same. In the plot, we depict the dB mean squared error of the system distance. This modified training scheme aims to elicit a model with a "smaller" learning rate, as the longer data lengths prioritize peak performance over convergence speed. However, the results are more complex than that. Training a model with longer signals does result in slower convergence with higher peak values, but also results in better generalization to longer signals. Possibly the most interesting trend in this plot is that the larger $H = 32$ model achieves better performance on signals longer than those encountered at training, depicting an often observed phenomenon that larger DNN models tend to generalize better. All accumulated loss models outperform the independent loss models, even the independent loss model was trained on the full-length signals. Training Meta-AF on longer signals has multiple dependent effects that lead to interesting behavior.

3.5.6 Demonstration Conclusion

In summary, Meta-AF models have fascinating scaling and training properties and demand a new way to reason about and approach AF problems. We discuss various perspectives in the next section. For a non-toy task and detailed analysis of per-task performance please see Chapter 6.

3.6 OTHER PERSPECTIVES

There are various other perspectives on what Meta-Learned Adaptive Filtering means and what exactly the learned optimizer is doing. We touch on some of them below in an attempt to highlight future research directions and work that we haven't been able to formally write up.

3.6.1 Learning a Loss

Meta-AF is formulated as the process of learning the parameters of a function $g_{\boldsymbol{\phi}}$, which is used to update a filter $h_{\boldsymbol{\theta}}$ through the additive update,

$$\boldsymbol{\theta}_{\mathrm{k}}[\tau+1] = \boldsymbol{\theta}_{\mathrm{k}}[\tau] + \boldsymbol{\Delta}_{\mathrm{k}}[\tau]. \tag{3.22}$$

This formulation bears resemblance to an LMS-like setup,

$$\boldsymbol{\theta}_{\mathrm{k}}[\tau+1] = \boldsymbol{\theta}_{\mathrm{k}}[\tau] + \lambda \nabla \mathcal{L}_{\mathrm{k}}[\tau], \tag{3.23}$$

where the gradient of the loss is scaled and then additively mixed with previous filter parameters. Thus, one can argue that Meta-AF can be viewed as learning the gradient of some implicit loss function for LMS where

$$\boldsymbol{\Delta}_{\mathrm{k}}[\tau] = \lambda \nabla \mathcal{L}_{\mathrm{k}}. \tag{3.24}$$

It could be intriguing to explore the possibility of recovering the implicit $\mathcal{L}$ from the integral of $g_{\boldsymbol{\theta}}$ and further investigate this interpretation. Exploring and understanding the recovered

$\mathcal{L}$ could provide valuable theoretical and practical insights for enhancing the performance and interpretability of Meta-AF algorithms.

3.6.2 Safeguarding the Outputs of the Optimizer

Ensuring the convergence of a learned optimizer, $g_{\boldsymbol{\phi}}$, can be challenging due to the complexity and non-linear nature of the optimization problem. However, we could employ a safeguarding approach [139] that compares the update produced by the learned optimizer with a known and reliable optimizer, such as the conventional LMS algorithm. By leveraging the convergence guarantee of the conventional optimizer, we can establish a lower bound on the behavior of the learned optimizer and enhance its reliability. We describe this procedure below. However, we do not use this setup in our experiments.

The safeguarding process involves comparing the updates produced by both the learned optimizer $g_{\boldsymbol{\phi}}$ and the conventional LMS algorithm. Given the current filter parameters $\boldsymbol{\theta}[\tau]$, we obtain the updates as follows:

$$\boldsymbol{\Delta}_g = g_{\boldsymbol{\phi}}(\boldsymbol{\xi}_{\mathrm{k}}[\tau], \boldsymbol{\psi}_{\mathrm{k}}[\tau]) \tag{3.25}$$

$$\boldsymbol{\Delta}_{\mathrm{LMS}} = \lambda \nabla \mathcal{L}(\ldots), \tag{3.26}$$

where $\boldsymbol{\Delta}_g$ represents the update produced by the learned optimizer, and $\boldsymbol{\Delta}_{\mathrm{LMS}}$ represents the update computed by the conventional LMS algorithm with a step size of λ.

To safeguard the outputs, we choose the update that results in the most significant improvement in terms of the objective function. Specifically, we update the filter parameters $\boldsymbol{\theta}[\tau]$ as:

$$\boldsymbol{\theta}[\tau + 1] = \boldsymbol{\theta}[\tau] + \max(\boldsymbol{\Delta}_g, \boldsymbol{\Delta}_{\mathrm{LMS}}), \tag{3.27}$$

where $\max(\cdot)$ selects the update that maximizes the improvement in the objective function. By comparing the updates and choosing the better one, we establish a lower bound on the performance of the learned optimizer, as it is guaranteed to at least match the behavior of the reliable conventional LMS algorithm. The proof can be broken into two cases, one where LMS update is chosen, which inherits the LMS guarantees, and one where the meta-update is chosen. The meta-update is only chosen when it is better, which means the LMS update is worse and bounds the worst case. It could be interesting to study carrying this same argument out for combinations of adaptive filters [146] or even boosted adaptive filter [147] as well.

Chapter 4: Meta-Adaptive Optimizer Architectures

Framing adaptive filter optimizer design as a learning problem allows us to leverage various viewpoints to enhance performance and streamline the design process. In this section, we will delve into these viewpoints and suggest enhancements, all of which involve making adjustments to specific components within the Meta-AF architecture. In this discussion, we will explore various aspects of optimization, including the design of input features for the optimizer, the interaction between the optimizer and the optimizee, the detailed architecture of the optimizer's DNN, and the operational architecture of the optimizer's execution.

4.1 INPUT FEATURES

A common approach in machine learning is feature engineering, which involves finding a suitable representation or transformation of the data to enhance the performance of a machine learning model. In our context, this corresponds to discovering good signal representations for DNN optimizers to leverage when adapting a filter.

4.1.1 Input Feature Whitening

One important technique is feature whitening or normalization. Whitening the data is a common preprocessing step in machine learning that often improves performance. This performance gain can come from improved numerical stability, better convergence, decorrelation, or robustness. Simply put, it makes things better behaved.

In our work, the dynamic range of input signals, output signals, and gradients can be very high. So, we always re-scale the inputs element-wise via

$$\ln(1+|\boldsymbol{\xi}|)e^{j\angle\boldsymbol{\xi}}. \tag{4.1}$$

This transformation reduces the dynamic range of the signals while preserving their phases. The goal is to approximate a circular complex Gaussian model. An example of the input signal distributions before and after this transformation is shown in Fig. 4.1. The figure shows that after the transform, the magnitudes are closer to a Gaussian, and contain fewer extreme outliers. This preprocessing step has proven useful in several previous works [92, 101], although those works employed explicit clipping, which we have found to be unnecessary. This particular log transformation is a special case of the Box-Cox or power transform. For more rigorous empirical evidence, we provide results on a system identification task

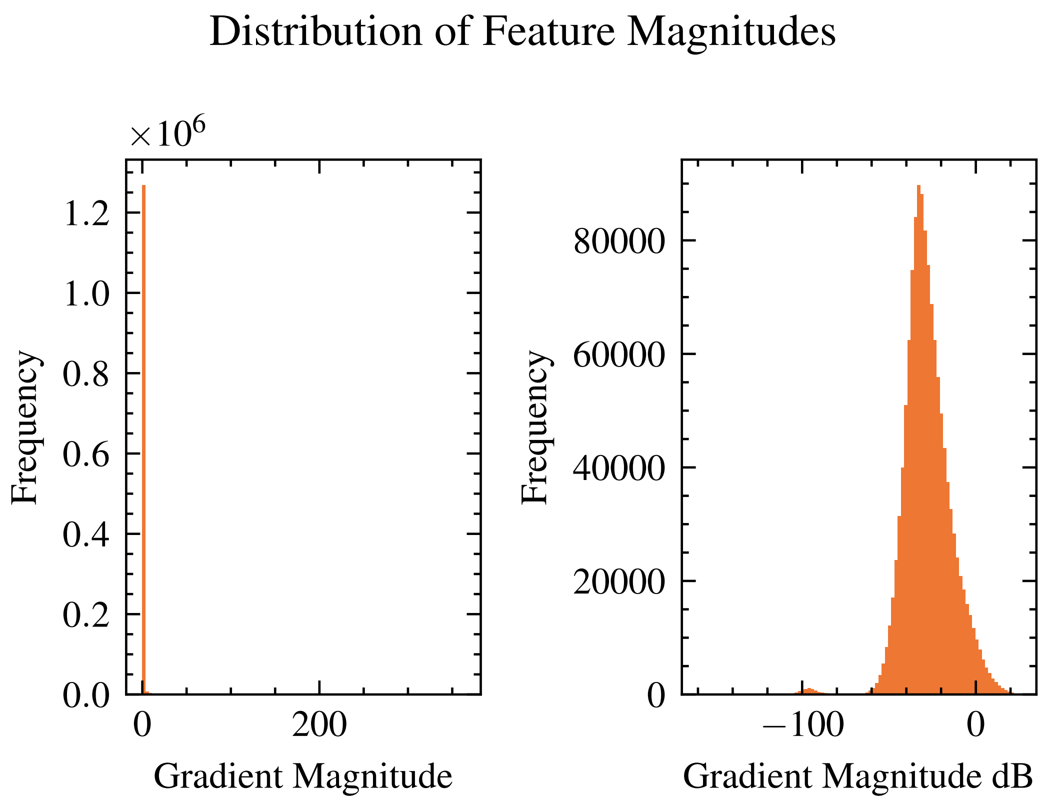


Figure 4.1: Distribution of input features for a system identification task before (left) and after (right) the exponential transform.

in the main ablation of Section 6.1.4, where Fig. 6.3 shows the effect of this transform on performance.

4.1.2 Input Feature Selection

Incorporating domain-specific knowledge into the feature selection process can improve the performance of meta-adaptive filters. By providing additional input features that may not have an analytically established relationship with filter performance, we enable the neural model to learn from them and potentially benefit from their inclusion. Examples of such domain-specific features could include direction of arrival information, voice activity information, user desire information, and more. These features provide valuable context and allow the meta-adaptive filter to adapt its behavior based on specific aspects of the input signals or user requirements.

On the simpler side, this corresponds to selecting input signals that are most relevant to the optimization problem. We break features into two categories: open and closed-loop features. Open loop features at time τ are not affected by optimizer decisions at time $< \tau$. However, closed-loop features are. From the signal set:

$$\boldsymbol{\xi}_{\mathrm{k}}[\tau] = [\nabla_{\mathrm{k}}[\tau], \mathbf{u}_{\mathrm{k}}[\tau], \mathbf{d}_{\mathrm{k}}[\tau], \mathbf{e}_{\mathrm{k}}[\tau], \mathbf{y}_{\mathrm{k}}[\tau]], \tag{4.2}$$

we identify $\nabla, \mathbf{e}, \mathbf{y}$ as closed-loop and $\mathbf{u}, \mathbf{d}$ as open loop. We find that closed-loop features are particularly critical for performance, and show this in the demo below. We revisit these input features in section 4.4, where we motivate a particular combination that leads to the best performance.

4.1.3 Demonstration

In this section, we train a Meta-AF and demonstrate its behavior by generating a synthetic system identification task. We reuse the setup from section 3.5 and the task from section 3.5.1. First, we evaluate the use of feature engineering, specifically that of the log preprocessing which approximately whitens the inputs to the optimizer. We show results in Fig. 4.2 and find that this scheme improves performance in terms of peak and convergence ability. For evidence on real-world tasks, see 6, where the higher dynamic range of real-world signal leads to a much larger effect.

Next, we show the value of providing the right input features as well as the importance of an accompanying training procedure. In Fig. 4.3 we show different combinations of input features, some of which provide feedback in a closed loop manner, some of which are open loop, all accompanied by an improved architecture and training scheme for handling larger feature sets. The improved architecture and training scheme, shown via the orange x's is comparable to the models in Fig. 4.2. To summarize, it consists of various regularization and training tricks which are useful when many input features are being used. The full feature set is composed of both open and closed-loop features. Open-loop features do not depend on the models' past behavior whereas closed-loop features do. Closed-loop features dramatically improve performance, as shown by removing them, which corresponds to the open-loop-only model in blue with squares.

4.2 OPTIMIZEE INTERFACE ARCHITECTURE

The output features, which dictate how the optimizer interfaces with the optimizee, are of critical importance. They can make enforcing constraints and other kinds of properties easier or harder.

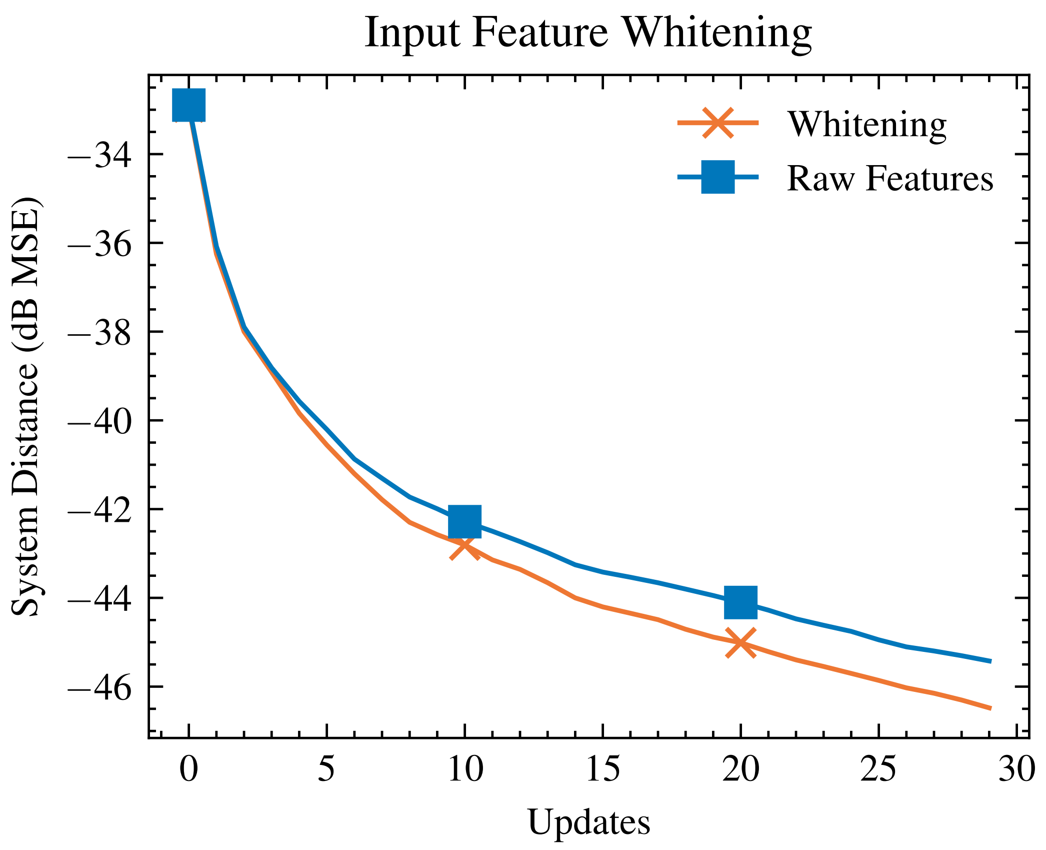


Figure 4.2: Here, we demo Meta-AF on a toy noiseless system identification task and evaluate the use of whitening as a preprocessing step.

4.2.1 Direct Parameterization

Most optimizees can be formulated in terms of their raw frequency domain filter coefficients or in another potentially more structured domain.

Direct Updates: When interfacing with raw coefficients, additive updates as shown in (3.2) and below,

$$\boldsymbol{\theta}[\tau+1] = \boldsymbol{\theta}[\tau] + g_{\boldsymbol{\phi}}(\cdot), \tag{4.3}$$

can be a natural first step. They are simple to implement and achieve a good balance in terms of structure enforced and structure learned.

Direct Prediction: Alternatively, it is possible to directly interface with the optimizee by predicting the filter coefficients from scratch instead of updating them. For example,

$$\boldsymbol{\theta}[\tau+1] = g_{\boldsymbol{\phi}}(\cdot), \tag{4.4}$$

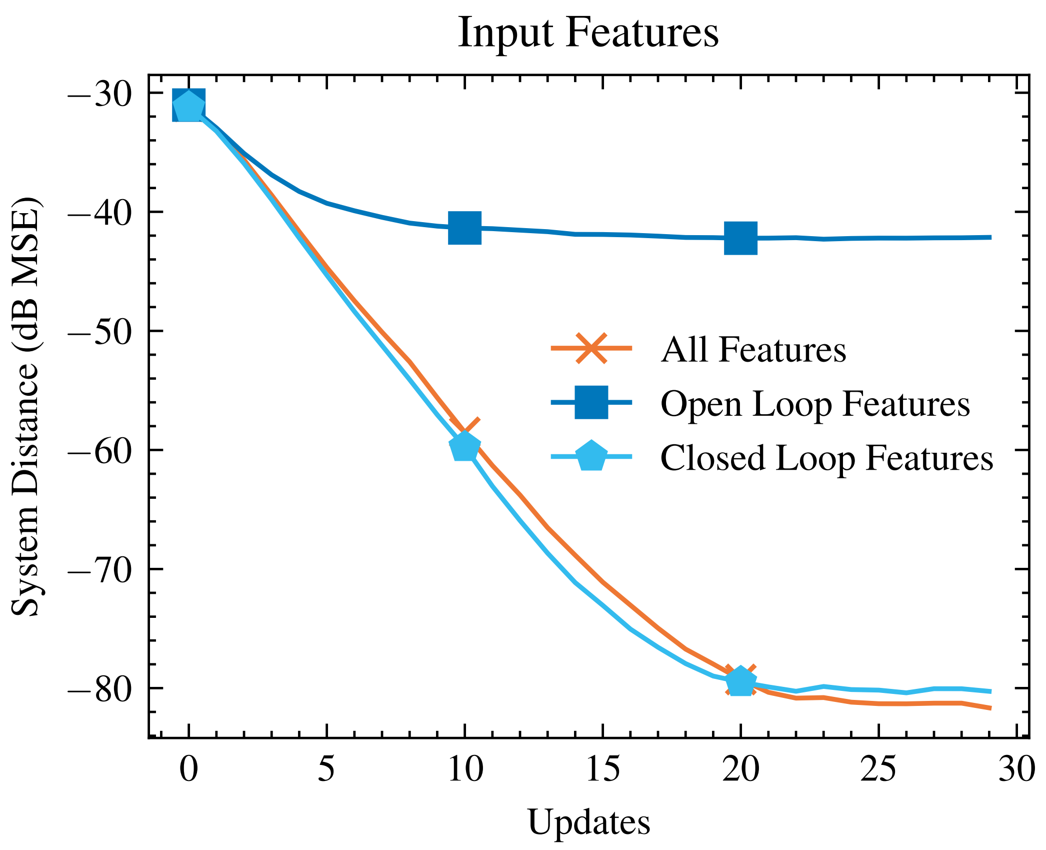


Figure 4.3: Here, we demo Meta-AF on a toy noiseless system identification task and evaluate the use of different input configurations.

depending on the filtering task, different parameterizations work better. Generally, we found that direct prediction works best in scenarios where convergence speed is most important.

4.2.2 Indirect Parameterization

However, it is also possible to parameterize the filters indirectly. One common filter statistic to track and estimate, instead of the raw values, is the per-frequency covariance matrix. Covariance matrices possess unique structures and can be spatial in the case of beamformers or temporal in the case of multi-frame filters. In both cases, covariance matrices have a unique structure.

Rank-1 Updates: One approach to indirect parameterization involves interpreting the network output as a rank-1 symmetric update to an existing $\mathbf{\Phi}^{-1}[\tau-1]$, where we drop the frequency subscript for simplicity. This is accomplished by setting $g_\phi(\cdot)$ to output $\boldsymbol{\varphi}[\tau] \in \mathbb{C}^M$ and computing the update via the fast rank-1 update formula:

$$\boldsymbol{\Phi}^{-1}[\tau] = \boldsymbol{\Phi}^{-1}[\tau-1] - \frac{\boldsymbol{\Phi}^{-1}[\tau-1]\boldsymbol{\varphi}[\tau]\boldsymbol{\varphi}^{\mathsf{H}}[\tau]\boldsymbol{\Phi}^{-1}[\tau-1]}{1+\boldsymbol{\varphi}^{\mathsf{H}}[\tau]\boldsymbol{\Phi}^{-1}[\tau-1]\boldsymbol{\varphi}[\tau]}, \tag{4.5}$$

This rank-1 update preserves most of the structure of a covariance matrix and serves as a natural initial step toward a learned covariance update module.

Cholesky Updates: Another approach allows the network to internally manage updates in a non-linear fashion while ensuring that the output remains a valid inverse covariance. This is achieved by predicting the Cholesky decomposition of the output matrix, where $\boldsymbol{\varphi}[\tau] \in \mathbb{C}^{M\times M}$ is interpreted as a lower triangular matrix within the Cholesky decomposition. To obtain a lower triangular matrix, elements above the diagonal are set to zero, and elements on the diagonal are passed through the absolute value function. The estimate is then given by:

$$\boldsymbol{\Phi}^{-1}[\tau] = \bar{\boldsymbol{\varphi}}[\tau]\bar{\boldsymbol{\varphi}}^{\mathsf{H}}[\tau], \tag{4.6}$$

where $\bar{\boldsymbol{\varphi}}[\tau]$ represents the lower-triangular absolute-value modified $\boldsymbol{\varphi}[\tau]$. This relaxes the rank-1 update constraint while still producing outputs within the manifold of valid covariances.

Arbitrary Updates: Finally, we explore a more flexible approach where all constraints are removed, allowing the optimizer to output an arbitrary square matrix $\boldsymbol{\varphi}[\tau] \in \mathbb{C}^{M\times M}$:

$$\boldsymbol{\Phi}^{-1}[\tau] = \boldsymbol{\varphi}[\tau]. \tag{4.7}$$

We refer to this approach as "Arbitrary" since any structure within the estimate is learned rather than being imposed by construction. It is important to note that arbitrary parameterization of the covariance is not equivalent to arbitrary parameterization of the raw filter values. Selecting the appropriate parameterization style often requires empirical testing or is driven by desired guarantees. Typically, it is easier to enforce model requirements when working in the covariance domain.

The choice of output parameterization in adaptive filters involves important tradeoffs that impact performance and flexibility. Direct updates, where raw filter coefficients are modified, offer a simple and balanced approach, striking a compromise between enforcing structure and learning. Direct prediction, on the other hand, allows for faster convergence but may sacrifice some interpretability. Indirect parameterizations, such as rank-1 updates and Cholesky updates, leverage the structure of covariance matrices to preserve desirable properties, but they impose additional constraints and may require more computational resources. Finally, arbitrary updates provide the greatest indirect flexibility but require careful consideration of the desired structure and may lead to more complex optimization

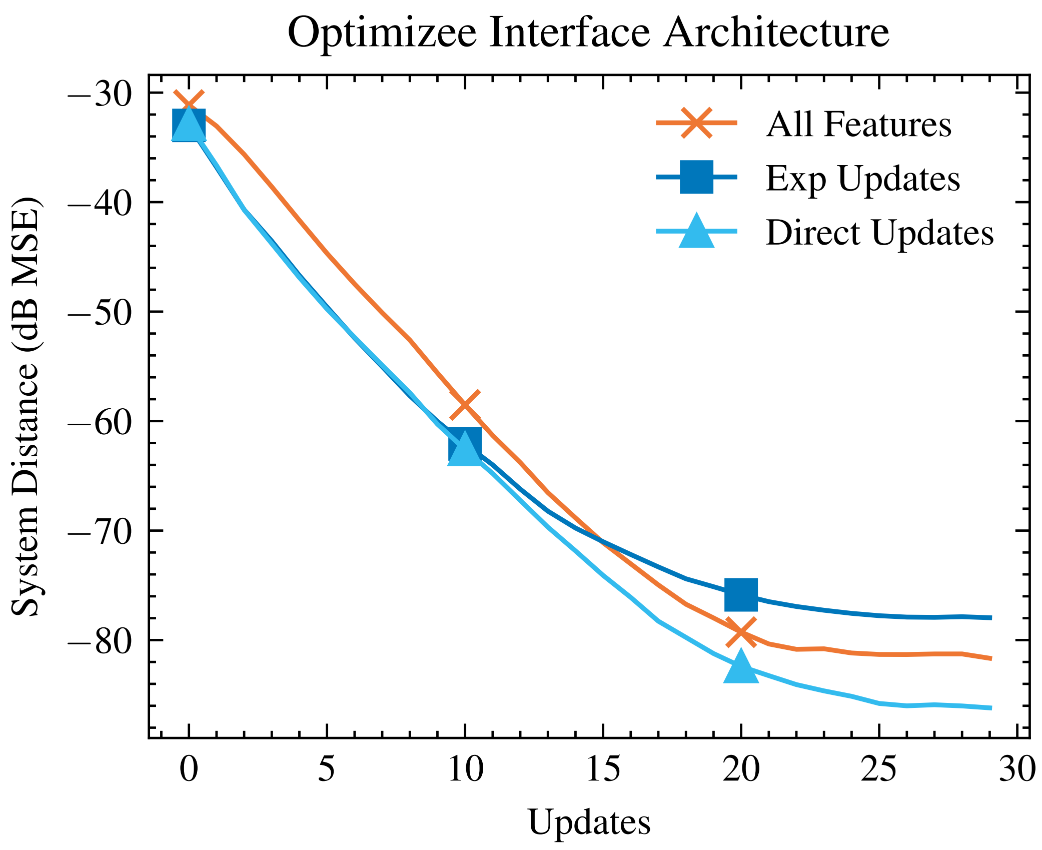


Figure 4.4: Here, we demo Meta-AF on a toy noiseless system identification task and evaluate the use of different optimzee interface configurations.

challenges. Choosing the appropriate output parameterization depends on the specific task requirements, computational constraints, and the desired level of interpretability and control over the learned models.

4.2.3 Demonstration

In this section, we train a Meta-AF and demonstrate its behavior by generating a synthetic system identification task. We reuse the setup from section 3.5 and the task from section 3.5.1. We evaluate various ways for the optimizer to interface with the optimizee. We compare additive updates, exponential parameterization of the additive updates, and a direct prediction scheme. The orange x's is the all-features model from the previous demo. The blue squares are the exponential parameterization of the raw filter coefficients, which do not improve performance here. We found that this helps when the dynamic range of the coefficients is very high. Interestingly, directly predicting the filter parameters (blue up

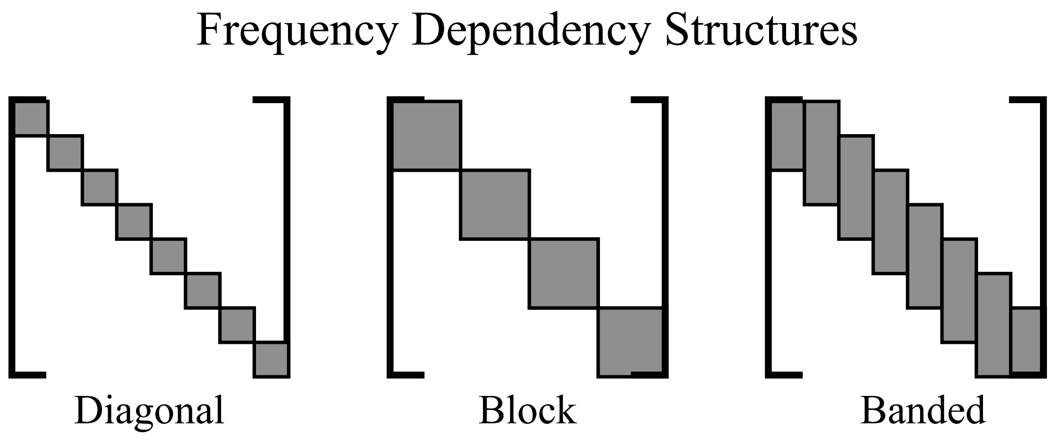


Figure 4.5: Frequency dependency structures for meta-adaptive filters. (Left) Diagonal. (Center) Block. (Right) Banded.

triangle), via a fully learned update scheme improves performance in this scenario. We find this is generally the case when enough data is available.

While clearly, the direct and update approaches have the same range, they have different training properties. The additive nature of the update approach acts as a residual connection. The advantage of this is smoother gradients, but the disadvantage of this it discourages the model from very aggressive updates. Perhaps a hybrid scheme could work best.

4.3 OPTIMIZER NEURAL NETWORK ARCHITECTURE

Here, we treat the task of optimizer design as one of neural network design. When designing neural network models, a common strategy is to consider what relationships exist in your data structure and how to exploit them. One good place to find these is spots where conventional models make simplifying assumptions for tractability. For example, most AF optimizers assume frequency-independent processing, which results in diagonalized and much smaller models. These are typically favorable since they are easier to design. However, using deep learning we don't need to perform explicit design. Instead, we just need to provide the structure to learn that design. Here, we design higher-order meta-adaptive filters, a key improvement to meta-adaptive filters that incorporate higher-order frequency dependencies.

To learn and leverage frequency dependencies, we introduce learnable downsampling $\mathcal{S}$ and upsampling $\mathcal{U}$ layers before and after our optimizer network. The downsampling layer projects the per-frequency inputs $\boldsymbol{\xi}_{\mathrm{k}}[\tau]$ into C groups, where $C \leq K$. We run the optimizer $g_{\boldsymbol{\phi}}$ independently per coupled group c instead of per frequency k and use the upsampling layer to expand the group update $\boldsymbol{\Delta}_{\mathrm{c}}[\tau]$ to a per-frequency update. Formally, we modify (3.11) and (3.12) to be

$$\begin{aligned}(\boldsymbol{\Delta}_{\mathrm{c}}[\tau], \boldsymbol{\psi}_{\mathrm{c}}[\tau+1]) &= g_{\boldsymbol{\phi}}(\ \mathcal{S}(\boldsymbol{\xi}[\tau])_{\mathrm{c}}\ ,\ \boldsymbol{\psi}_{\mathrm{c}}[\tau]\ ) && (4.8)\\ \boldsymbol{\theta}_{\mathrm{k}}[\tau+1] &= \boldsymbol{\theta}_{\mathrm{k}}[\tau] + \mathcal{U}(\boldsymbol{\Delta}_{\mathrm{c}})_{\mathrm{k}}, && (4.9)\end{aligned}$$

where the modified network state $\boldsymbol{\psi}_{\mathrm{c}}[\tau+1]$ stores state per group. By applying $g_{\boldsymbol{\phi}}$ per group of frequencies, we can model interactions within a group and share state/computation within groups, significantly reducing the computational cost. Each shaded square/rectangle in Fig. 4.5 represents a group.

4.3.1 Dependency structures

Our approach allows for arbitrary frequency dependencies by imposing structure into the up-/down- sampling layers. We focus on three different forms of structure as shown in Fig. 4.5, including diagonal (left), block (middle), and banded (right). On an intuitive level, each coupling structure implies a different inter-frequency covariance matrix. Diagonal corresponds to frequency-independent processing, while block/banded model higher-order relationships and allows information sharing across frequency groups. Larger groups model more interactions but at the cost of sharing a single H dimensional state. Thus, there is a trade-off between group size, state size, performance, and efficiency. We describe three potential dependency structures below.

Diagonal: For diagonal frequency dependencies, we set $\mathcal{S}$ and $\mathcal{U}$ to be dense layers operating identically on each frequency. We use this configuration as a baseline as it defaults to [93]. The complexity of $g_{\boldsymbol{\phi}}$ is $\mathcal{O}(\mathrm{H}^2)$, with K executions per frame resulting in a total complexity of $\mathcal{O}(K\mathrm{H}^2)$.

Block: For block frequency dependencies, we reshape each of the K per-frequency network inputs,

$$\boldsymbol{\xi}_{\mathrm{k}}[\tau] = [\nabla_{\mathrm{k}}[\tau], \mathbf{u}_{\mathrm{k}}[\tau], \mathbf{d}_{\mathrm{k}}[\tau], \mathbf{e}_{\mathrm{k}}[\tau], \mathbf{y}_{\mathrm{k}}[\tau]] \in \mathbb{C}^5, \tag{4.10}$$

into C per-group features $\boldsymbol{\xi}_c[\tau] \in \mathbb{C}^{5\mathrm{B}}$ or a $\mathrm{C} \times 5\mathrm{B}$ matrix, where $C = K/B$ and B is the group size. We then apply a dense layer on the latter dimension to produce a $\mathrm{C} \times \mathrm{H}$ output and apply the optimizer separately to each of the C columns. Thus, we impose a non-overlapping block-group structure and enable information and computation sharing within groups. This is reminiscent of sub-band processing [15]. The cost of $g_{\boldsymbol{\phi}}$ is $\mathcal{O}(\mathrm{H}^2)$, the up/down sampling layers cost $\mathcal{O}(\mathrm{BH})$, and the number of executions per frame is $\frac{K}{\mathrm{B}}$, for a total of $\mathcal{O}(\frac{K}{\mathrm{B}}(\mathrm{H}^2 + \mathrm{BH}))$.

Dependencies	Computational Big-$\mathcal{O}$	Space Big-$\mathcal{O}$
Diagonal	$\mathcal{O}(\mathrm{K}(\mathrm{H}^2 + \mathrm{MBH}))$	$\mathcal{O}(\mathrm{KH})$
Block	$\mathcal{O}(\frac{K}{\mathrm{B}}(\mathrm{H}^2 + \mathrm{BH}))$	$\mathcal{O}(\frac{K}{\mathrm{B}}\mathrm{H})$
Banded	$\mathcal{O}(\frac{K}{\mathrm{B}}(\mathrm{H}^2 + \mathrm{BH}))$	$\mathcal{O}(\frac{K}{\mathrm{B}}\mathrm{H})$

Table 4.1: Comparison of different Meta-AF optimizer configurations. Note that all optimizers have about the same number of learnable parameters in $\boldsymbol{\phi}$.

Banded: For banded frequency dependencies, we modify the block-reshape operation described above to return overlapping groups of B frequencies and retain all other block dependency operations. By doing so, we enable information and computation sharing across overlapping groups of frequencies and better model adjacent frequency relationships. By increasing and decreasing the overlap, we modulate the number of adjacent frequencies. In this work, we set the overlap to $\frac{\mathrm{B}}{2}$. This style of dependencies was explored in past work [148, 149]. The complexity of $g_{\boldsymbol{\phi}}$ is $\mathcal{O}(\mathrm{H}^2)$, the up/down sampling layers cost $\mathcal{O}(\mathrm{BH})$, and the number of executions per frame is $2\frac{K}{\mathrm{B}}$ for a total of $\mathcal{O}(\frac{K}{\mathrm{B}}(\mathrm{H}^2 + \mathrm{BH}))$.

We summarize this in the complexity table, Table 4.1, which places these $\mathcal{O}$ compute tradeoffs along with $\mathcal{O}$ space requirements side-by-side. As we will highlight in the experimental section, block, and banded models can both use less compute and perform better than diagonal approaches. Note that banded, while equivalent in compute and memory to block, typically uses twice as many FLOPS and MB. Though, the learned $\boldsymbol{\phi}$s are equivalent in size.

4.3.2 Practical Implementation

Practically, we implement all dependency structures using standard deep-learning operations. We implement the downsampling layer with a 1-D convolution and the upsampling layer with a transposed convolution. We configure different strategies with different filter sizes (B), and stride sizes. A filter size of $\mathrm{B} = 1$ with stride one implements diagonal, a filter size where $\mathrm{B} > 1$ with stride of B implements block, and a filter size of $\mathrm{B} > 1$ and stride of B/2 implements banded.

4.4 SCALABLE ARCHITECTURES

Improving the performance of AFs continues to pose an intricate challenge, requiring a nuanced approach to optimizer design. Consequently, AF algorithm designers have relied on mathematical insights to create tailored optimizers, starting from the foundational development of the least mean squares algorithm (LMS) [150] to the Kalman filter [1, 2, 3, 5, 6].

In contrast, we have witnessed countless remarkable deep learning algorithm advancements in other domains through the principle of “scaling” [151, 152, 153, 154]. The scaling approach involves improving an existing method by deploying additional computational resources. Scaling methodologies are particularly enticing, as they tap into the increasing computational capabilities of modern smart devices, minimizing the need for labor-intensive manual tuning and intervention.

This section described a new online AF method called supervised multi-step AF (SMS-AF). This method integrates a series of algorithm improvements on top of meta-learning methods [93, 94] that together enable scaling performance by increasing model capacity and/or inference cost.

In this section, we describe AFs as operating via the following steps [5]:

$$\begin{aligned} \underline{\mathbf{e}}[\tau] &= h_{\boldsymbol{\theta}[\tau-1]}(\underline{\mathbf{d}}[\tau], \underline{\mathbf{u}}[\tau]) & (4.11)\\ \boldsymbol{\Delta}[\tau] &= g_{\boldsymbol{\phi}}(\boldsymbol{\xi}[\tau], \cdots) & (4.12)\\ \boldsymbol{\theta}[\tau] &= \boldsymbol{\theta}[\tau-1] + \boldsymbol{\Delta}[\tau], & (4.13) \end{aligned}$$

where (4.11) applies the filter, (4.12) updates the optimizer, and (4.13) uses the outputs to update the filter parameters for the next frame. Note, that the optimizer typically inputs filter output $\underline{\mathbf{e}}[\tau]$ created via filter parameters $\boldsymbol{\theta}[\tau-1]$ from the previous frame, creating a feedback loop.

The Kalman filter (KF) extends the AF process above via distinct “predict” and “update” steps. In the KF predict step, (4.11)-(4.13) are run as normal. In the KF update step, however, the filter output is reprocessed after (4.13) using the latest available data:

$$\underline{\mathbf{e}}[\tau] = h_{\boldsymbol{\theta}[\tau]}(\underline{\mathbf{d}}[\tau], \underline{\mathbf{u}}[\tau]). \quad (4.14)$$

As the foundation of our SMS-AF method, we combine Meta-AFs [93] with a higher-order optimizer [94] with per-frequency inputs $\boldsymbol{\xi}_{\mathrm{k}}[\tau]$, and then extend it with three task-agnostic improvements and one task-specific change to scale up. Our complete training and inference methods are summarized in Alg. 4.1.

Our first insight is to use only three key features to control filter adaptation: knowledge of the filter input, final filter output, and filter state. This is deeply motivated by the findings in 4.1.1 and makes use of both closed and open loop features. Compared to past work [93] that uses a large set of inputs,

$$\boldsymbol{\xi}_{\mathrm{k}}[\tau] = [\nabla_{\mathrm{k}}[\tau], \mathbf{u}_{\mathrm{k}}[\tau], \mathbf{d}_{\mathrm{k}}[\tau], \mathbf{e}_{\mathrm{k}}[\tau], \mathbf{y}_{\mathrm{k}}[\tau]], \quad (4.15)$$

where $\nabla_{\mathrm{k}}[\tau]$ is a gradient w.r.t. instantaneous loss $\mathcal{L}(\cdots)$, we use

$$\boldsymbol{\xi}_{\mathrm{k}}[\tau] = [\mathbf{u}_{\mathrm{k}}[\tau], \mathbf{e}_{\mathrm{k}}[\tau], \boldsymbol{\theta}_{\mathrm{k}}[\tau]]. \tag{4.16}$$

This pruning reduces complexity by lowering input dimension and memory requirements for the optimizer, while eliminating inference-time gradient computation w.r.t $\mathcal{L}(\cdots)$, as shown in line 8 of Alg. 4.1. Our second insight is to use a high-quality supervised loss, instead of an unsupervised loss. This opens up several new perspectives, which we cover in more detail in section 5. Previous methods have explored supervised losses such as frame-wise independent supervised losses for echoes [134] or oracle filter parameters [155]. These methods, however, treat frequency bins, adjacent frames, and other channels as distinct optimization entities and have not scaled well [134].

To overcome this, we compute our supervised loss in the time-domain after all AF operations have been performed. This strategy is similar to (5.2), but we use a supervised training signal. Our supervision is non-causal; the loss at τ depends on updates from $< \tau$, enabling the optimizer to learn anticipatory updates. Better loss functions exclusively impact the training phase, without contributing to test-time complexity or data requirements, making this change cost-free for inference. This update corresponds to line 22 of Alg. 4.1.

Our third insight is to leverage the iterative nature of optimizers by executing multiple optimization steps per time frame. By doing so, we offer our optimizers a more powerful feedback mechanism and use the most current parameters for the filter output. Specifically, we run our optimizer update via (4.11)-(4.13), (4.11)-(4.14), or looping over (4.11)-(4.14) multiple times. The first option follows Meta-AF, the second option follows a typical KF and the third extends the KF. We denote the number of (4.11)-(4.13) iterations via C. Incorporating multi-step optimization in Alg. 4.1 involves three changes. First, initializing each frame's filter and optimizer state with results from the last frame (lines $3-4$). Second, iteratively progressing through steps within a frame (line 5), while updating filter parameters/outputs, and optimizer state (lines $8-10$). Last, running a final filter forward pass using the latest parameters (line 11). We find this approach to be a compelling alternative to increasing the dimension of the optimizer, H. Notably, it avoids increasing the parameter count, and it linearly scales complexity, in stark contrast to the quadratic complexity effects associated with H.

Our modifications are a notable departure from the Meta-AF methodology but still aim to learn a neural optimizer for AFs end-to-end. First, by pruning input features and introducing a supervised loss, we eliminate the need for explicit meta-learning, leading to a more streamlined BPTT training process. Second, we replace the past unsupervised loss with a

Algorithm 4.1 Training and inference algorithm.

```
 1: function FORWARD($g_{\boldsymbol{\phi}}, h_{\boldsymbol{\theta}}, \underline{\mathbf{u}}, \underline{\mathbf{d}}_{\mathrm{m}}$)
 2:     for $\tau \leftarrow 0$ to $L$ do
 3:         $\boldsymbol{\theta}_{\mathrm{k}}[\tau] \leftarrow \boldsymbol{\theta}_{\mathrm{k}}[\tau-1]$                    ▷ Initialize with last estimate
 4:         $\boldsymbol{\psi}_{\mathrm{k}}[\tau] \leftarrow \boldsymbol{\psi}_{\mathrm{k}}[\tau-1]$                    ▷ Initialize state with last state
 5:         for c $\leftarrow 0$ to $C$ do                                   ▷ For each predict-update iteration
 6:             $\underline{\mathbf{e}}[\tau] \leftarrow h_{\boldsymbol{\theta}[\tau]}(\underline{\mathbf{d}}[\tau], \underline{\mathbf{u}}[\tau])$
 7:             for k $\leftarrow 0$ to $K$ do                               ▷ Predict step
 8:                 $\boldsymbol{\xi}_{\mathrm{k}}[\tau] \leftarrow [\mathbf{u}_{\mathrm{k}}[\tau], \mathbf{e}_{\mathrm{k}}[\tau], \boldsymbol{\theta}_{\mathrm{k}}[\tau]$
 9:                 $(\boldsymbol{\Delta}_{\mathrm{k}}[\tau], \boldsymbol{\psi}_{\mathrm{k}}[\tau]) \leftarrow g_{\boldsymbol{\phi}}(\boldsymbol{\xi}_{\mathrm{k}}[\tau], \boldsymbol{\psi}_{\mathrm{k}}[\tau])$
10:                 $\boldsymbol{\theta}_{\mathrm{k}}[\tau] \leftarrow \boldsymbol{\theta}_{\mathrm{k}}[\tau] + \boldsymbol{\Delta}_{\mathrm{k}}[\tau]$
11:         $\underline{\mathbf{e}}[\tau] \leftarrow h_{\boldsymbol{\theta}[\tau]}(\underline{\mathbf{d}}[\tau], \underline{\mathbf{u}}[\tau])$                     ▷ Update step
12:     $\bar{\mathbf{e}} \leftarrow$ CAT($\underline{\mathbf{e}}[\tau]\ \forall\tau$)
13:     return $\bar{\mathbf{e}}, \boldsymbol{\psi}[\tau], h_{\boldsymbol{\theta}[\tau]}$
14: function TRAIN($\mathcal{D}$)
15:     $\boldsymbol{\phi} \leftarrow$ [145] init
16:     while $\boldsymbol{\phi}$ not CONVERGED do
17:         $\underline{\mathbf{u}}, \underline{\mathbf{d}}_{\mathrm{m}} \leftarrow$ SAMPLE($\mathcal{D}$)                    ▷ Get batch from dataset $\mathcal{D}$
18:         $\boldsymbol{\theta}, \boldsymbol{\psi} \leftarrow \mathbf{0}, \mathbf{0}$
19:         for $n \leftarrow 0$ to end do
20:             $\underline{\mathbf{u}}, \underline{\mathbf{d}}_{\mathrm{m}} \leftarrow$ NEXTL($\underline{\mathbf{u}}, \underline{\mathbf{d}}_{\mathrm{m}}$)                 ▷ Grab next $L$ frames
21:             $\underline{\mathbf{u}}, \boldsymbol{\psi}, h_{\boldsymbol{\theta}} \leftarrow$ FORWARD($g_{\boldsymbol{\phi}}, \boldsymbol{\psi}, h_{\boldsymbol{\theta}}, \underline{\mathbf{u}}, \underline{\mathbf{d}}_{\mathrm{m}}$)
22:             $L_S \leftarrow \mathcal{L}_S(\cdots)$                          ▷ Task-dependent objective
23:             $\boldsymbol{\nabla} \leftarrow$ GRAD($L_S$, $\boldsymbol{\phi}$)                     ▷ Truncated BPTT
24:             $\boldsymbol{\phi}[n+1] \leftarrow$ METAOPT($\boldsymbol{\phi}[n], \boldsymbol{\nabla}$)            ▷ i.e. run Adam
        return $\hat{\boldsymbol{\phi}}$
```

new, strong supervised signal and loss, helping us scale up. Third, we leverage a multi-step optimization scheme. This creates a generalization of the Kalman filter, where all parameters are entirely learned while retaining explicit prediction and update steps. This also effectively deepens our optimizer networks by sharing parameters across layers in a depth-wise manner. Of particular interest is a modification to the loss function, which we explore more deeply in the next section.

4.5 OTHER PERSPECTIVES

Given the interdisciplinary nature of AFs, there are several other perspectives worth exploring, which may lead to significant improvements and even open up new application areas.

4.5.1 Control Theory

One perspective to consider is the design of AFs from a control theory standpoint, examining open or closed-loop features. Closed-loop feedback is a fundamental principle in control theory, and in the context of adaptive filtering, it becomes essential to provide the filter with information about its performance to effectively adapt to changing conditions. In our initial experiments, incorporating closed-loop feedback has demonstrated substantial performance gains. It is reasonable to assume that other control concepts could also enhance the performance of AFs and provide further improvements. Exploring control theory principles in the design and optimization of AF systems holds promising potential for achieving better adaptation and robustness in various applications.

4.5.2 Graph Signal Processing

Graph signal processing is a field that focuses on analyzing and processing signals defined on graph structures. Traditional AFs typically operate on time-domain or frequency-domain signals, but many real-world problems involve signals defined on complex graphs or networks. A major advantage of meta-learned adaptive filters is that they can learn to operate on arbitrary structures, reducing the large mathematical workload typically associated with leveraging signal processing on graphs. The section on diagonal, block, and banded modules is a primer on this and can be thought of as setting up different adjacency graphs. I suspect more sophisticated connection schemes or even learnable connection schemes could excel in other domains.

4.5.3 Reinforcement Learning

Reinforcement learning offers a promising perspective for adaptive filter design by formulating the adaptation process as an agent-environment interaction and learning optimal policies through trial and error. By treating the adaptive filter as an RL agent, and the optimizer as a controller, one could explore adaptation to more extreme degrees. For example, selecting the number of filter taps, frames, or channels to use all discrete actions. Exploring the synergy between reinforcement learning and adaptive filtering can open up new avenues for developing adaptive systems that excel in challenging and dynamic signal processing tasks.

Chapter 5: Meta-Adaptive Training Objectives

In Section 3.4, we introduced two conceptual flavors of losses: frame-independent losses that can be decomposed across filter updates, and frame-accumulated losses that cannot. Then, we conceptually described the differences between self and full supervision before discussing how full supervision is critical to scaling performance with compute in Section 4.4. In this section, we discuss the construction of these losses based on specific tasks or use cases.

5.1 SELF-SUPERVISION

Self-supervision leverages the inherent structure within a given filter and accompanying data to enable learning without explicit oracle input/output pairs. Adaptive filters, by nature, operate in a self-supervised manner, using signal models to optimize their parameters in real-time without knowledge of the true filter values. This concept can be extended to training optimization rules, where losses are computed over time spans of L filter updates. This allows for the customization of optimizers for specific data distributions, signal classes, and temporal dependencies, potentially leading to superior performance compared to conventional models. Of major note is that these losses are non-causal; the values at τ depend on outputs from $< \tau$. However, the fully trained model is still causal at test-time.

Self-supervision can be applied in both frame-independent and frame-accumulated contexts. In the frame-independent case, the loss is computed for each individual frame. For example, in the case of the ISE objective for system identification, the frame-independent loss is defined as:

$$\mathcal{L}_M = \ln \frac{1}{L} \sum_{\tau}^{\tau+L} E[||\mathbf{d}_{\mathrm{m}}[\tau] - \mathbf{y}_{\mathrm{m}}[\tau]||^2], \tag{5.1}$$

where $\mathbf{y}$ represents the adaptive filter outputs and $\mathbf{d}$ represents the desired responses. The desired responses are often contaminated by noise, distortions, or other signals, depending on the specific application. Self-supervision allows us to train the model using these imperfect but informative desired responses.

Alternatively, the loss can be computed in the time domain, resulting in the frame-accumulated variant:

$$\mathcal{L}_M = \ln E[||\bar{\mathbf{d}}_{\mathrm{m}}[\tau] - \bar{\mathbf{y}}_{\mathrm{m}}[\tau]||^2], \tag{5.2}$$

where $\bar{\mathbf{d}}$ and $\bar{\mathbf{y}}$ represent the accumulated desired responses and filter outputs, respectively. In this case, the accumulated desired response is noisy and contaminated by distortions or other signals.

For example, in the case of acoustic echo cancellation, the self-supervision framework can be applied to train adaptive filters. The desired response in this case is the unknown system output, which is contaminated by noise, speech, and other signals,

$$\underline{\mathbf{d}}[\mathrm{t}] = \sigma(\underline{\mathbf{u}}[\mathrm{t}]) * \underline{\mathbf{w}} + \underline{\mathbf{n}}[\mathrm{t}] + \underline{\mathbf{s}}[\mathrm{t}], \tag{5.3}$$

where $\underline{\mathbf{n}}$ represents noise, $\underline{\mathbf{s}}$ represents speech, $\sigma(\cdot)$ is a nonlinearity, $\underline{\mathbf{w}}$ is an unknown transfer function, and $\underline{\mathbf{u}}$ is the input signal. Despite the presence of noise and other distortions, self-supervision allows us to train the adaptive filter using these imperfect but informative desired responses.

Self-supervision provides a powerful framework for training adaptive filters in a meta-learning context. By leveraging the available information within the data, adaptive filters can learn and adapt effectively in real-world applications. The flexibility of self-supervision allows for the exploration of various loss functions and assumptions, opening up new possibilities for improving adaptive filter performance and addressing complex signal processing tasks. While frame-independent and frame-accumulated losses perform similarly in some tasks, they may differ in others. In the next section on full supervision, we will discuss scenarios where constructing frame-independent losses becomes challenging or infeasible.

5.2 FULL-SUPERVISION

In the case of full-supervision, we have prior knowledge about the expected output of the adaptive filter. This knowledge can be used to guide the training process and optimize the filter's performance. A theme of this section is that better training losses do not introduce additional inference time complexity, making them especially appealing in compute-constrained scenarios because we can reduce the model size while achieving the same level of performance.

5.2.1 Signal-Level Training

One approach in full-supervision is signal-level training, where we have information about the desired signal that should be produced by the adaptive filter. This is particularly applicable when the downstream task involves generating a specific signal. In such cases, accumulated losses are commonly used. One simple form of accumulated loss is the supervised mean squared error (MSE):

$$\mathcal{L}_M = \ln E[||\bar{\mathbf{s}}_{\mathrm{m}}[\tau] - \bar{\mathbf{y}}_{\mathrm{m}}[\tau]||^2], \tag{5.4}$$

where $\bar{\mathbf{s}}_{\mathrm{m}}$ represents the accumulated desired signal and $\bar{\mathbf{y}}_{\mathrm{m}}$ represents the accumulated output of the adaptive filter. The MSE loss quantifies the discrepancy between the desired signal and the output signal, allowing the adaptive filter to adjust its parameters to minimize the error. However, depending on the specific downstream task, other loss functions such as scale-invariant source-to-distortion ratio (SI-SDR), short-time objective intelligibility (STOI), or other task-specific metrics can also be employed. It is important to select a loss function that aligns with the objectives and requirements of the downstream task. Of note is that better losses only contribute to training-time data and compute requirements, making them cost-free in terms of inference.

Full-supervision provides explicit guidance to the training process by incorporating knowledge about the desired output. This approach allows for fine-tuning the adaptive filter to closely match the target signal and optimize its performance for specific tasks. By leveraging different loss functions tailored to the downstream objectives, we can effectively train the adaptive filter for settings and objectives which would be intractable to do by hand either due to complexity or data requirements.

It's worth noting that in some cases, the downstream task itself may involve another model rather than a human listener, and the loss can be designed accordingly to optimize the performance of the subsequent model. In the next section, we discuss losses for such tasks and how to use them.

5.2.2 Classification Training

In some cases, the downstream task is another model and not a listener. In these cases, we can pull the entire training scheme into the meta-loss. For classification-based downstream tasks, this leads to a new class of training paradigms. We call these approaches classification-trained Meta-AF (CT-Meta-AF). CT-Meta-AF learns an AF optimizer for downstream classification tasks in a data-driven manner, without task/model-dependent engineering or the need for oracle signals like clean speech. Specifically, we use an adaptive filter, such as an echo canceller, whose time-varying parameters $\boldsymbol{\theta}[\tau]$ are updated by a meta-learned update rule $g_{\boldsymbol{\phi}}$. The goal is to use the processed signals from the adaptive filter for a downstream classification task, such as keyword spotting, performed by a model $m_{\boldsymbol{\varphi}}$. To this end, we employ an end-to-end training setup that enables joint optimization of the adaptive filter, optimizer, and classification model. Our approach uses feedback from the classifier during training to learn a custom optimizer, resulting in enhanced performance without requiring explicit coupling or the need for test-time feedback. This approach is also suitable for training on real mixtures.

To address the downstream objective, we modify the original meta-loss from (5.2) that incorporates classification feedback, which we call $\mathcal{L}_{CM}$. Specifically, $\mathcal{L}_{CM}$ consists of two components: the classification loss, denoted by $\mathcal{L}_C(c, \hat{c})$, where c is the true class and $\hat{c} = m_{\boldsymbol{\varphi}}(\cdot)$ is the predicted class, and the self-supervised loss, denoted by $\mathcal{L}_M$, as defined in (5.2). The joint loss,

$$\mathcal{L}_{CM}(\bar{\mathbf{e}}, c, \hat{c}) = \lambda \cdot \mathcal{L}_C(c, \hat{c}) + (1 - \lambda) \cdot \mathcal{L}_M(\bar{\mathbf{e}}), \tag{5.5}$$

uses $\lambda \in [0, 1]$ to control the weighting between classification and self-supervised losses. This approach implicitly couples the representations learned by $g_{\boldsymbol{\phi}}$ and $m_{\boldsymbol{\varphi}}$ without explicitly sharing their parameters, using the idea of "task-splitting" [85, 156]. The hyperparameter λ needs to be tuned. However, we find that its value is not critical as long as it is greater than 0 (the no classification feedback setting). Optimizing an AF for classification has interesting ramifications. Typically, it is important to avoid artifacts. Here, however, they don't matter if the classifier can ignore them.

Pretrained Classifier When dealing with complex classification tasks, it is common to use off-the-shelf or pre-trained models due to data, compute, or engineering constraints. In this setting, we assume that the classifier, $m_{\boldsymbol{\varphi}}$, is frozen and not trainable. This leads to a modified (3.3),

$$\hat{\boldsymbol{\phi}} = \arg\min_{\boldsymbol{\phi}} E_{\mathcal{D}}[\ \mathcal{L}_{CM}(\ m_{\boldsymbol{\varphi}}, g_{\boldsymbol{\phi}}, \mathcal{L}(h_{\boldsymbol{\theta}}, \cdots)\ )\ ]. \tag{5.6}$$

This equation incorporates $m_{\boldsymbol{\varphi}}$ without modifying it by using $m_{\boldsymbol{\varphi}}$ as an additional loss term. This method allows custom training of the optimizer for the classifier without the classifier needing to be aware of the custom preprocessing. It customizes the optimizer to the classifier, automatically performing the model unification proposed by Seltzer et al. [73] in a model and task-independent fashion.

Jointly Trained Classifier When ample data and compute resources are available, joint or end-to-end training is a powerful approach. This involves training both the optimizer $g_{\boldsymbol{\phi}}$ and classifier $m_{\boldsymbol{\varphi}}$ simultaneously using the modified meta-loss function $\mathcal{L}_{CM}$. The parameters of the optimizer and classifier, $\boldsymbol{\phi}$ and $\boldsymbol{\varphi}$, are jointly optimized via

$$\hat{\boldsymbol{\phi}}, \hat{\boldsymbol{\varphi}} = \arg\min_{\boldsymbol{\phi}, \boldsymbol{\varphi}} E_{\mathcal{D}}[\ \mathcal{L}_{CM}(\ m_{\boldsymbol{\varphi}}, g_{\boldsymbol{\phi}}, \mathcal{L}(h_{\boldsymbol{\theta}}, \cdots)\ )\ ]. \tag{5.7}$$

During joint training, classification feedback is used to adjust the parameters of the opti-

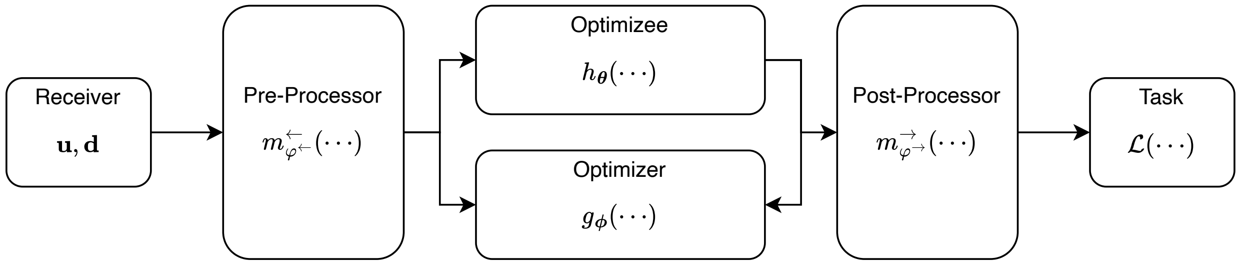

Receiver
$\mathbf{u}, \mathbf{d}$
Pre-Processor
$m^{\leftarrow}_{\varphi^{\leftarrow}}(\cdots)$
Optimizee
$h_{\boldsymbol{\theta}}(\cdots)$
Optimizer
$g_{\boldsymbol{\phi}}(\cdots)$
Post-Processor
$m^{\rightarrow}_{\varphi^{\rightarrow}}(\cdots)$
Task
$\mathcal{L}(\cdots)$

Figure 5.1: A generic AF pipeline composed of pre-processor, optimizee, optimizer, and post-processor. All components are connected to solve some overall task such as recognizing speech in a noisy moving vehicle.

mizer, improving its ability to generate outputs that aid in classification. At the same time, the classifier learns to operate on AF outputs. This builds upon the pre-trained classifier setup by producing a custom-trained classifier that can better leverage the AF. To speed up training, we can initialize $\boldsymbol{\phi}$ and $\boldsymbol{\varphi}$ with pre-trained weights. It is also possible to improve performance by allowing $m_{\boldsymbol{\varphi}}$ to use information beyond the outputs of $h_{\boldsymbol{\theta}}$.

5.3 END-TO-END TRAINING

With AF systems framed using fully supervised learning with potentially learned loss functions, we can now explore pre-/post-processors or complete AF pipelines. This highlights the fact that AF systems are typically integrated into larger pipelines, and optimizing individual components in isolation may not yield the best overall system performance. Fig. 5.1 illustrates a generic pipeline structure. The modules involved in the pipeline can encompass various tasks such as reference estimation, transfer function tracking, speech enhancement, keyword spotting, or automatic speech recognition. We denote these pre-/post-modules as $m^{\leftarrow}_{\varphi^{\leftarrow}}(\cdots)$ and $m^{\rightarrow}_{\varphi^{\rightarrow}}(\cdots)$, which can be tailored to specific tasks. It's worth noting that multiple AFs can be cascaded, with the first AF acting as a pre-processor for a secondary AF.

The discussion of self-supervision, full-supervision, and the study of classification-based training with joint-trained classifiers all angle towards a single unified idea: That all processing modules within a system should be developed in a unified or joint fashion. This culminates in the idea of end-to-end training, which attempts to learn all modules of a processing pipeline jointly,

$$\hat{\boldsymbol{\phi}}, \hat{\boldsymbol{\varphi}}^{\leftarrow}, \hat{\boldsymbol{\varphi}}^{\rightarrow} = \arg \min_{\boldsymbol{\phi}, \boldsymbol{\varphi}^{\leftarrow}, \boldsymbol{\varphi}^{\rightarrow}} E_{\mathcal{D}}[\, \mathcal{L}_{CM}(\, m^{\leftarrow}_{\boldsymbol{\varphi}^{\leftarrow}}, g_{\boldsymbol{\phi}}, m^{\rightarrow}_{\boldsymbol{\varphi}^{\rightarrow}}, \mathcal{L}(h_{\boldsymbol{\theta}}, \cdots)\,)\,]. \tag{5.8}$$

Here, model parameters are optimized simultaneously, leading to a higher-performing system with processing that is aware of the preceding and following operations.

When dealing with an AF system comprising multiple components, each component may possess its local optima when trained independently. However, it is crucial to acknowledge that optimizing each component separately does not guarantee achieving the global optimum for the overall system. This is especially significant in large-scale systems where the interactions between components can exhibit high complexity.

In contrast, by jointly optimizing the entire system, we explore a global optimization landscape that extends beyond the constraints of individual component landscapes. This approach opens up the potential for discovering solutions that may not be attainable through isolated component optimization. In essence, end-to-end training allows us to escape local optima that are specific to individual components and instead pursue more advantageous solutions that maximize the overall system performance.

This advantage becomes particularly relevant in large-scale AF systems, where the interactions between components introduce intricate and nonlinear relationships. The process of joint optimization takes into account these complex dependencies, enabling the exploration of global optima that lead to superior system performance. A key advantage of end-to-end training is that it does not incur any additional inference time complexity. We explore an example of this in Section 6 where we train a multi-stage speech enhancement pipeline.

5.4 REGULARIZATION

Whether training smaller simpler self-supervised Meta-AF models or much larger end-to-end models, regularization can have a large effect on performance. Here we discuss two kinds of training time regularization, one which operates on the outputs of the optimizer, and one which operates on the optimizer itself.

5.4.1 Temporal Consistency Regularization

A significant challenge in integrating powerful neural optimizers with flexible filters is the potential for over-adaptation. The filter and its associated loss are typically based on a specific signal model, designed to address noise or distortion within that model's framework. Neural optimizers, on the other hand, possess the capability to compensate for modeling inaccuracies and extend beyond the boundaries of the signal model. However, this can lead to over-adaptation, which may result in undesirable side effects.

Algorithm 5.1 Training and inference algorithm with temporal consistency.

```
1: function FORWARD($g_{\boldsymbol{\phi}}, h_{\boldsymbol{\theta}}, \underline{\mathbf{u}}, \underline{\mathbf{d}}_{\mathrm{m}}$)
2:     for $\tau \leftarrow 0$ to $L$ do
3:         $\boldsymbol{\theta}_{\mathrm{k}}[\tau] \leftarrow \boldsymbol{\theta}_{\mathrm{k}}[\tau-1]$                ▷ Initialize with last estimate
4:         $\boldsymbol{\psi}_{\mathrm{k}}[\tau] \leftarrow \boldsymbol{\psi}_{\mathrm{k}}[\tau-1]$                ▷ Initialize state with last state
5:         for c $\leftarrow 0$ to $C$ do                ▷ For each predict-update iteration
6:             $\underline{\mathbf{e}}[\tau] \leftarrow h_{\boldsymbol{\theta}[\tau]}(\underline{\mathbf{d}}[\tau], \underline{\mathbf{u}}[\tau])$
7:             for k $\leftarrow 0$ to $K$ do                ▷ Predict step
8:                 $\boldsymbol{\xi}_{\mathrm{k}}[\tau] \leftarrow [\mathbf{u}_{\mathrm{k}}[\tau], \mathbf{e}_{\mathrm{k}}[\tau], \boldsymbol{\theta}_{\mathrm{k}}[\tau]$
9:                 $(\boldsymbol{\Delta}_{\mathrm{k}}[\tau], \boldsymbol{\psi}_{\mathrm{k}}[\tau]) \leftarrow g_{\boldsymbol{\phi}}(\boldsymbol{\xi}_{\mathrm{k}}[\tau], \boldsymbol{\psi}_{\mathrm{k}}[\tau])$
10:                $c_{tc} \leftarrow$ Bernoulli(p)                ▷ Randomly sample
11:                if $c_{tc} == 1$ then                ▷ Apply the update
12:                    $\boldsymbol{\theta}_{\mathrm{k}}[\tau] \leftarrow \boldsymbol{\theta}_{\mathrm{k}}[\tau] + \boldsymbol{\Delta}_{\mathrm{k}}[\tau]$
13:                else                ▷ Skip the update
14:                    $\boldsymbol{\theta}_{\mathrm{k}}[\tau] \leftarrow \boldsymbol{\theta}_{\mathrm{k}}[\tau]$
15:        $\underline{\mathbf{e}}[\tau] \leftarrow h_{\boldsymbol{\theta}[\tau]}(\underline{\mathbf{d}}[\tau], \underline{\mathbf{u}}[\tau])$                ▷ Update step
16:    $\bar{\mathbf{e}} \leftarrow$ CAT($\underline{\mathbf{e}}[\tau]\ \forall\tau$)
17:    return $\bar{\mathbf{e}}, \boldsymbol{\psi}[\tau], h_{\boldsymbol{\theta}[\tau]}$
```

To mitigate the risk of overfitting, we have developed a simple yet effective technique called temporal consistency regularization. This approach involves randomly skipping updates from the optimizer with a certain probability, denoted as p (typically set around 0.01). The rationale behind this strategy is to prevent the optimizer from excessively fitting to any specific frame. By introducing randomness in the update process, we promote a more generalized adaptation behavior.

To incorporate this technique into the multi-step optimization scheme, we propose the modified algorithm shown in Algorithm 5.1. This adaptation ensures that the optimizer's updates are occasionally skipped, introducing a level of regularization and preventing over-adaptation. Our experiments have shown that this approach is particularly beneficial for filters that require rapid adaptation.

The temporal consistency scheme is only used at training time, not at test time. We dropped the update as a whole but did experiment with dropping individual frequencies or bins and found that to be less effective. Our most significant finding was that this was able to remove clicking artifacts from a OLS filter trained of AEC when trained with both predict and update steps.

5.4.2 Randomized Unroll Regularization

The choice of setting the unroll length during training is a complex decision. If the unroll length is too short, the optimizer may struggle to capture long-term dependencies in the data. On the other hand, if the unroll length is too long, the training process can become unstable and excessively time-consuming. To address this challenge, we have developed a simple randomization scheme that effectively randomizes the unroll length, allowing for an implicit selection of shorter unrolls without sacrificing the opportunity to train on longer-term information.

The technique works by introducing a randomization process to determine whether or not to use the stop-gradient ($\text{SG}(\cdot)$) operator. During the forward pass, the function $\text{SG}(\cdot)$ acts as an identity function, while during the backward pass, it acts as a zero function. The unroll length, denoted as L, is set to a large value such as 128, and a small probability p, typically around 0.05, is defined. This setup creates an unroll distribution instead of a fixed unroll length. More complex unroll distributions can be defined, but we have found this approach to be simple to implement and enables embarrassingly parallel batch processing, providing a significant advantage.

To incorporate this technique into the multi-step optimization scheme, we propose the modified algorithm outlined in Algorithm 5.2. This randomized unroll scheme eliminates the need to fine-tune the unroll length L and almost always improves performance. It provides the flexibility to explore shorter unrolls while still ensuring exposure to longer-term information during the training process

By incorporating the randomized unroll scheme, we eliminate the need for manual tuning of the unroll length and achieve improved performance. This technique enables the algorithm to dynamically explore shorter unrolls while still benefiting from exposure to longer-term information. Furthermore, it ensures that the training process remains efficient and can be easily parallelized, offering a significant advantage in batch-processing scenarios.

5.4.3 Demonstration

In this section, we train a Meta-AF and demonstrate the role of regularization by generating a synthetic system identification task. We reuse the setup from section 3.5 and the task from section 3.5.1. We provide a simple study of regularization using both temporal consistency and randomized unrolls in Fig. 5.2.

In Fig. 5.2 we train a model using a 32-step unroll, and also double the number of epochs so that the number of training updates is the same. We find that both regularizers improve

Algorithm 5.2 Training and inference algorithm with random unroll.

```
 1: function FORWARD($g_{\boldsymbol{\phi}}, h_{\boldsymbol{\theta}}, \underline{\mathbf{u}}, \underline{\mathbf{d}}_{\mathrm{m}}$)
 2:     for $\tau \leftarrow 0$ to $L$ do
 3:         $\boldsymbol{\theta}_{\mathrm{k}}[\tau] \leftarrow \boldsymbol{\theta}_{\mathrm{k}}[\tau-1]$                        ▷ Initialize with last estimate
 4:         $\boldsymbol{\psi}_{\mathrm{k}}[\tau] \leftarrow \boldsymbol{\psi}_{\mathrm{k}}[\tau-1]$                        ▷ Initialize state with last state
 5:         for c $\leftarrow 0$ to $C$ do                          ▷ For each predict-update iteration
 6:             $\underline{\mathbf{e}}[\tau] \leftarrow h_{\boldsymbol{\theta}[\tau]}(\underline{\mathbf{d}}[\tau], \underline{\mathbf{u}}[\tau])$
 7:             for k $\leftarrow 0$ to $K$ do                                    ▷ Predict step
 8:                 $\boldsymbol{\xi}_{\mathrm{k}}[\tau] \leftarrow [\mathbf{u}_{\mathrm{k}}[\tau], \mathbf{e}_{\mathrm{k}}[\tau], \boldsymbol{\theta}_{\mathrm{k}}[\tau]$
 9:                 $(\boldsymbol{\Delta}_{\mathrm{k}}[\tau], \boldsymbol{\psi}_{\mathrm{k}}[\tau]) \leftarrow g_{\boldsymbol{\phi}}(\boldsymbol{\xi}_{\mathrm{k}}[\tau], \boldsymbol{\psi}_{\mathrm{k}}[\tau])$
10:                 $c_u \leftarrow$ Bernoulli(p)                          ▷ Randomly sample
11:                 if $c_u == 1$ then                                ▷ Apply the update
12:                     $\boldsymbol{\theta}_{\mathrm{k}}[\tau] \leftarrow \boldsymbol{\theta}_{\mathrm{k}}[\tau] + \boldsymbol{\Delta}_{\mathrm{k}}[\tau]$
13:                 else                                 ▷ Apply update but zero gradients
14:                     $\boldsymbol{\theta}_{\mathrm{k}}[\tau] \leftarrow \boldsymbol{\theta}_{\mathrm{k}}[\tau] + \mathrm{SG}(\boldsymbol{\Delta}_{\mathrm{k}}[\tau])$
15:         $\underline{\mathbf{e}}[\tau] \leftarrow h_{\boldsymbol{\theta}[\tau]}(\underline{\mathbf{d}}[\tau], \underline{\mathbf{u}}[\tau])$                              ▷ Update step
16:     $\bar{\mathbf{e}} \leftarrow$ CAT($\underline{\mathbf{e}}[\tau]\ \forall\tau$)
17:     return $\bar{\mathbf{e}}, \boldsymbol{\psi}[\tau], h_{\boldsymbol{\theta}[\tau]}$
```

convergence performance but not peak performance for this toy task. Though, empirically we observe that these regularizers become more important as the unroll is increased or for more challenging tasks. Of note is that the regularized models bottom out at about -70dB, which is about as high performing as we could expect a model to be in a realistic scenario. As such, we find they always improve performance on real tasks. One particular strength is that they do not add any additional inference complexity and are drop-in compatible with any AF task.

5.5 OTHER PERSPECTIVES

The importance of losses and data is only magnified when using a Meta-AF. Interestingly, there are various interesting perspectives on this dependence.

5.5.1 Distilling the Meta-Loss

The objective of Meta-AF is to minimize a meta-loss function $\mathcal{L}_M$ across a given dataset in expectation. The meta-loss, being based on acausal information, potentially includes oracle knowledge or other types of information that are not attainable during the inference phase. Consequently, this meta-loss effectively distills oracle knowledge into the learned function

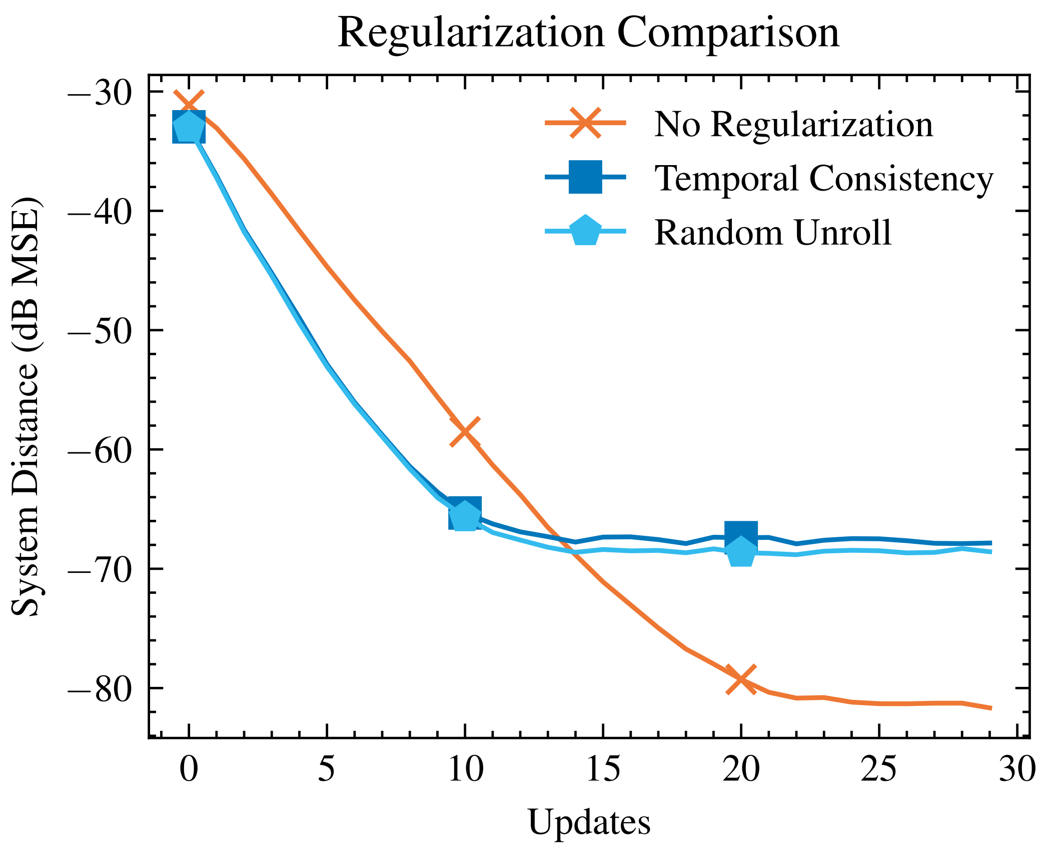


Figure 5.2: Here, we demo Meta-AF on a toy noiseless system identification task and different kinds of regularization including a temporal consistency regularizer and a randomized unroll regularizer.

g_ϕ. From this perspective, the meta-training procedure trains g_ϕ to solve the meta-loss $\mathcal{L}_M$ using only mixtures instead of relying on oracle signals. This learned function generalizes and effectively captures the knowledge distilled from the meta-loss. This approach allows for the adaptation and improvement of the adaptive filter optimization process, even when oracle information is not available during inference.

Exploring different forms and properties of the meta-loss function $\mathcal{L}_M$, as well as understanding the impact of the meta-training procedure on the performance and generalization of the learned function g_ϕ, holds great potential for advancing the field of adaptive filter theory and its practical applications.

Chapter 6: Adaptive Filter Gym

With our conceptual meta-framework in place, we can move to evaluation. We seek to evaluate our claims of developing general optimizers using our conceptual Meta-AF framework by using the `meta-af` package. We evaluate using a collection of datasets and task implementations, which we call the `AF-Gym`. The `AF-Gym` serves as a testing ground for AF. It contains implementations of a suite of AFs as well as simulated scenarios and other already established data sources. We open-source the gym, generation code, and all meta-learned modules at `https://github.com/adobe-research/MetaAF`. Below, we describe evaluation metrics, constituent tasks, and results one by one. To the best of our knowledge, the `AF-Gym` is the first dataset that enables the development of end-to-end AF telephony pipelines.

6.1 META-AF DESIGN AND SYSTEM IDENTIFICATION

To ablate the design and construction of Meta-AF, we select a system identification problem. Through this task, we display initial Meta-AF configurations and reason about design. We conclude with experiments on more advanced Meta-AF architectures and highlight various unique properties. This section is comparable to a more thorough version of the demonstration experiments from section 3.5. The prior works typically build throughout each section.

6.1.1 System Identification

In system identification, the goal is to recover the transfer function (unknown system) between a source and receiver, as shown in Fig. 6.1. This is a common task across modalities from radio-frequency communication to acoustics. Acoustic system identification is a critical component of spatial rendering pipelines which often require head-related transfer functions to mimic spatialization. In the AF-Gym, we model the unknown system (e.g. acoustic room) with a linear frequency-domain filter $h_{\boldsymbol{\theta}}$ (optimizee architecture), inject input signal $\mathbf{u}$ into the system, measure the response $\mathbf{d}$, and adapt the filter weights $\boldsymbol{\theta} = \{\mathbf{w}\}$ using $g_{\boldsymbol{\phi}}$. The AF loss is task dependent but is often the ISE between the desired response, $\mathbf{d}_{\mathrm{k}}[\tau]$, and the AF predicted response, $\mathbf{y}_{\mathrm{k}}[\tau] = \mathbf{w}_{\mathrm{k}}[\tau]^{\mathsf{H}}\mathbf{u}_{\mathrm{k}}[\tau]$. In this task, we evaluate performance by simple signal-level metrics that describe how accurately the output of the system was predicted.

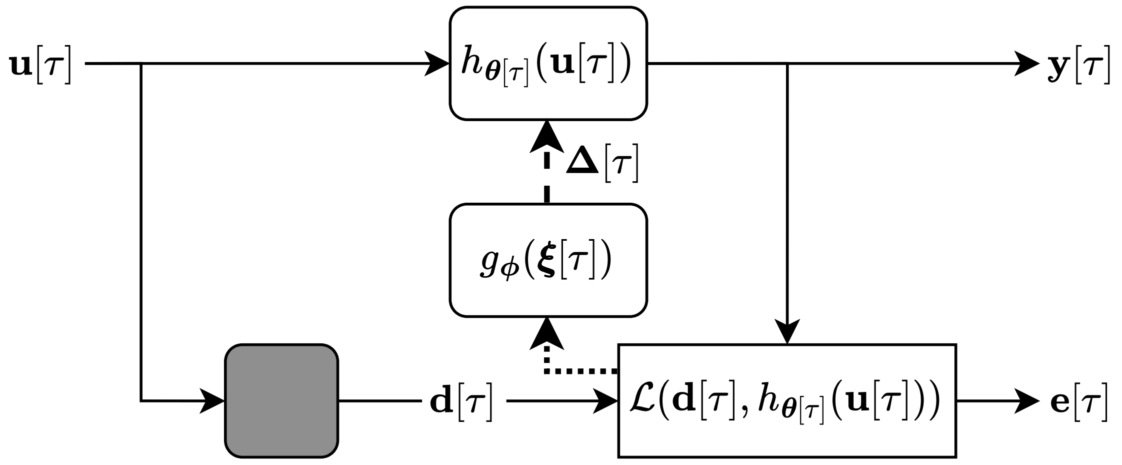

$\mathbf{u}[\tau]$
$h_{\boldsymbol{\theta}[\tau]}(\mathbf{u}[\tau])$
$\mathbf{y}[\tau]$
$\boldsymbol{\Delta}[\tau]$
$g_{\boldsymbol{\phi}}(\boldsymbol{\xi}[\tau])$
$\mathbf{d}[\tau]$
$\mathcal{L}(\mathbf{d}[\tau], h_{\boldsymbol{\theta}[\tau]}(\mathbf{u}[\tau]))$
$\mathbf{e}[\tau]$

Figure 6.1: System identification block diagram. System inputs are fed to both the adaptive filter and the true system (shaded box). The adaptive filter is updated to mimic the true system.

6.1.2 Prior Works

Initial work on automatically tuning AFs has been explored in incremental delta-bar-delta [95], Autostep [96], and elsewhere. Recent machine learning work discusses the idea of using DNNs to learn entirely novel optimizer update rules from scratch [101, 102, 103, 104]. We take inspiration from this work but make numerous advances specific to AFs. In particular, past learned optimizers [101] are element-wise, offline, real-valued, only a function of the gradient, and are trained to optimize general-purpose neural networks. In contrast, we design *online* AF optimizers that use multiple input signals, are complex-valued, adapt block FD linear filters, and integrate domain-specific insights to reduce complexity and improve performance (coupling across channels and time). Moreover, we deploy learned optimizers to solve AF tasks as the end goal and do not use them to train downstream neural networks. We also note recent work that uses a supervised DNN to control the step-size of a D-KF for AEC [87] and another that uses a supervised DNN to predict both the step-size and a nonlinear reference signal for AEC [88]. Compared to these, we replace the entire update with a neural network, do not need supervisory signals, and investigate many tasks.

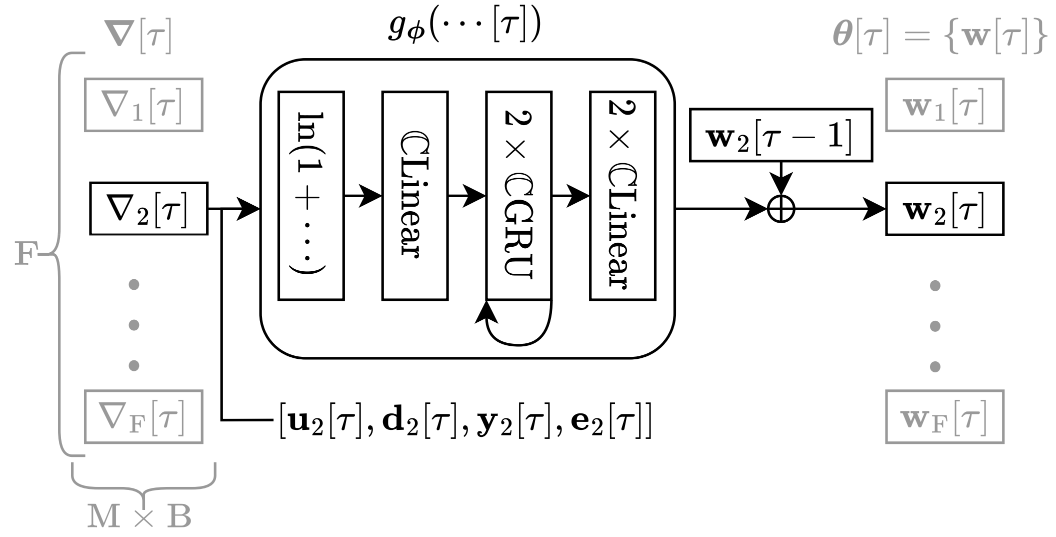


Figure 6.2: Coupled Meta-AF diagram showing inputs, pre-processing, and neural network architecture. This diagram applies to all tasks discussed in this section.

6.1.3 Experimental Setup and Results

We train g_ϕ via Algorithm 3.1 using a single GPU to adapt an OLS filter with a rectangular window size $N = 2048$ and hop size $R = 1024$ on 16 kHz audio. For training data, we created a dataset by convolving the far-end speech from the single-talk portion of the ICASSP 2021 AEC Challenge data [157] with room impulse responses (RIRs) from [158] truncated to 1024 taps.

For evaluation metrics, we use the segmental SNR (SNR$_d$) between the desired and estimated signals in dB. We compute this per-frame as,

$$\mathrm{SNR}_d(\underline{\mathbf{d}}[\tau], \underline{\mathbf{y}}[\tau]) = 10 \cdot \log_{10}\left(\frac{\|\underline{\mathbf{d}}[\tau]\|^2}{\|\underline{\mathbf{d}}[\tau] - \underline{\mathbf{y}}[\tau]\|^2}\right), \tag{6.1}$$

where higher is better. In AEC literature, this is often referred to as segmental echo-return loss enhancement (ERLE). We use these two terms interchangeably in this section. The `AF-Gym` contains this task in the noiseless setting with configurable system order both for the filter and for the true system. It also has options to include noise at the receiver side to evaluate robustness. Finally, it contains options for longer or shorter signals and system changes to study convergence and steady-state properties.

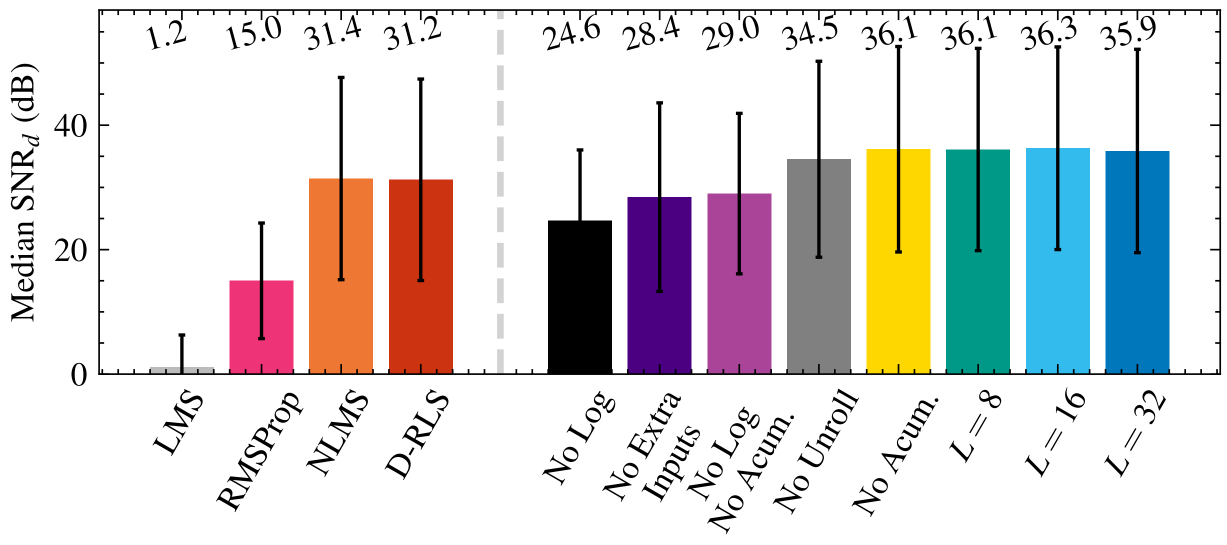


Figure 6.3: Optimizer design decision ablation. Using an accumulated log-loss leads to our best model, particularly for more complex tasks we address later on. RLS-like inputs are also useful. The exact value of L is not critical, but larger is better.

6.1.4 Optimizer Design Results & Discussion

In Fig. 6.3, we change one aspect of our final design at a time and show how each change negatively affects performance. Our final design, $L = 16$, is colored light blue, alternative configurations are colored differently, and our previous work [92] is approximately equivalent to a no log, no accumulation, no extra inputs setting. After this ablation, we fix these values for all remaining experiments and do not perform any further tuning except changing the dataset used for training and using different checkpoints caused by early stopping on validation performance. In contrast, we re-tune all conventional optimizer baselines for all subsequent tasks on held-out validation sets.

Loss Function: First, we compare our selected frame accumulated loss model (light blue) to the frame accumulated loss without log scaling (black) as well as the frame independent loss (yellow) and without log scaling (light purple). As shown, the log-loss has the single largest effect on SNR$_d$ and yields an astounding 11.7/7.3 dB improvement compared to the no-log losses. When we compare the independent vs. accumulated loss, the accumulation provides a .2 dB improvement. However, when we listen to the estimated response, especially in more complex tasks, we find that the accumulated loss introduces fewer artifacts and perceptually sounds better. Thus, we fix the optimizer loss to be (3.15).

Input Features: Having selected the optimizer loss, we compare the model inputs. We compare setting the optimizer input to be only the gradient $\boldsymbol{\xi}_{\mathrm{k}}[\tau] = [\boldsymbol{\nabla}\mathrm{k}[\tau]]$ for an LMS-like learned optimizer (dark purple) and setting the optimizer to be the full signal set,

$$\boldsymbol{\xi}_{\mathrm{k}}[\tau] = [\boldsymbol{\nabla}\mathrm{k}[\tau], \mathbf{u}_{\mathrm{k}}[\tau], \mathbf{d}_{\mathrm{k}}[\tau], \mathbf{y}_{\mathrm{k}}[\tau], \mathbf{e}_{\mathrm{k}}[\tau]], \tag{6.2}$$

for our selected RLS-like learned optimizer (light blue). As shown, the inputs have the second largest effect on SNR$_d$, and using the full signal set yields a notable 7.9 dB improvement. Thus, we set the input to be the full signal set.

AF Unroll: With the optimizer loss and inputs fixed, we evaluate four different values of AF unroll length, $L = 2, 8, 16, 32$, where L is the number of time-steps over which the optimizer loss is computed in (3.15). Intuitively, a larger unroll introduces less truncation bias but may be more unstable during training due to exploding or vanishing gradients. The case where $L = 2$ corresponds to no unroll, since for $L = 1$ the meta loss is not a function of the optimizer parameters and yields a zero gradient. As shown, for no unroll $L = 2$ (grey), we get a reduction of the SNR$_d$ by 1.8 dB compared to our best model. When selecting the unroll between $8, 16, 32$, however, there is a small (< 1 dB) overall effect. That said, we find that longer unroll values work better in challenging scenarios but take longer to train. As a result, we select an unroll length of 16, as it represents a good trade-off between performance and training time. Note, that the unroll length only affects training and is not used at test time.

System Order Modeling Error Results & Discussion: Given our fixed set of optimizer and meta-optimizer parameters, we evaluate the robustness of our Meta-ID AF to modeling errors by studying what happens when we use an optimizee filter that is too short or too long compared to the true system. We do so by testing a learned optimizer with 1) optimizee filter lengths between 256 and 4096 taps and 2) held-out signals with true filter lengths between 256 and 4096, as well as full-length systems (up to $32,000$ taps).

Results are shown in Fig. 6.4. We measure performance by computing the segmental SNR$_d$ score via (6.1) at convergence. As expected, when the adaptive filter order is equal to or greater than the true system order, we achieve SNRs of ≈ 40 dB. It is interesting to note that our learned AF was only ever trained on optimizee filters with an order of 1024 and 1024 tap true systems. This experiment suggests our learned optimizers can generalize to new optimizee filter orders. Given that these filters can recover an unknown system in a noiseless setting to high degrees of accuracy, we switch to a more challenging dataset with noise mixed into $\mathbf{d}$.

Higher-Order Dependencies – Noisy Not Real-Time: We show the effect of different higher-order dependencies on performance (left) and complexity (right) in Fig. 6.5. In this setting, we use the Microsoft AEC challenge dataset with noise (double-talk). We compare diagonal,

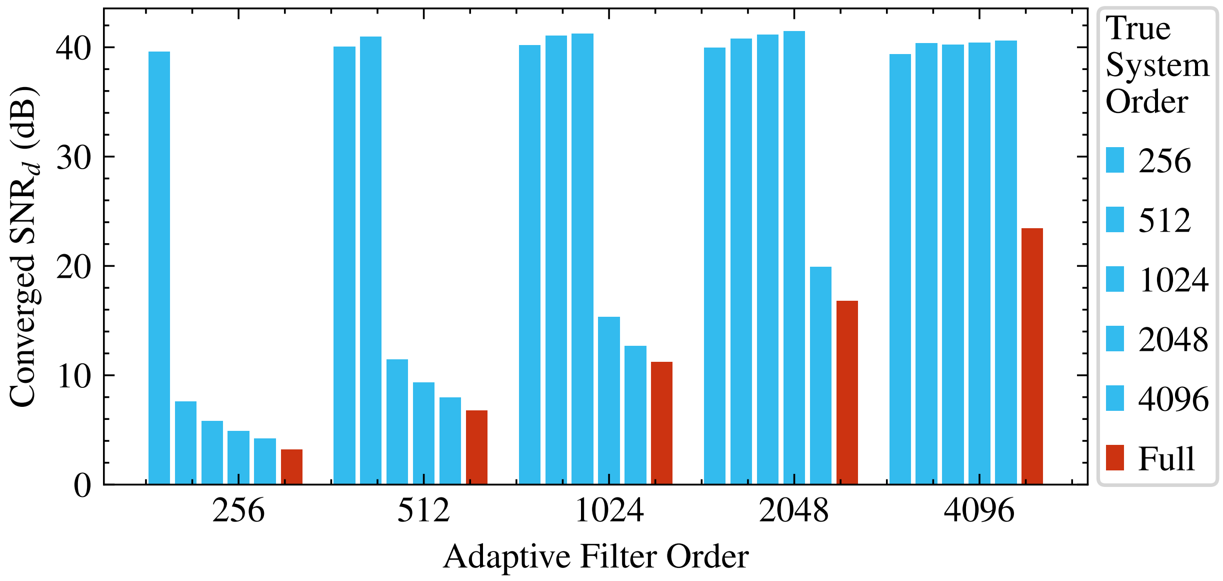


Figure 6.4: Evaluating the effect of different true system orders and adaptive filter orders. Our learned AFs can operate well across a variety of linear system orders, even when training is restricted to systems of a fixed length (1024 taps).

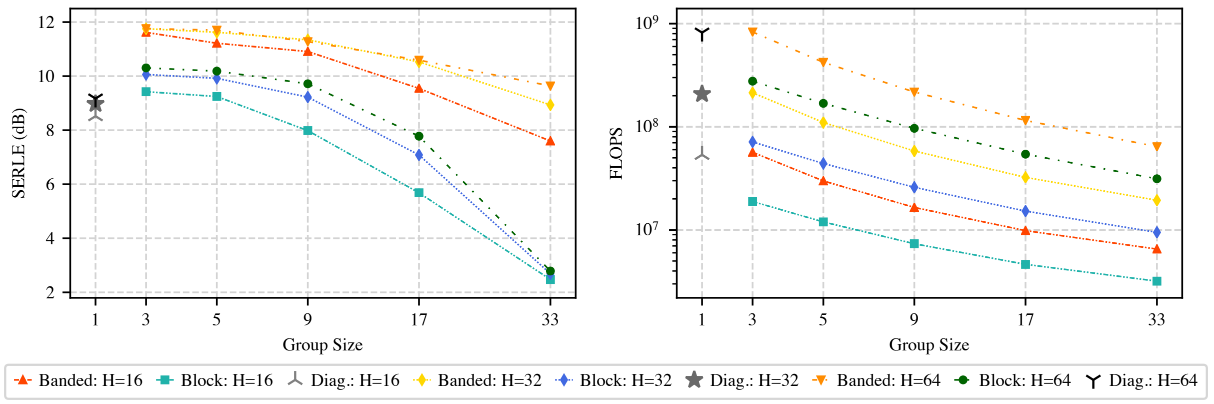


Figure 6.5: Higher-order frequency dependency comparison. Block and banded improve SERLE while reducing FLOPS.

block, and banded dependencies across state sizes of $H = \{16, 32, 64\}$. Block and banded optimizers outperform their diagonal counterparts and reduce complexity.

For diagonal optimizers, scaling H has little impact on performance and increases complexity. However, for higher-order optimizers, scaling H improves performance. For the higher-order optimizers, larger groups force more updates to be processed by a single GRU. For small groups, this improves performance and reduces complexity. Intuitively, neighboring frequencies are related, and grouping them allows the optimizer to exploit such relationships.

All frequencies within a group share the same hidden state, which reduces the number of states, which reduces complexity. Overall, banded has the best SERLE but block is more efficient. For this, we used an offline setup with a 4096 point window and a 2048 point hop. Next, we move to a real-time capable setup, that could be used in a real telephony system.

Scalable Multi-Step Adaptive Filters – Noisy Real-Time: Next, we explore the low-latency regime where we evaluate multi-block filters with a 512 point window a 256 point hop, and $B = 8$ blocks. This section uses the same Microsoft AEC challenge dataset with noise (double-talk) as the last section. We present the results of our experiments in Fig. 6.6, where the best-performing optimizer from Fig. 6.3 is shown as Meta-AF-M in the third column, and the banded group size 5 optimizer from Fig. 6.5 is shown in columns $4 - 6$. Interestingly, scaling trends do not hold in the lower latency regime. However, the scaling proposal from 4.4 results in the three right-most columns. These proposed models incorporate all Meta-AF insights including higher-order modeling, closed-loop features, full supervision, and multi-step optimization. The ability enabled by these gains is scaling, where a user can reliably and effectively trade compute for more performance.

To validate these techniques, we perform an initial within method ablation, then study scaling on a synthetic AEC task and vary 1) optimizer model sizes with small (S), medium (M), and large (L) models 2) an unsupervised (U) or supervised (S) loss and 3) the number of predict (P) and update (U) steps per frame. Each experiment is labeled with an identifier (e.g. S·S·PU), indicating the size, supervision, and number of predict/update steps.

We perform an initial within-method ablation on the task of AEC to understand our modifications. First, we compare a M·U·P variant trained with the full feature vs. the pruned feature set. The full feature set achieves an ERLE of 6.39dB (not shown), while the pruned set scores 7.85dB, over a dB gain. We expand on the pruned model and add supervision, resulting in an ERLE improvement to 9.84dB, a gain of nearly 2dB. We then use multiple steps per frame. Extending the supervised model with an update step increases ERLE to 11.22dB, and doubling the iterations reaches 11.77dB. Combined, our changes yield a 4dB ERLE gain. An interesting note which becomes important for the AEC section, is that we used a modified OLA scheme to mitigate clicking artifacts. This change reduces the ERLE by ≈ 1 dB ERLE, but removes clicking and sounds better.

The insights and advances proposed in this section apply to various real-world filtering problems. In the next sections, we systematically explore a suite of more challenging real-world tasks. These range from perceptually motivated tasks, to smart-system tasks, to multi-channel, and full end-to-end pipelines. We take the results here and apply them as-is. That is, we do not make any further custom designs for Meta-AF powered modules.

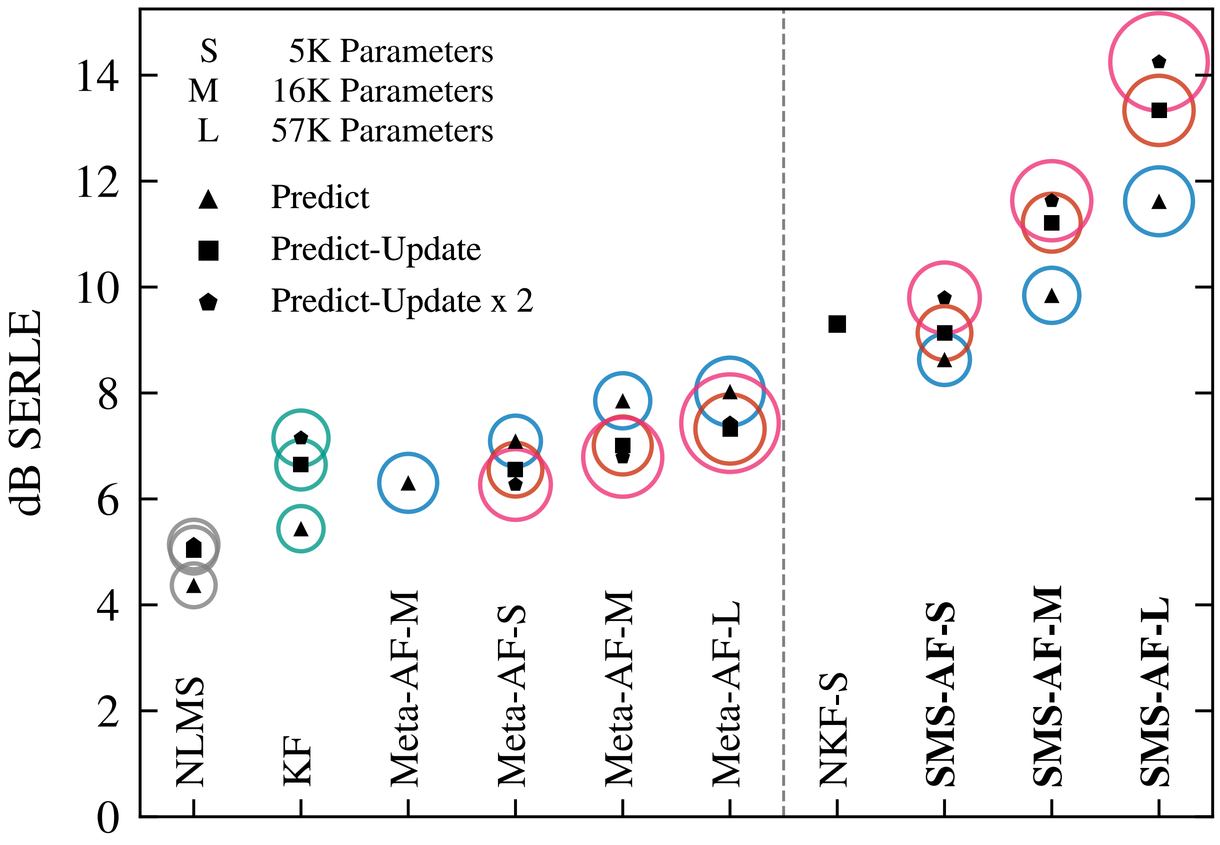


Figure 6.6: Acoustic echo cancellation performance vs. model size, optimization steps per time-frame (opt. steps), and supervision levels. Bubble size shows real-time-factor (RTF) where smaller is faster, inner-shape shows opt. steps, and the vertical dotted line separates unsupervised (left) and supervised (right) approaches. Our proposed models, shown in bold on the far right, demonstrate robust scaling performance in terms of parameters, and RTF.

6.2 ISOLATED ACOUSTIC ECHO CANCELLATION

In acoustic echo cancellation, the goal is to remove the far-end echo from a near-end signal for voice communication by mimicking a time-varying transfer function, as shown in Fig. 6.7. The far-end refers to the signal transmitted to a local listener and the near-end is captured by a local mic. This is effective system identification with noise and real-time constraints. We model the unknown transfer function between the speaker and mic with a linear multi-delay frequency-domain filter $h_{\boldsymbol{\theta}}$, measure the noisy response $\mathbf{d}$ which includes the input signal $\mathbf{u}$, noise $\mathbf{n}$, and near-end speech $\mathbf{s}$, and adapt the filter weights $\boldsymbol{\theta}$ using $g_{\boldsymbol{\phi}}$. The time-domain signal model is $\underline{\mathbf{d}}[\mathrm{t}] = \underline{\mathbf{u}}[\mathrm{t}] * \underline{\mathbf{w}} + \underline{\mathbf{n}}[\mathrm{t}] + \underline{\mathbf{s}}[\mathrm{t}]$. The AF loss is most often ISE between the near-end and the predicted response. The FDAF near-end speech estimate is $\hat{\mathbf{s}}_{\mathrm{k}}[\tau] = \mathbf{d}_{\mathrm{k}}[\tau] - \mathbf{w}_{\mathrm{k}}[\tau]^{\mathsf{H}}\mathbf{u}_{\mathrm{k}}[\tau]$. It is common for the AEC task to rely on post-processors (as

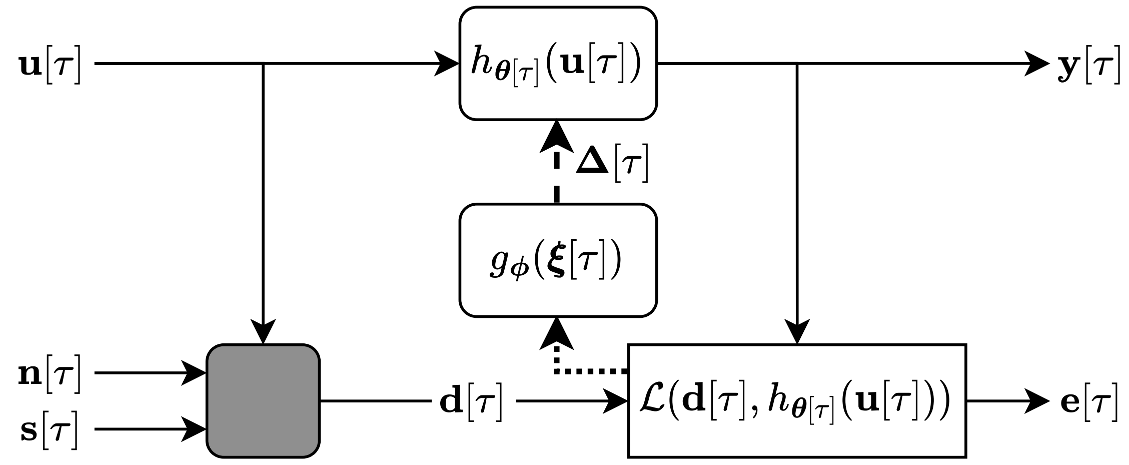

$\mathbf{u}[\tau]$
$h_{\boldsymbol{\theta}[\tau]}(\mathbf{u}[\tau])$
$\mathbf{y}[\tau]$
$\boldsymbol{\Delta}[\tau]$
$g_{\boldsymbol{\phi}}(\boldsymbol{\xi}[\tau])$
$\mathbf{n}[\tau]$
$\mathbf{s}[\tau]$
$\mathbf{d}[\tau]$
$\mathcal{L}(\mathbf{d}[\tau], h_{\boldsymbol{\theta}[\tau]}(\mathbf{u}[\tau]))$
$\mathbf{e}[\tau]$

Figure 6.7: AEC block diagram. System inputs are fed to the adaptive filter and true system (shaded box). The adaptive filter is updated to mimic the true system. The desired response can be noisy due to near-end noise and speech ($\mathbf{n}[\tau], \mathbf{s}[\tau]$).

in our end-to-end telephony) so we also make the ideal clean speech available for training.

In multi-channel acoustic echo cancellation, the goal is again to remove the far-end echo from a near-end signal. However, there may be multiple sources of echo and there may be multiple near-ends to process. Practically, this still amounts to using an FDAF with multiple near-end speech estimates $\hat{\mathbf{s}}_{\mathrm{km}}[\tau] = \mathbf{d}_{\mathrm{km}}[\tau] - \mathbf{w}_{\mathrm{km}}[\tau]^{\mathsf{H}}\mathbf{u}_{\mathrm{k}}[\tau]$ where $\mathbf{u}_{\mathrm{k}}[\tau] \in \mathbb{C}^{BM'}$ and M' is the number of farend signals. This process is typically considered more challenging due to the presence of AF solutions that do not correspond to the true system but still minimize the loss. Such spurious solutions typically cause over-adaptation, over-suppression, or other issues.

We measure echo cancellation performance using segmental echo-return loss enhancement (ERLE) [159], short-time objective intelligibility (STOI) [160], and scale-invariant signal-to-distortion ratio (SI-SDR) [161]. Segmental ERLE is

$$\mathrm{ERLE}(\underline{\mathbf{d}}_{\mathbf{u}}[\tau], \underline{\mathbf{y}}[\tau]) = 10 \cdot \log_{10}\left(\frac{\|\underline{\mathbf{d}}_{\mathbf{u}}[\tau]\|^2}{\|\underline{\mathbf{d}}_{\mathbf{u}}[\tau] - \underline{\mathbf{y}}[\tau]\|^2}\right), \tag{6.3}$$

where $\underline{\mathbf{d}}_{\mathbf{u}}$ is the noiseless system response and higher values are better. When averaging, we discard silent frames using an energy-threshold VAD. In scenes with near-end speech, we use STOI $\in [0, 1]$ to measure the preservation of near-end speech quality. Higher STOI values are better. SI-SDR uses $\mathbf{a} = (\hat{\underline{\mathbf{s}}}^{\top}\underline{\mathbf{s}})/\|\underline{\mathbf{s}}\|$ and is $\text{SI-SDR}(\underline{\mathbf{s}}, \hat{\underline{\mathbf{s}}}) = 10 \cdot \log_{10}(\|\mathbf{a}\underline{\mathbf{s}}\|^2/\|\mathbf{a}\underline{\mathbf{s}} - \hat{\underline{\mathbf{s}}}\|^2)$.

6.2.1 Prior Works

When we survey further improvements to AFs with a focus on acoustic echo cancellation (AEC) [9, 10], numerous improvements have been proposed. For example, near-end signal models have been proposed for handling simultaneous far-end and near-end activity (double-talk) [8, 12, 18] as well as state-space formulations [11, 14, 15, 16]. A few works [148, 149] also propose to use higher-order frequency dependencies for AEC AFs, likely motivated by the success of multivariate statistics for source separation [39, 40], but showed varied performance improvement with increased complexity. We compare our approach to LMS, RMSProp, NLMS, BD-RLS, a diagonal Kalman filter (D-KF) model [11], and Speex [12] for a variety of acoustic echo cancellation scenarios. Our scenarios, in increasing difficulty, include single-talk, double-talk, double-talk with a path change, and noisy double-talk with a path change and non-linearity. Single-talk refers to the case where only the far-end input signal $\mathbf{u}$ is active. Double-talk refers to the case where both the far-end signal $\mathbf{u}$ and near-end talker signal $\mathbf{s}$ are active at the same time.

6.2.2 Evaluation

We train g_ϕ via Algorithm 3.1 using one GPU, which took ≈ 72 hours. We use a four-block multi-delay OLS filter (MDF) with window sizes of $N = 1024$ samples and a hop of $R = 512$ samples on 16 kHz audio. In double-talk scenarios, we use the noisy near-end, $\mathbf{d}$ as the target, and do not use oracle cancellation results (such as $\underline{\mathbf{d}}_{\mathbf{u}}$).

With respect to datasets for single-talk, double-talk, and double-talk with path-change experiments, we re-mix the synthetic fold of the ICASSP 2021 AEC Challenge dataset (ICASSP-2021-AEC) [157] with impulse responses from [158]. We partition [158] into the non-overlapping train, test, and validation folds and set the signal-to-echo-ratio randomly between $[-10, 10]$ with uniform distribution. To simulate a scene change, we splice two files such that the change occurs randomly between seconds four and six. For the noisy double-talk with nonlinearity experiments, we use the synthetic fold of [157]. We apply a random circular shift and random scale to all files, each ten seconds long. For each task, there are 9000 training, 500 validation, and 500 test files. Finally, we also use an unmodified version of the ICASSP-2021-AEC training, validation, and test set (does not include scene changes) to compare to other previously published works directly. Overall, we find that our approach significantly outperforms all previous methods in all scenarios, but has a larger advantage in harder scenes—more details discussed below.

Single-Talk: Our approach (light blue, x) exhibits strong single-talk performance and

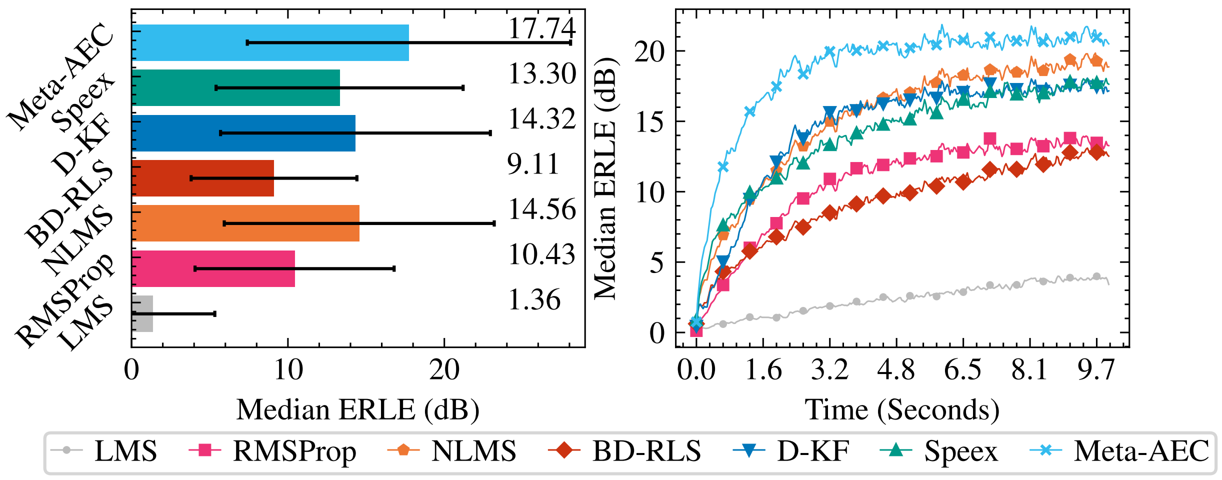


Figure 6.8: AEC single-talk performance. Meta-AEC converges rapidly and has better steady-state performance. We use this same legend for all AEC plots.

surpasses all baselines by $>\approx 3$ dB in both median and converged ERLE, as shown in Fig. 6.8. Additionally, Meta-AEC converges fastest, reaching steady-state ≈ 4 seconds before other baselines.

Double-Talk: Our method (light blue, x) converges fastest in double-talk, and matches the D-KF in converged performance, as shown in Fig. 6.9. Meta-AEC converges ≈ 5 seconds faster while scoring better in STOI. This result is striking as it is typically necessary to either explicitly freeze adaptation via double-talk detectors or implicitly freeze adaption via carefully derived updates as found in both the D-KF model (dark blue, down triangle) and Speex (green, up triangle). We hypothesize our method automatically learns how to adapt to double-talk in a completely autonomous fashion.

Double-Talk With Path Change: Our method (light blue, x) is able to more robustly handle double-talk with path changes compared to other methods as shown in Fig. 6.10. Similar to straight double-talk, our approach effectively learns how to deal with adverse conditions (i.e. a path change) without explicit supervision, converging and reconverging in ≈ 2.5 seconds, with .044 better median STOI. All other algorithms similarly struggle, even Speex (green, up triangle), which has explicit self-resetting and dual-filter logic.

Noisy Double-Talk With Nonlinearities and Path Change: When we evaluate scenes with noise, and nonlinearities to simulate loudspeaker distortion and path changes, we find that our Meta-AEC approach (light blue, x) continues to perform well, as shown in Fig. 6.11. That is, our peak performance is ≈ 2 dB above the nearest baseline. In STOI, Meta-AEC scores .027 above Speex (green, up triangle). We hypothesize that our approach effectively learns to compensate for the signal model inaccuracy, even if we only use a linear filter.

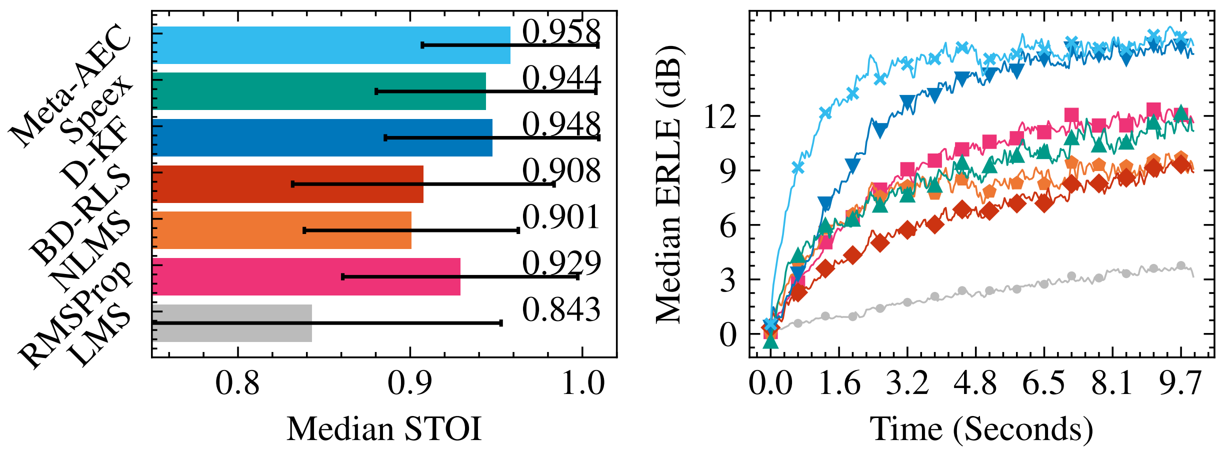


Figure 6.9: AEC double-talk performance. Meta-AEC converges fastest and has a similar peak performance to the D-KF while preserving near-end speech quality.

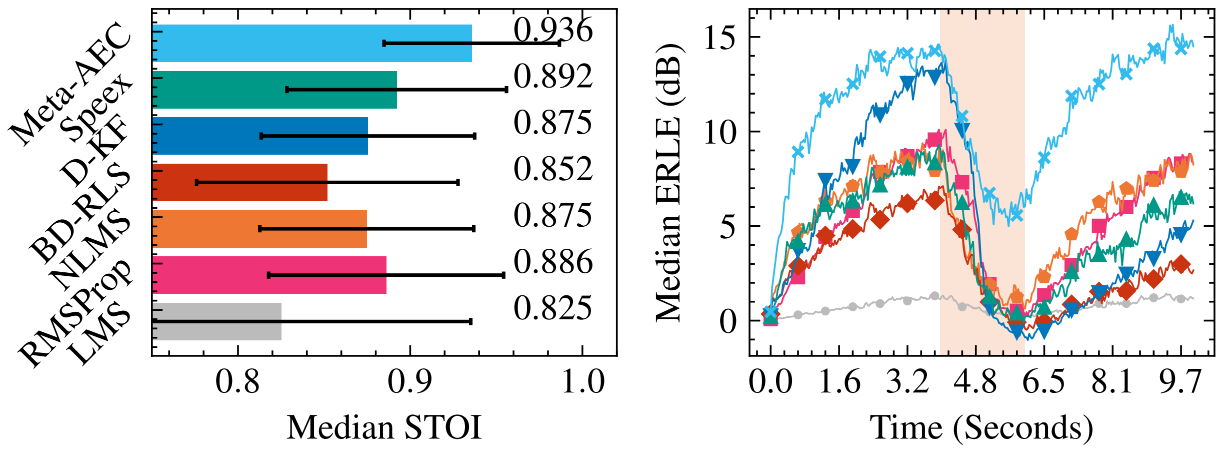


Figure 6.10: AEC double-talk with a path change (shaded region) performance. Our approach re-converges rapidly with high speech quality.

ICASSP 2021 AEC Challenge Results: In addition to testing with our own variant of the ICASSP-AEC-2021 dataset that includes scene changes, we test our work with an unmodified version of the test set in Table 6.1. Furthermore, we evaluate performance when we train (or tune) on our custom training dataset versus when we train on the original training dataset (denoted with *). See also a similar Table in past work [132]. To the best of our knowledge, this dataset is the most recent and widely used dataset for benchmarking AEC algorithms. Here, results from WebRTC-AEC3 and wRLS, $\beta = 0.2$ come from past work [132]. All other methods have the same MDF filtering architecture as described above.

Our approach outperforms all methods we compare against for both training datasets, including Speex and wRLS, which were the linear filters used in the first- and second-place

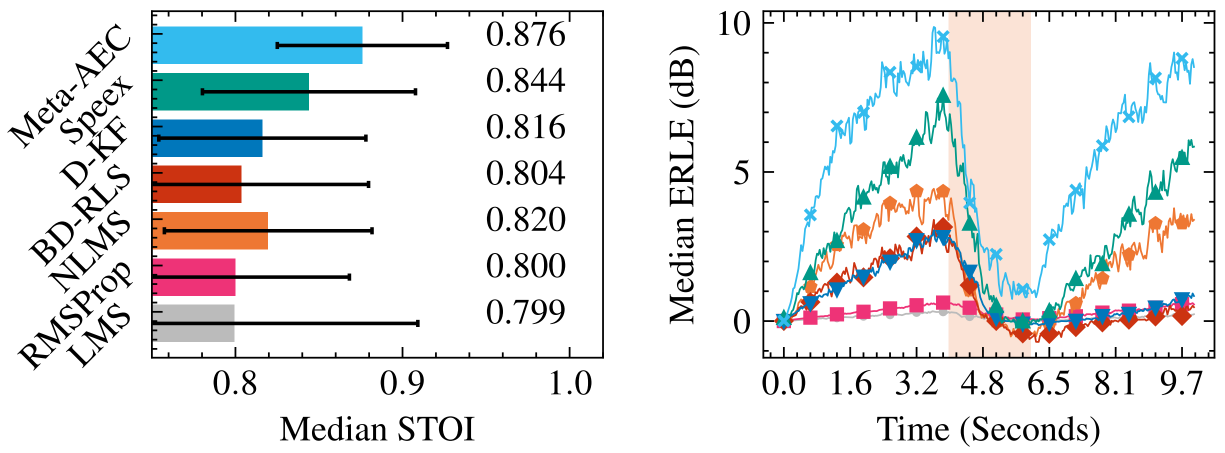


Figure 6.11: AEC noisy double-talk with nonlinearities and a path change (shaded region) performance. Meta-AEC learns to compensate for the nonlinearity.

Method	**STOI**	**STOI***	**ERLE** (dB)	**ERLE*** (dB)
LMS	0.794	0.794	0.937	0.560
RMSProp	0.802	0.856	1.02	4.63
NLMS	0.854	0.861	4.81	5.96
BD-RLS	0.829	0.835	3.66	4.07
D-KF [11]	0.817	0.875	1.98	6.55
wRLS, $\beta = 0.2$ [132]	N.A	0.85	N.A	N.A
WebRTC-AEC3 [133]	0.82	N.A	N.A	N.A
Speex [12]	0.869	N.A	3.92	N.A
Meta-AEC	**0.881**	**0.899**	**7.73**	**9.13**

Table 6.1: ICASSP-2021-AEC test set linear filter results. Our proposed method outperforms several past comparable linear-filtering approaches. A $\star$ denotes results when models were trained/tuned on the ICASSP-2021-AEC data.

winners of the ICASSP 2021 AEC Challenge [135]. Interestingly, there is a significant effect on training or tuning with data that includes scene changes (ours) vs. the original data (e.g. RMSProp and D-KF [87]). That is, because the ICASSP-2021-AEC train and test set does not include scene changes, most algorithms give better performance when trained/tuned on the matching, unmodified train set, even though such results are less realistic.

Computational Complexity: Our learned AF has a single CPU core real-time factor (RTF), computation / time, of ≈ 0.36, and 32 ms latency (OLS hop size). Our optimizer network alone has $\approx 14K$ complex-valued parameters and a single CPU core RTF of ≈ 0.31. While this performance is already real-time capable, we suspect it could easily be improved with better-optimized code.

6.2.3 Scaling-Up Acoustic Echo Cancellation

In this section, we evaluate the proposed SMS-AF approach. This method enables scaling performance by increasing model capacity and/or inference cost as shown. The goal of our experiments is to benchmark SMS-AF and demonstrate how it scales across. We take the initial ablation from the system identification section and then study scaling on an AEC task and vary 1) optimizer model sizes with small (S), medium (M), and large (L) models and 2) unsupervised (U) or supervised (S) loss and 3) the number of predict (P) and update (U) steps per frame. Each experiment is labeled with an identifier (e.g. S·S·PU), indicating the size, supervision, and number of predict/update steps. We label baseline methods when applicable.

Prior Works: Our baselines are NLMS, KF [11], Meta-AF [93], HO-Meta-AF [94], and Neural-Kalman Filter [134]. We also test several HO-Meta-AF model sizes as well as multi-step NLMS, KF, and HO-Meta-AF.

When using overlap-save, we noticed artifacts due to rapid filter adaptation. So, we applied a straightforward solution: overlap-add with a synthesis window, but no analysis window. This aims to maintain analysis signal model integrity, while still removing clicking. We use this synthesis-only overlap-add scheme for all models in this experiment.

Evaluation: We use a linear MDF filter with 8 blocks, each of size 512, a hop of 256, and construct the output using overlap-add with a Hann window (16ms latency). For details see Benesty et al. [9]. We use higher-order Meta-AF optimizers with banded coupling and a group size of 5 [94]. This amounts to a Conv1D layer, two GRU layers, and a transposed Conv1D layer. To train, we use Adam with a batch size of 16, a learning rate of 10^{-4}, and randomize the truncation length L with a maximum of 128. We apply gradient clipping and reduce the learning rate by half if the validation performance does not improve for 10 epochs, and stop training after 30 epochs with no improvement. We use log-MSE loss on the echo,

$$\mathcal{L}_S(\mathbf{d_u}, \mathbf{e}) \quad = \quad \ln E[||\underline{\bar{\mathbf{d}}}_{\mathbf{u}}[\tau] - \underline{\bar{\mathbf{e}}}[\tau]||^2], \tag{6.4}$$

where $\mathbf{d_u}[\tau] = \mathbf{u}[\tau] * \mathbf{w}[\tau]$ is the true echo. All models are complex and trained on one GPU. Note, our S, M, and L model sizes correspond to g_ϕ hidden state sizes of 16, 32, and 64 total parameters counts of about 5K, 16K, and 57K respectively.

Next, we explore scaling in AEC as shown in Fig. 6.6 and Table 6.2. We attempt to scale up our baselines and then do so with our proposed model. When scaling model size, we notice that scaling the unsupervised model from S to L (S·U·P to L·U·P) results in a peak gain of $\approx$ 1dB, to 8.03dB ERLE. In contrast, scaling the supervised model from S to L (S·S·P to L·S·P) yields larger gains, peaking at 11.62dB. When scaling optimization steps, we find the unsupervised models from S·U·P to S·U·PUx2 results in marginal or even a negative performance changes (indicating overfitting). In contrast, scaling from S·S·P to S·S·PUx2 provides +1dB, and L·S·P to L·S·PUx2 provides +2.65dB, showing supervision is crucial to unlock the benefit of multiple opt. steps per frame. Our best-performing L·S·PUx2 scores over 14dB ERLE, doubling the S·U·P performance of 7.09dB. In sum, SMS-AF outperforms prior models, yielding $\approx 0.085H + 8$dB, or an additional db for each 12 hidden units in the optimizer.

When benchmarking against competing methods, we note that SOTA supervised NKF method is the most comparable. Our S·S·PU model matches NKF performance while using only one-fifth of the NKF MFLOP count. Our M·S·PU model further enhances all metrics and uses fewer MFLOPs. For our best-performing L·S·PUx2, we score 14.25dB ERLE, a 4.96dB improvement over NKF. With respect to perceptual metrics, our top-performing L·S·PUx2 model achieves over 11dB in real ERLE and a real AEC-MOS of 3.94, while remaining real-time on a single CPU core. Surprisingly, RTF scales non-linearly with MFLOPs and model size, underscoring untapped scaling potential. In all, SMS-AF outperforms past work in signal-level metrics, perceptual metrics, and on real playback.

6.3 ACOUSTIC ECHO CANCELLATION AND SPEECH ENHANCEMENT

Acoustic echo cancellation is typically just one element within a telephony pipeline. Given the input mixture represented as $\underline{\mathbf{d}}[\mathrm{t}] = \underline{\mathbf{u}}[\mathrm{t}] * \underline{\mathbf{w}} + \underline{\mathbf{n}}[\mathrm{t}] + \underline{\mathbf{s}}[\mathrm{t}]$, the goal of AEC is to reduce it to $\underline{\mathbf{n}}[\mathrm{t}] + \underline{\mathbf{s}}[\mathrm{t}]$, leaving the desired speech signal still affected by noise. Generally, AEC is not flawless, and the speech signal may still carry some residual echo. To address this, engineers usually build a post-processing module for residual echo and noise suppression. In this section, we treat this as another learning problem by training a post-processing DNN to serve as a speech enhancer, ultimately recovering the clean, desired speech. This pipeline is illustrated in Fig. 6.12, where the output of AEC is fed into another DNN.

6.3.1 Prior Works

In AEC, neural networks can be trained for residual echo suppression, noise suppression, reference estimation, and more [45, 46, 47, 48, 49, 50, 51, 52, 53]. We compare block-frequency NLMS, RLS, KF [11], and diagonalized Meta-AF [93] to higher-order Meta-

[1]The public NKF implementation is not an online algorithm.

Model	ERLE↑	Real↑ ERLE	AEC-MOS↑	Real AEC-MOS↑	MFLOPs↓	RTF↓
Mixture	-	-	2.33	2.69	-	-
NLMS·P	4.37	1.74	3.16	3.06	0.08	0.31
KF·P	5.44	2.71	3.32	3.18	0.12	0.32
KF·PU	6.65	3.83	3.65	3.38	0.12	0.35
KF·PUx2	7.15	5.58	3.89	3.66	0.18	0.39
M·U·P [93]	6.30	2.99	3.62	3.31	9.02	0.41
S·U·P	7.09	3.30	3.57	3.26	2.80	0.36
M·U·P	7.85	3.75	3.62	3.30	7.07	0.39
L·U·P	8.03	3.91	3.65	3.33	20.42	0.48
NKF·S·PU	9.29	6.38	3.69	3.59	10.16	[1]
S·S·P	8.63	3.81	3.63	3.31	2.80	0.36
M·S·P	9.84	4.62	3.73	3.40	7.07	0.39
L·S·P	11.62	5.52	3.83	3.50	20.42	0.48
S·S·PU	9.13	6.05	3.85	3.55	2.81	0.38
M·S·PU	11.22	7.87	3.99	3.72	7.08	0.41
L·S·PU	13.34	9.78	4.05	3.85	20.43	0.49
S·S·PUx2	9.80	6.96	3.85	3.66	5.56	0.50
M·S·PUx2	11.77	8.86	4.04	3.80	14.11	0.56
L·S·PUx2	**14.25**	**11.15**	**4.12**	**3.94**	40.81	0.69

Table 6.2: AEC Performance vs. Computational Cost on both the real and synthetic folds of the Microsoft AEC Challenge dataset.

AF with block and banded frequency dependency structures. We also train a pure neural network-based speech enhancer based on the model in Microsoft AEC challenge [157].

6.3.2 Evaluation

For AEC, we use a 4096 pt. window and a 2048 pt. hop. We train Meta-AF models using (3.16) and the training scheme from algorithm 3.1 with Adam ($\lambda = 10^{-4}$). We train the DNN-SE using a 512 point window, 256 point hop, Adam ($\lambda = 6 \cdot 10^{-4}$), with AEC-processed inputs and clean speech as the target using mean-squared error on the magnitude short-time Fourier-transform. We set $L = 20$ for Meta-AF training and $L = 150$ for DNN-SE training. We use gradient clipping, cut the learning rate in half if validation performance does not improve for 5 epochs, and stop training after 16 with no improvement.

For our enhancer $m_{\boldsymbol{\varphi}}(\mathbf{e}[\tau], \mathbf{u}[\tau])$, we follow [157] and use a 2-layer GRU with log-magnitude

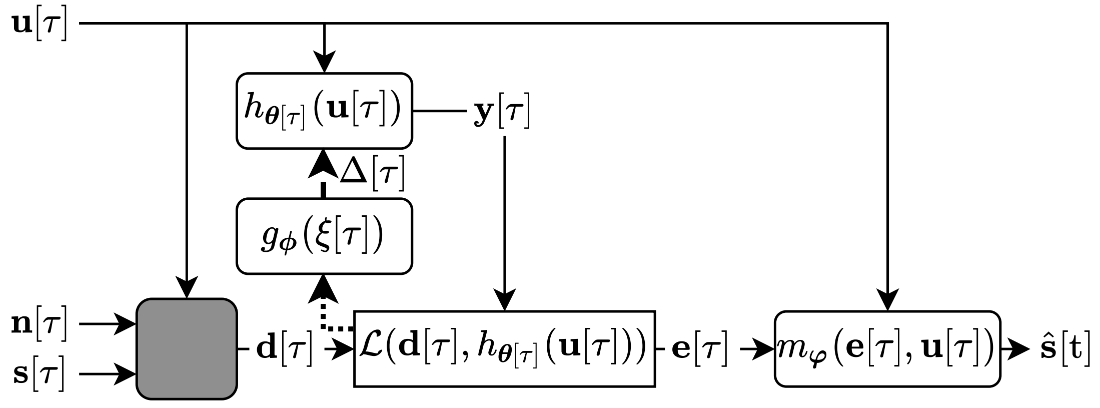

$\mathbf{u}[\tau]$
$h_{\boldsymbol{\theta}[\tau]}(\mathbf{u}[\tau])$
$\mathbf{y}[\tau]$
$\Delta[\tau]$
$g_{\boldsymbol{\phi}}(\xi[\tau])$
$\mathbf{n}[\tau]$
$\mathbf{s}[\tau]$
$\mathbf{d}[\tau]$
$\mathcal{L}(\mathbf{d}[\tau], h_{\boldsymbol{\theta}[\tau]}(\mathbf{u}[\tau]))$
$\mathbf{e}[\tau]$
$m_{\boldsymbol{\varphi}}(\mathbf{e}[\tau], \mathbf{u}[\tau])$
$\hat{\mathbf{s}}[\mathbf{t}]$

Figure 6.12: Meta-AF for AEC with a DNN speech enhancer. The shaded box represents an unknown system. The system acts as a complete telephony pipeline and attempts to remove echo, residual echo, and any non-speech noise.

short-time Fourier transforms of the far-end, $\mathbf{u}[\tau]$ and the AEC output $\mathbf{e}[\tau]$ as inputs with output,

$$\hat{\mathbf{s}}[\tau] = \mathbf{e}[\tau] \odot \mathbf{M}[\tau], \tag{6.5}$$

using magnitude mask, $\mathbf{M}[\tau] \in \mathbb{R}^K$ bounded with a Sigmoid function. $\odot$ is the hadamard product. $m_{\boldsymbol{\varphi}}(\cdot)$ is trained to remove noise and residual echo.

We show the effect of AEC on speech enhancement performance in Fig. 6.13. In solid colors, we show the performance of the AEC, and in striped colors we show the performance of AEC along with a DNN-SE. We show three Meta-AEC models all with $H = 32$: diagonal, banded with group 9, and banded with group 3. Banded-3 performs best and beats KF by 3.21 dB SI-SDR and .038 STOI. The less complex Banded-9 performs similarly and both beat diagonal Meta-AEC by > 1 dB SI-SDR and $> .01$ STOI. The trend holds when paired with a DNN-SE. Banded-9 surpasses KF by 2.2 dB SI-SDR and .018 STOI and diagonal by 1.3 dB SI-SDR and .01 STOI. This demonstrates that modeling higher-order dependencies translates to better downstream performance and highlights that AF advances can improve overall system performance. The raw mixture scores -1.15 dB SI-SDR and 0.78 STOI. Oracle AEC and DNN-SE score 32.27 dB SI-SDR and 0.97 STOI.

Perceptually, the Banded-3 model with a DNN-SE sounds very good. Residual echo is basically eliminated and noise levels are very low. The models are also quite practical. All AECs run in real-time on a single CPU core with RTFs of: 0.12 for KF, 0.15 for Diag., 0.18 for Banded-3, and 0.13 for Banded-9. Banded-9 is as fast as KF and outperforms Diag. Meta-AEC models have 14K complex parameters. Note that this DNN-SE approach

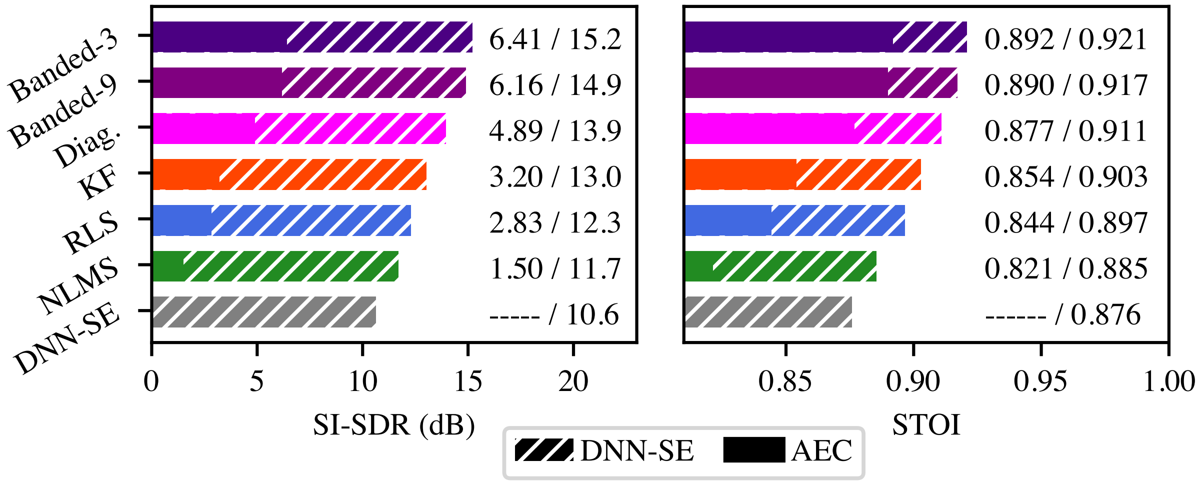


Figure 6.13: Effect of AEC on DNN-SE performance. Block Meta-AEC performs best before and after the DNN-SE.

is compatible with any AEC, including the SMS-AF approaches for even better performance. We evaluated the $M \cdot S \cdot PU$ SMS-AF model with this same DNN-SE setup and scored some 16.23dB SI-SDR all while running with a latency of 16ms, and a real-time-factor (processing time / elapsed-time) of 0.48.

6.4 ACOUSTIC ECHO CANCELLATION AND KEYWORD SPOTTING

Adaptive filters (AFs) are essential for many smart systems, including automatic speech recognition, sound event detection, and keyword spotting [162]. These AFs serve as key preprocessing steps and are designed to enhance the performance of their downstream components by removing noise, reverberation, and other distortions from their input signals. Traditionally, AFs have been optimized for signal-level objectives, such as minimizing mean-squared error, using techniques like recursive least squares [1, 3, 5]. However, recent advancements in deep learning have introduced methods for controlling pre-built AF update rules [78, 155, 163], or for learning entirely new update rules from scratch [92, 93]. Despite these innovations, existing techniques often do not take into account the downstream task and may not lead to the best performance in practice.

Our goal is a framework that allows for the incorporation of downstream classification task knowledge into AFs without task-dependent design or test-time feedback, CT-Meta-AF. CT-Meta-AF is unique in that it does not require oracle signal-level supervision such as impulse responses or clean speech. Instead, it leverages a combination of signal-level self-supervision and automatic classifier feedback. This eliminates the need for manual

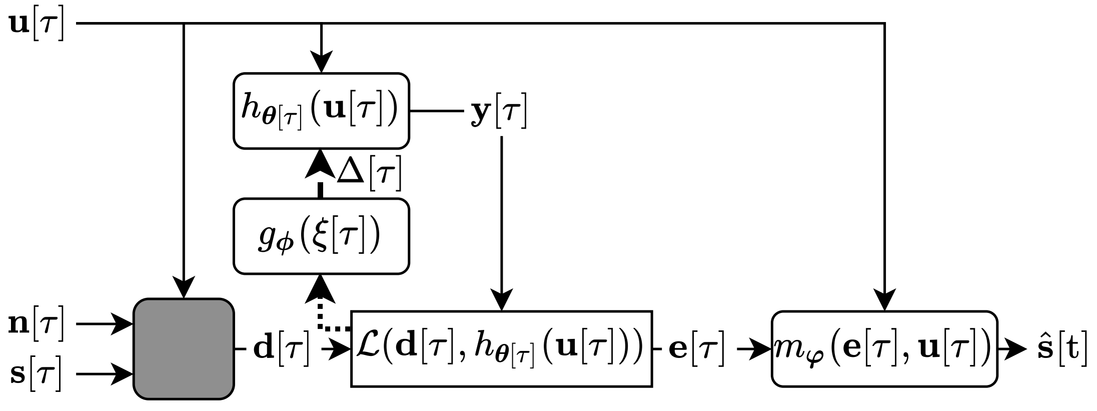

$\mathbf{u}[\tau]$
$h_{\boldsymbol{\theta}[\tau]}(\mathbf{u}[\tau])$
$\mathbf{y}[\tau]$
$\Delta[\tau]$
$g_{\boldsymbol{\phi}}(\xi[\tau])$
$\mathbf{n}[\tau]$
$\mathbf{s}[\tau]$
$\mathbf{d}[\tau]$
$\mathcal{L}(\mathbf{d}[\tau], h_{\boldsymbol{\theta}[\tau]}(\mathbf{u}[\tau]))$
$\mathbf{e}[\tau]$
$m_{\boldsymbol{\varphi}}(\mathbf{e}[\tau], \mathbf{u}[\tau])$
$\hat{\mathbf{s}}[\mathbf{t}]$

Figure 6.14: The echo canceller $h_{\boldsymbol{\theta}[\tau]}$, controlled by optimizer $g_{\boldsymbol{\phi}}$, cancels own-playback and passes its output to keyword classifier $m_{\boldsymbol{\varphi}}$. The optimizer is trained end-to-end with the classifier, improving performance without the need for additional tuning.

task/model-dependent tuning and is directly compatible with both off-the-shelf and end-to-end jointly trained models.

6.4.1 Prior Works

Several pioneering works demonstrated that incorporating of downstream task knowledge into AF control through manual unification schemes can enhance performance [73, 164, 165]. This improvement necessitates a complex, task/model-dependent derivation process as well as feedback from the downstream model to the AF at test time. This line of work inspired many DNN-based approaches. These DNN approaches can be broadly grouped into two categories. The first explicitly decouples preprocessing and downstream operations, treating them as sequential components within a larger DNN pipeline [76, 83, 162, 166]. This is highly modular but typically requires more complex training schemes and engineering overhead [80]. The second takes an end-to-end strategy, trading modularity for simpler design and training schemes [81, 85, 156, 167]. These approaches have varied trade-offs, making them suitable for different scenarios.

6.4.2 Evaluation

To evaluate our framework, we apply it to two fundamental tasks for smart devices: acoustic echo cancellation (AEC) and keyword spotting (KWS). AEC is a critical component of nearly all devices with own-playback, and KWS is essential for enabling hands-free operation

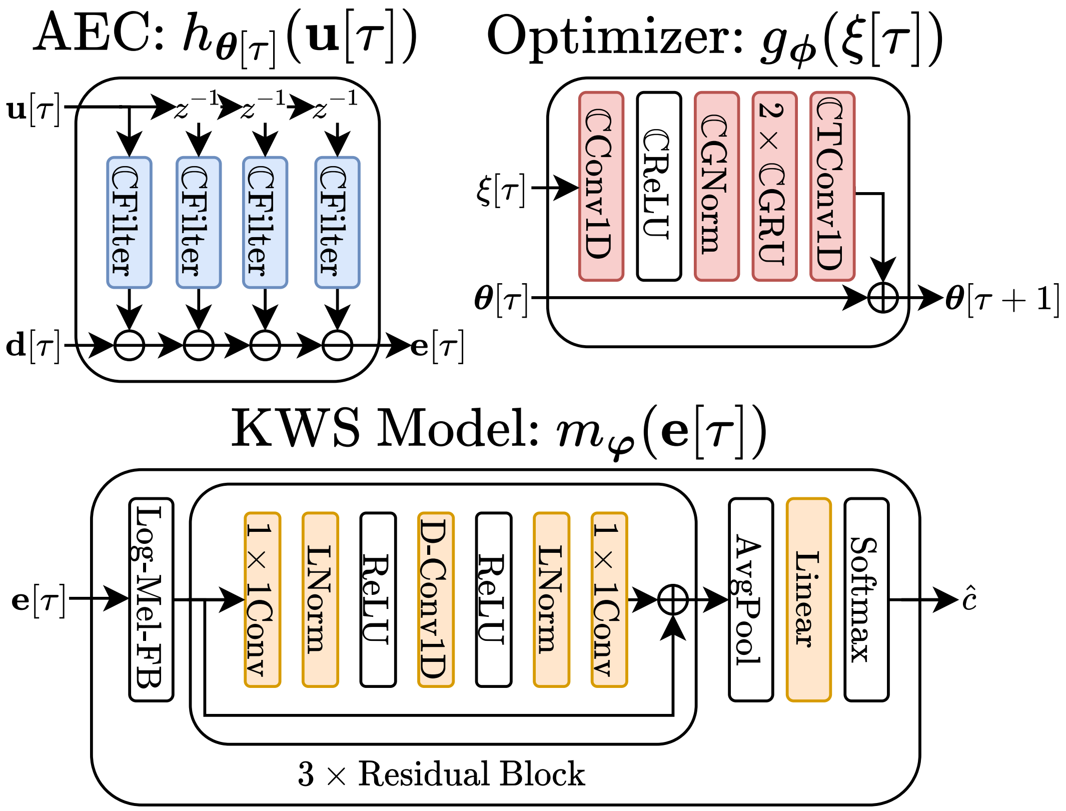


Figure 6.15: Detailed view of the AEC, Optimizer, and KWS. The AEC preprocesses the KWS inputs and the optimizer controls the AEC parameters. At test time, only the AEC parameters change

of smart devices. We construct a challenging joint AEC and KWS dataset by combining two existing datasets [157, 168], and benchmark CT-Meta-AFs against Meta-AF [94] and a Kalman Filter [15], two state-of-the-art approaches for signal-level objectives. Results show that CT-Meta-AF achieves consistent performance gains across synthetic and real playback scenarios, different playback lengths, various task/model configurations, and both pre-trained and joint-trained keyword spotting models.

The goal of our setup is to demonstrate that CT-Meta-AF strikes the right balance between flexibility and structure, enabling high performance while still being a direct replacement for existing approaches. We evaluate the performance of CT-Meta-AF on KWS with device playback. The AEC removes own-playback from recordings by learning the echo path, as shown in Fig. 6.14. For model details, see Fig. 6.15. We evaluate three scenarios: one with a pre-trained KWS, one with test-time swapped KWS, and one with a jointly trained KWS, all implemented using JAX [141], Haiku [142], and Meta-AF [93]. All CT-Meta-AFs use the classifier for training. The jointly trained KWS is optimized end-to-end and trained simultaneously with the AEC.

To perform AEC, we fit an AF to cancel echo, $\underline{\mathbf{e}}[\tau]$. The signal model is $\underline{\mathbf{d}}[\mathrm{t}] = \sigma(\underline{\mathbf{u}}[\mathrm{t}]) * \underline{\mathbf{w}} + \underline{\mathbf{n}}[\mathrm{t}] + \underline{\mathbf{s}}[\mathrm{t}]$, where $\underline{\mathbf{n}}$ is noise, $\underline{\mathbf{s}}$ is a keyword, $\sigma(\cdot)$ is a nonlinearity, and $\underline{\mathbf{w}}$ is the transfer function. We only assume access to the playback, $\underline{\mathbf{u}}$ and its recording, $\underline{\mathbf{d}}$. We set up KWS as a multi-class classification task, where the keyword $\underline{\mathbf{s}}$ belongs to a single class c. All classification-trained Meta-AFs use the classifier for training. The jointly trained KWS is optimized end-to-end and trained simultaneously to improve performance.

To perform AEC, we fit an AF to cancel the echo. At inference time, we only assume access to the playback, $\underline{\mathbf{u}}$ and echoic mixture, $\underline{\mathbf{d}}$. The AF uses frequency-domain overlap save with a window of $K = 1024$, a hop of $R = 512$, and $B = 4$ blocks. Each block is denoted by $\mathbb{C}$Filter in Fig. 6.15. Each AEC filters the farend $\mathbf{u}[\tau]$ with estimated filter $\hat{\mathbf{w}}[\tau]$ and subtracts the result from $\mathbf{d}[\tau]$. For a review of AEC see Benesty et al. [9]. We compare our approach to four baselines: regular Meta-AF [93], a Kalman filter AEC (Diag. KF) [15], No-Echo, and No-AEC. We grid-search-tune Diag. KF using KWS accuracy. No-Echo simulates running the KWS model in a playback-free environment, while No-AEC performs no echo cancellation and simulates deploying a KWS without AEC.

CT-Meta-AF and Meta-AF use the same higher-order architecture [94] and training scheme, with the only difference being the loss. For CT-Meta-AF, we use $\lambda = 0.5$ in (5.5), and for regular Meta-AF, we use 0. We set the group size, group hop, and hidden size to 5, 2, and 48, respectively (see Fig. 6.15 for details). We use Adam with a batch size of 16, a learning rate of $2 \cdot 10^{-4}$, momentum $\beta_1 = 0.99$, and we randomize the truncation length L. Additionally, we apply gradient clipping and reduce the learning rate by half if the validation performance does not improve for 10 epochs, and we stop training after 30 epochs with no improvement. For pre-trained KWS experiments, we train $\boldsymbol{\phi}$ from scratch, and for joint training, we initialize with the best pre-trained $\boldsymbol{\phi}$. When joint training with Diag. KF or Meta-AF we do not update the pre-trained $\boldsymbol{\phi}$ and just train the KWS. Each model has 32 K complex parameters and takes < 48 hours to train on a single RTX 2080Ti.

We set up KWS as a multi-class classification task, where the keyword $\underline{\mathbf{s}}$ belongs to a single class c. We base our KWS on the model proposed by Cornell et al. [85]. Our version uses 40 log-mel-filter-bank inputs and short-time Fourier transform with a 256 point hop and a 512 point window. The KWS has 3 residual blocks, each with a 1×1 convolution, layer norm, ReLU, a dilated convolution with kernel size 5, layer norm, ReLU, and a final 1×1 convolution. The last residual block is averaged across time and fed to a dense layer with softmax to predict the class distribution. See Fig. 6.15. We use a binary cross-entropy loss and train the KWS for 50 epochs with a batch size of 128, a learning rate of 10^{-3}, and use the best checkpoint based on validation performance. We pre-train on a dataset without playback. For joint training, due to limited GPU capacity, we use a batch size of 16, a learning rate of 10^{-4}, $\beta_1 = .9$, reduce the learning rate by half after 10 epochs with no improvement, and stop training after 50 epochs with no improvement. All KWS models have $\approx$ 300K parameters and use $\approx$ 20 MFLOPs per second.

We use synthetic playback during training and evaluate our models using both synthetic and real playback data from the Microsoft AEC Challenge [157] and keywords from the Google Speech Commands V2 dataset [168], sampled at 16 KHz. During training, we trim playback to 3 seconds, randomly mix playback with keywords, zero-pad keywords as needed, apply a random shift, and set the signal-to-echo ratio (SER) uniformly at random between -25 dB and 0 dB. Each keyword is used once. To test, we use playback trimmed to either 4 or 12 seconds. We evaluate three datasets: a 35-class dataset and two smaller datasets with 10 and 2 classes, respectively. The smaller datasets contain approximately one-third and one-tenth of the full dataset. We always use the same folds and never mix across folds. To evaluate performance, we use macro and micro averaged F1, where higher is better. The goal is to reflect the class imbalance in the KWS Dataset. We display results as macro F1 (micro F1).

Here, we investigate the effect of classification training for Meta-AF on AEC with a downstream KWS. This task simulates a device attempting to classify a spoken keyword while emitting playback. We show results for zero playback (No-Echo), no cancellation (No-AEC), and Diag. KF, Meta-AF-based AEC without classification training (Meta-AEC), and Meta-AF-based AEC with classification training (CT-Meta-AEC). The Diag. KF and Meta-AEC baselines are state-of-the-art traditional and data-driven approaches. All models are trained on synthetic playback with challenging SERs distributed uniformly at random between -25 dB and 0 dB, and real playback is only used for evaluation. We use an off-the-shelf KWS that is not customized for AEC in sections 6.4.2 and 6.4.3 but do experiment with retraining the KWS in section 6.4.3. We evaluate longer playback and compute statistical significance for select models. Across experiments, CT-Meta-AF proved highly stable and required no additional tuning, making it a drop-in replacement for prior approaches.

Pretrained KWS: First, we test the effectiveness of classification training as described in (5.6). In Table 6.3, all AEC models share a frozen KWS model pre-trained on keywords without playback. This setup mimics using an off-the-shelf KWS without retraining but with access to your own dataset, a common real-world setup. For CT-Meta-AEC training, we use $\lambda = 0.5$ when computing $\mathcal{L}_{CM}$. The F1 degradation from No-Echo to No-AEC shows the importance of AEC.

Model	*Synthetic Playback*	*Real Playback*
No-Echo	.928 (.935)	.931 (.936)
No-AEC	.087 (.098)	.102 (.117)
Diag. KF	.196 (.208)	.165 (.180)
Meta-AEC	.335 (.340)	.226 (.236)
Meta-AEC-KWS	**.500 (.508)**	**.317 (.326)**

Table 6.3: F1 Macro (F1 Micro) with a pretrained KWS and SER $\in [-25, 0]$ dB. No-AEC and No-Echo are lower/upper bounds.

In Table 6.3, CT-Meta-AEC significantly outperforms its signal-level Diag. KF and Meta-AEC counterparts, as indicated by better F1 scores. In synthetic playback (left column), CT-Meta-AEC improves over Meta-AEC by some .165 F1. This gap persists in real playback, where CT-Meta-AEC is the top-performing model by .091 F1. Interestingly, CT-Meta-AEC achieves a 5.80 dB reduction in echo, while regular Meta-AF attains a 9.57 dB reduction. This observation underscores the limitations of signal-level metrics as effective descriptors of downstream performance, especially in the context of nonlinear processing methods such as DNN-based KWS. Classification-based training effectively addresses this discrepancy and optimizes downstream performance. In a secondary evaluation with 12-second signals, CT-Meta-AEC outperforms Meta-AEC by 0.279 F1 in synthetic playback and 0.119 F1 in real playback.

6.4.3 Specialization and Sensitivity

Next, we study the specialization capabilities of CT-Meta-AEC in different KWS setups. We use the approach from (5.7) to train three CT-Meta-AEC models: CT-Meta-AEC-35, CT-Meta-AEC-10, and CT-Meta-AEC-2, corresponding to datasets with 35, 10, and 2 keyword classes, respectively. We assess the performance of each CT-Meta-AEC model on all three datasets, but always use a downstream KWS model specifically trained on a matching keyword setup. Table 6.4 presents the results of our experiments on synthetic playback scenarios, where each row corresponds to a distinct AEC model and each column represents a unique test-time KWS configuration. The first row displays our Meta-AEC baseline. Table entries on the diagonal show matched train/test setups.

In all columns of Table 6.4, CT-Meta-AEC with a matching KWS performs best. This demonstrates that CT-Meta-AEC is learning a specialized optimizer (the diagonal) which outperforms both generalist optimizer (top row) and optimizers trained on different models (off-diagonal). Notably, results above the diagonal show that specialization can yield

Model	*KWS-35*	*KWS-10*	*KWS-2*
Meta-AEC	.335 (.340)	.476 (.465)	.751 (.754)
Meta-AEC-KWS-35	**.500** (**.508**)	.513 (.483)	.786 (.788)
Meta-AEC-KWS-10	.344 (.348)	**.533** (**.529**)	.781 (.781)
Meta-AEC-KWS-2	.274 (.281)	.413 (.400)	**.802** (**.802**)

Table 6.4: Specialization of classification-trained Meta-AEC with performance shown as F1 Macro (F1 Micro). A different model with different keywords is swapped in at test time. SER $\in [-25, 0]$ dB.

Model	*Synthetic Playback*	*Real Playback*
No-Echo	.922 (.928)	.931 (.936)
No-AEC	.609 (.619)	.583 (.593)
Diag. KF	.778 (.781)	.690 (.698)
Meta-AEC*	.870 (.876)	.746 (.754)
Meta-AEC-KWS*	**.871** (**.877**)	**.754** (**.762**)

Table 6.5: F1 Macro (F1 Micro) with a jointly trained KWS and SER $\in [-25, 0]$ dB. No-AEC and No-Echo are lower/upper bounds. An asterix denotes that we averaged three trials.

superior results even in the presence of limited training data. Specifically, Meta-AEC and CT-Meta-AEC-35, which are trained with three times more data than CT-Meta-AEC-10, are surpassed by the latter on 10 class KWS. This suggests that classification training is a data-efficient approach for improving performance on downstream tasks. The results from mismatched train/test setups show that CT-Meta-AECs are good at related tasks, and usually still outperform Meta-AEC.

Jointly Trained KWS: We investigate joint training of the AEC and KWS to simulate having sufficient data for training a complete system, without oracle signal-level supervision. Both the AEC and KWS models undergo joint training, following the approach in (5.7). The KWS models undergo a pre-training phase on scenes devoid of playback, followed by fine-tuning using the output of their corresponding AEC models. This represents a significant departure from the approach outlined in (5.6), where KWS models were trained without consideration for echo presence. We call this approach JCT-Meta-AEC. For statistical robustness, we ran three separate trials of the meta-models.

In Table 6.5, we find that the training scheme from (5.7) improves performance. JCT-Meta-AEC outperforms Meta-AEC by .008 F1 (p-value .01) in real playback scenarios, underscoring the efficacy of classification training for real-world performance. While JCT-Meta-AEC outperforms Meta-AEC on synthetic data, the improvement is not significant. However, on the longer 12-second test set, JCT-Meta-AEC beats Meta-AEC by a margin of

0.057 (p-value .0005). Of note, we did not encounter any training stability issues.

Varying the size of Meta-AEC and Meta-AEC-KWS models showed that the performance gap increased with size. At all sizes, the performance gap increased with SER, implying a better KWS architecture could improve performance significantly. Meta-AEC-KWS was also tolerant to quantization, unlike classic AFs. Finally, Meta-AEC is real-time capable and KWS improvements come without additional inference costs.

In this section, we proposed a simple yet effective approach for improving the performance of downstream classification tasks by incorporating classifier feedback into learned adaptive filter update rules. Our approach, classification-trained meta-adaptive filtering (CT-Meta-AF) enables end-to-end training without oracle single-level supervision. We evaluated CT-Meta-AF on echo cancellation and keyword spotting and observed consistent performance gains across synthetic and real playback, multiple keyword configurations, and both pre and jointly-trained keyword models. CT-Meta-AF yields performance improvements without additional inference costs or manual tuning. We believe our approach has the potential to improve many adaptive filter pipelines, thanks to its plug-and-play design and promising real-world performance.

6.5 EQUALIZATION

Inverse modeling is a class of AF problems where the goal is to invert some unknown system. In equalization, the goal is to estimate the inverse of an unknown transfer function, while only observing the input and outputs of the forward system, as shown in Fig. 6.16. This is a common component of loudspeaker tuning or active noise control. We model the unknown inverse transfer function with a linear frequency-domain filter $h_{\boldsymbol{\theta}}$, measure the response $\mathbf{d}$ to an input signal $\mathbf{u}$, and adapt the filter weights $\boldsymbol{\theta}$ using $g_{\boldsymbol{\phi}}$. The AF loss is usually the ISE between the true and predicted responses. More precisely, the frequency-domain AF output is $\mathbf{y}_{\mathrm{k}}[\tau] = \mathbf{w}_{\mathrm{k}}[\tau]^{\mathsf{H}}\mathbf{u}_{\mathrm{k}}[\tau]$.

We measure performance with signal SNR_d, and system SNR_w. We define these as

$$\mathrm{SNR}_d(\underline{\mathbf{d}}, \underline{\mathbf{y}}) = 10 \cdot \log_{10}\left(\frac{||\underline{\mathbf{d}}||^2}{||\underline{\mathbf{d}} - \underline{\mathbf{y}}||^2}\right) \tag{6.6}$$

$$\mathrm{SNR}_w(\hat{\mathbf{w}}^{-1}, \mathbf{w}^{-1}) = 10 \cdot \log_{10}\left(\frac{|||\mathbf{w}^{-1}|||^2}{|||\mathbf{w}^{-1}| - |\hat{\mathbf{w}}^{-1}|||^2}\right) \tag{6.7}$$

respectively, where higher is better. We compute SNR_w using the inverse system magnitude, which ignores the phase.

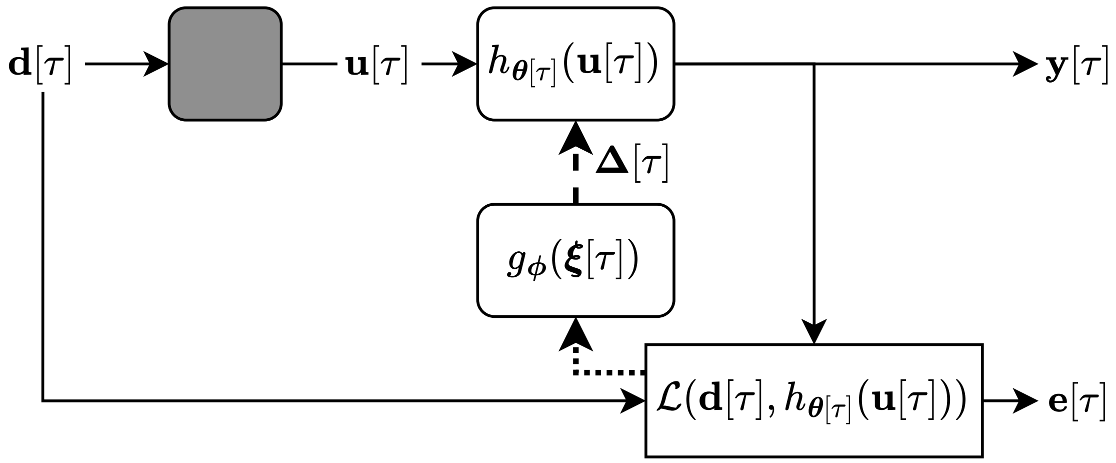

$\mathbf{d}[\tau]$
$\mathbf{u}[\tau]$
$h_{\boldsymbol{\theta}[\tau]}(\mathbf{u}[\tau])$
$\mathbf{y}[\tau]$
$\boldsymbol{\Delta}[\tau]$
$g_{\boldsymbol{\phi}}(\boldsymbol{\xi}[\tau])$
$\mathcal{L}(\mathbf{d}[\tau], h_{\boldsymbol{\theta}[\tau]}(\mathbf{u}[\tau]))$
$\mathbf{e}[\tau]$

Figure 6.16: Inverse modeling block diagram. System outputs are fed to the adaptive filter. The adaptive filter is continually updated to invert the unknown system (shaded box).

6.5.1 Prior Works

Equalization (EQ) can be formulated as an inverse modeling problem, has been explored in single- and multi-channel formats, and is particularly useful for sound zone reproduction and active feedback or noise control [19, 20, 21, 22, 23, 24, 169]. We compare our Meta-EQ approach to LMS, RMSProp, NLMS, and D-RLS on the task of frequency equalization.

6.5.2 Evaluation

We train $g_{\boldsymbol{\phi}}$ via Algorithm 3.1 on one GPU, which took at most 36 h. We use an OLS filter with a window size of $N = 1024$ samples and a hop of $R = 512$ samples on 16 kHz audio. To construct the equalization dataset, we use speech from the DAPS dataset [170], take the *cleanraw* recordings as inputs, and apply random equalizer filters from the sox library to generate the outputs, where we randomly pick between $[5, 15]$ filters with settings $c \in [1, 8]$ kHz, $g \in [-18, 18]$, and $q \in [.1, 10]$. All values are sampled uniformly at random to produce $16,384$ train, 2048 validation, and 2048 test signals, all 5 seconds long. At train, validation, and test time we truncate the system response to 512 taps. Additionally, we ablate the equalization filtering mechanics for two cases: constrained and unconstrained filters (optimizee architecture modifications). In the constrained case, we set $h_{\boldsymbol{\theta}}$ to use standard OLS. However, in the unconstrained case, we set $h_{\boldsymbol{\theta}}$ to use aliased OLS where

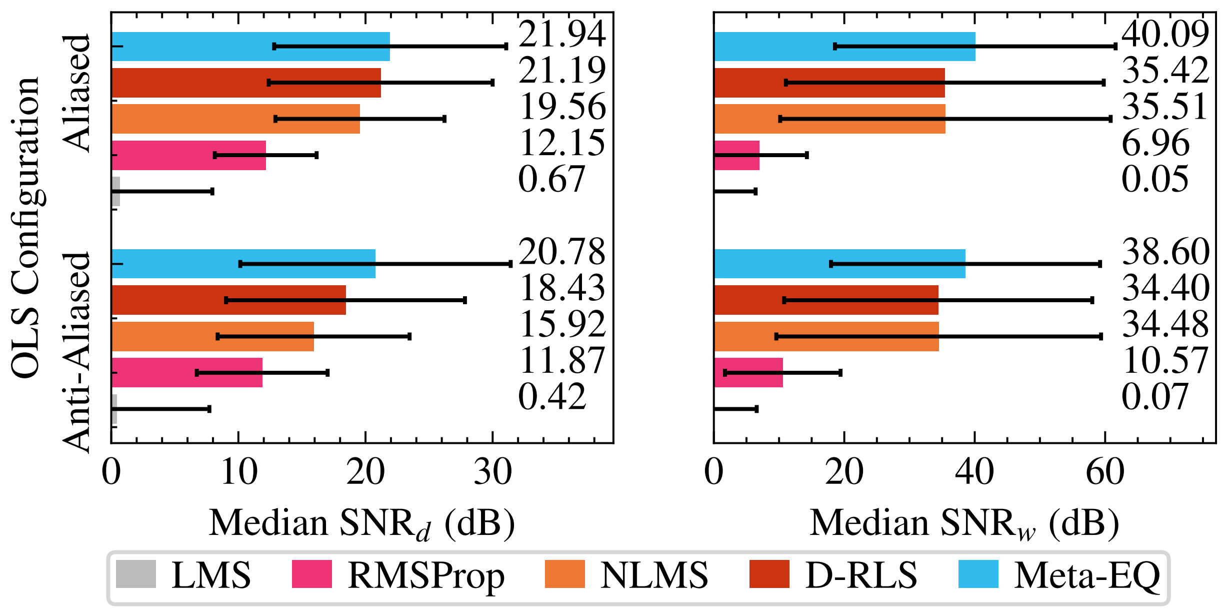


Figure 6.17: Equalization results for signal (SNR$_d$) and system (SNR$_w$) SNR. Meta-EQ performance is the least impacted by constraints.

$\mathbf{Z}_w = \mathbf{I}_K$. This comparison lets us test if Meta-EQ is automatically learning constraint-aware update rules. We train a new g_ϕ for each case (no separate tuning) and tune all baselines for each case.

Results & Discussion: We find our approach (blue, solid) outperforms LMS, RMSProp, NLMS, and D-RLS for our equalization task by a noticeable margin as shown in Fig. 6.17 and further verify with a qualitative analysis plot in Fig. 6.18.

Constrained vs Unconstrained: For the unconstrained case, our method outperforms D-RLS in SNR$_d$ by .75 dB and by 4.67 dB in SNR$_w$. When we look at the constrained case, the performance for all models is degraded. Interestingly, however, our performance is proportionally degraded the least. We hypothesize that our approach learns to perform updates that are aware of the constraint.

Temporal Performance Analysis: We display the final system and convergence results in Fig. 6.18. Our Meta-EQ model finds better solutions more rapidly than D-RLS. D-RLS diverged ≈ 300 times but Meta-EQ never did.

Computational Complexity: Our learned AF has a single CPU core RTF of ≈ 0.24, and 32 ms latency. Our optimizer network alone has $\approx 14K$ complex-valued parameters and a single CPU core RTF of ≈ 0.19.

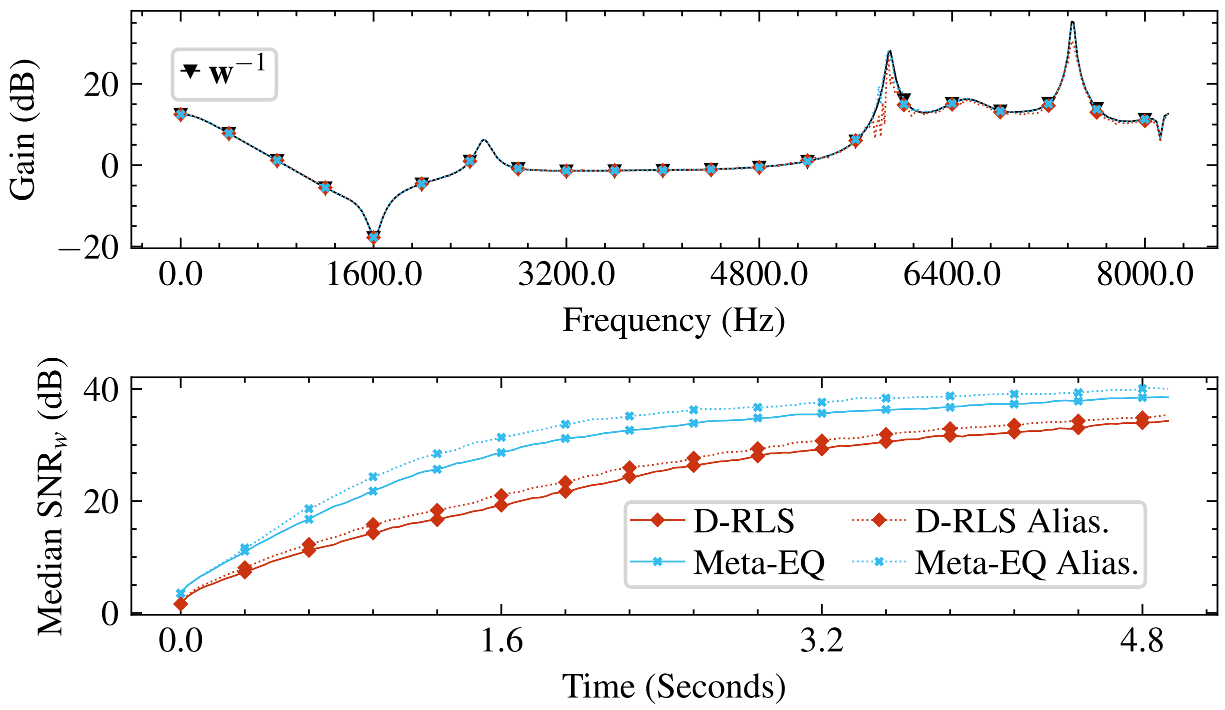


Figure 6.18: Comparison of true and estimated systems over time. The Meta-EQ system rapidly fits to the correct inverse model. The top plot shows an example system and the bottom shows SNR$_w$ over time across the test set.

6.6 DEREVERBERATION

Prediction tasks are widespread in AFs. In some prediction tasks recovering the system is the goal and in others, the predicted values are the most important. In dereverberation, the goal is to remove reverb from a signal. We do this via multi-channel linear prediction (MCLP) or the weighted prediction error (WPE) formulation, as is commonly used for speech-to-text pre-processing. The WPE formulation is based on the idea of being able to predict the reverberant part of a signal from a linear combination of past samples, most commonly in the frequency-domain [25, 26] and shown as a block diagram in Fig. 6.19. We use a multi-channel linear frequency-domain filter $h_{\boldsymbol{\theta}}$ and adapt the filter weights $\boldsymbol{\theta}$ using a $g_{\boldsymbol{\phi}}$. Often, normalized ISE is the loss.

Assuming an array of M microphones, we estimate a dereverberated signal with a linear model via

$$\hat{s}_{\mathrm{km}}[\tau] = d_{\mathrm{km}}[\tau] - \mathbf{w}_{\mathrm{k}}[\tau]^{\mathsf{H}}\mathbf{u}_{\mathrm{k}}[\tau] \tag{6.8}$$

where $\hat{s}_{\mathrm{km}}[\tau] \in \mathbb{C}$ is the current dereverberated signal estimate at frequency k and channel m, $d_{\mathrm{km}}[\tau] \in \mathbb{C}$ is the input microphone signal, $\mathbf{w}_{\mathrm{k}}[\tau] \in \mathbb{C}^{BM}$ is a per frequency filter with B

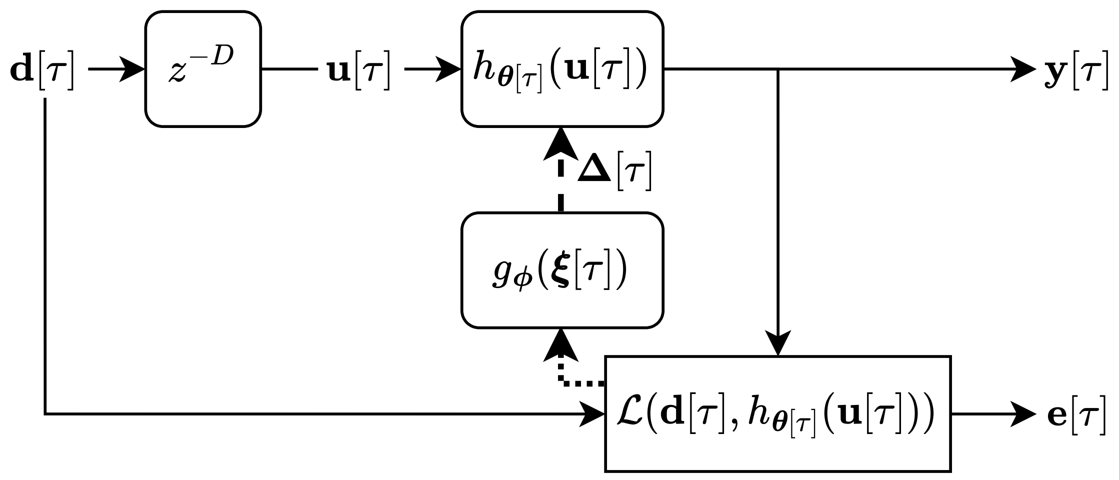

$\mathbf{d}[\tau]$
z^{-D}
$\mathbf{u}[\tau]$
$h_{\boldsymbol{\theta}[\tau]}(\mathbf{u}[\tau])$
$\mathbf{y}[\tau]$
$\boldsymbol{\Delta}[\tau]$
$g_{\boldsymbol{\phi}}(\boldsymbol{\xi}[\tau])$
$\mathcal{L}(\mathbf{d}[\tau], h_{\boldsymbol{\theta}[\tau]}(\mathbf{u}[\tau]))$
$\mathbf{e}[\tau]$

Figure 6.19: Prediction block diagram. A buffer of past inputs are used to estimate a future, unknown signal. The delay, z^{-D} signifies a delay by D frames.

time frames and M channels flattened into a vector, and $\mathbf{u}_\mathrm{k}[\tau] \in \mathbb{C}^{BM}$ is a running flattened buffer of $\mathbf{d}_\mathrm{k}[\tau - D]$.

We then minimize a per channel and frequency loss

$$\mathcal{L}(\hat{s}_\mathrm{km}[\tau], \lambda_\mathrm{k}[\tau]) = \frac{\|\hat{s}_\mathrm{km}[\tau]\|^2}{\lambda_\mathrm{k}^2[\tau]}, \tag{6.9}$$

$$\lambda_\mathrm{k}^2[\tau] = \frac{1}{M(B+D)} \sum_{\mathrm{n}=\tau-B-D}^{\tau} \mathbf{d}_\mathrm{k}[\mathrm{n}]^\mathsf{H} \mathbf{d}_\mathrm{k}[\mathrm{n}], \tag{6.10}$$

where $\lambda_\mathrm{k}^2[\tau]$ is a running average estimate of the signal power and $\mathbf{d}_\mathrm{k}[\tau] \in \mathbb{C}^M$. We use this formation within our framework to perform online multi-channel dereverberation or Meta-WPE and focus on dereverberating a single output channel.

We measure performance with two metrics, segmental speech-to-reverberation ratio (SRR) [31] and STOI. SRR is a signal level metric and measures how much energy was removed from the signal. It is computed as

$$\mathrm{SRR}(\mathbf{d}_\mathrm{k}[\tau], \hat{\mathbf{s}}_\mathrm{k}[\tau]) = 10 \cdot \log_{10}\left(\frac{\|\hat{\mathbf{s}}_\mathrm{k}[\tau]\|^2}{\|\mathbf{d}_\mathrm{k}[\tau] - \hat{\mathbf{s}}_\mathrm{k}[\tau]\|^2}\right), \tag{6.11}$$

where smaller values indicate more removed energy and better performance. STOI is computed between the dereverberated signal estimate and the ground truth anechoic signal.

6.6.1 Prior Works

Dereverberation can be formulated as a weighted prediction (WPE) problem and has been extensively studied in this context [25, 26, 27, 28, 29, 30, 31]. We compare our Meta-WPE to frame-online NARA-WPE [30], a BD-RLS-based AF that uses the WPE formulation.

6.6.2 Evaluation

We train g_ϕ via Algorithm 3.1 on two GPUs, which took at most 24 h. We use an OLA filter with a Hann window size of $N = 512$ samples and a hop of $R = 256$ samples on 16 kHz audio. We fix the buffer size $B = 5$ taps and the delay to $D = 2$ frames. We use the simulated REVERB challenge dataset [171]. The REVERB challenge contains echoic speech mixed with noise at 20 dB in small (T60 = .25 sec), medium (T60 = .5 sec) and large (T60 = .7 sec) rooms at near and far distances. The array is circular with a diameter of 20 cm. Background noises are generally stationary. The dataset has 7861 training files, 1484 validation files, and 2176 test files. We ablate the filter size and inputs across $M = 1, 4, 8$ microphones (optimizee size and input modification). We seek to test if our method can scale from single- to multi-channel tasks without modification. We train a new g_ϕ for each M (no tuning). We find that our approach (blue, solid) outperforms NARA-WPE in SRR across all filter configurations, but is worse in STOI as shown in Fig. 6.20. We discuss this below.

Overall and Temporal Performance As shown in Fig. 6.20, Meta-WPE (blue, solid) scores strongly on SRR, where our single-channel Meta-WPE model scores better than 4 and 8 channel NARA-WPE (red, dotted) models. However, as shown by STOI, the perceptual quality is poor. While Meta-WPE is solving the prediction more rapidly, as shown by segmental SRR, it is not doing so in a perceptual pleasing manner. Previous studies [31, 172] have also encountered this phenomenon, and propose a variety of regularization tools to align the instantaneous optimization objective with perceptually pleasing processing. We re-ran these experiments with a buffer of size $B = 10$ as well as with larger and smaller optimizer network capacities and found this trend was consistent. As a result, we conclude our approach is very effectively improving the online optimization of the target loss, but the instantaneous loss itself needs to be changed to better align with perception.

Computational Complexity The 4 channel learned AF has a single CPU core RTF of ≈ 0.47, and 16 ms latency. Our Meta-WPE optimizer network alone has $\approx 17K$ complex-valued parameters and single CPU core RTF of ≈ 0.38.

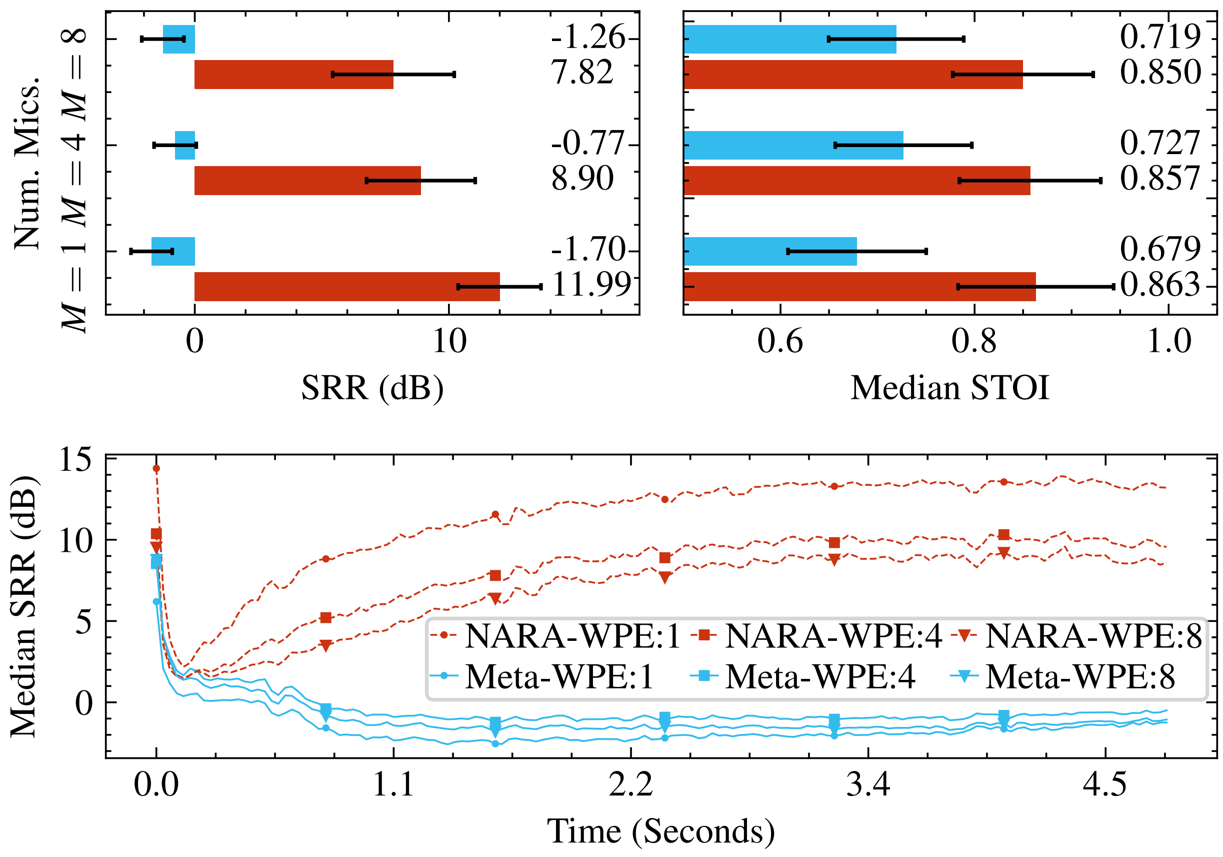


Figure 6.20: Dereverberation performance in terms of SRR. Meta-WPE excels in SRR, a metric that measures energy removed. However, in STOI, Meta-WPE scores worse.

6.7 ISOLATED BEAMFORMING

In interference cancellation, the task is to process some signal given some sort of processed or otherwise extracted signal. In this task, the goal is to remove the interfering noise provided spatial information about the source of interest. We do this by using the minimum variance distortionless response (MVDR) beamformer. The MVDR beamformer can be implemented as an AF using the generalized sidelobe canceller (GSC) [38] formulation and is commonly used for far-field voice communication and speech-to-text pre-processing. We depict a version of this problem setup in Fig. 6.21 and use a linear frequency-domain filter $h_{\boldsymbol{\theta}}$. We use the mixture $\mathbf{d}[\tau]$ as the target, informed input $\mathbf{u}[\tau]$, and adapt the filter weights $\boldsymbol{\theta}[\tau]$ using our learned Meta-GSC AF $g_{\boldsymbol{\phi}}$ and ISE AF loss.

Assuming an array of M microphones, we have the time-domain signal model for mic m via,

$$\underline{u}_{\mathrm{m}}[\mathrm{t}] = \underline{r}_{\mathrm{m}}[\mathrm{t}] * \underline{s}[\mathrm{t}] + \underline{n}_{\mathrm{m}}[\mathrm{t}] \tag{6.12}$$

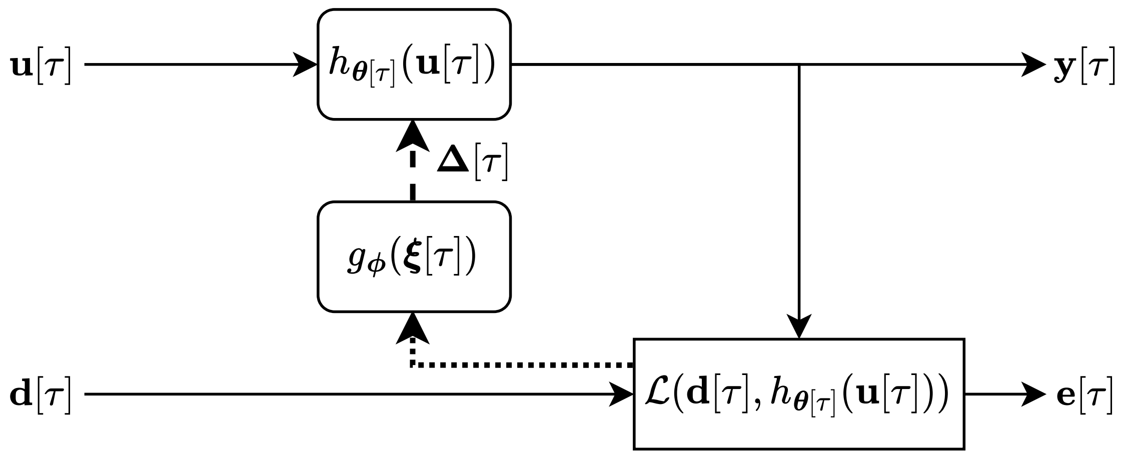

$\mathbf{u}[\tau]$
$h_{\boldsymbol{\theta}[\tau]}(\mathbf{u}[\tau])$
$\mathbf{y}[\tau]$
$\boldsymbol{\Delta}[\tau]$
$g_{\boldsymbol{\phi}}(\boldsymbol{\xi}[\tau])$
$\mathbf{d}[\tau]$
$\mathcal{L}(\mathbf{d}[\tau], h_{\boldsymbol{\theta}[\tau]}(\mathbf{u}[\tau]))$
$\mathbf{e}[\tau]$

Figure 6.21: Informed interference cancellation block diagram. An auxiliary signal is used as input to an adaptive filter which is fit to an alternate signal.

where $\underline{u}_{\mathrm{m}}[\mathrm{t}] \in \mathbb{R}$ is the input signal, $\underline{n}_{\mathrm{m}}[\mathrm{t}] \in \mathbb{R}$ is the noise signal, $\underline{s}[\mathrm{t}] \in \mathbb{R}$ is the target signal, and $\underline{r}_{\mathrm{m}}[\mathrm{t}] \in \mathbb{R}$ is the impulse response from the source to mic m. In the time-frequency domain with a sufficiently long window, this can be reformulated as

$$u_{\mathrm{km}}[\tau] = r_{\mathrm{km}}[\tau] s_{\mathrm{k}}[\tau] + n_{\mathrm{km}}[\tau]. \tag{6.13}$$

The GSC beamformer also assumes access to a steering vector, $\mathbf{v}_{\mathrm{k}}$. While estimating the steering vector is well studied [38], it remains non-trivial for real-world applications. For our case, we assume access to a clean speech recording $\mathbf{s}_{\mathrm{k}}[\tau]$ and first compute

$$\boldsymbol{\Phi}_{\mathrm{k}}^{ss}[\tau] = \gamma \boldsymbol{\Phi}_{\mathrm{k}}^{ss}[\tau - 1] + (1 - \gamma)(\mathbf{s}_{\mathrm{k}}[\tau]\mathbf{s}_{\mathrm{k}}[\tau]^{\mathsf{H}} + \lambda \mathbf{I}_{\mathrm{M}}) \tag{6.14}$$

where $\boldsymbol{\Phi}_{\mathrm{k}}^{ss}[\tau] \in \mathbb{C}^{M \times M}$ is a time-varying estimate of the target signal spatial covariance matrix, γ is a forgetting factor, and λ is a regularization parameter. We then estimate the steering vector by computing the normalized first principal component of the target source covariance matrix,

$$\tilde{\mathbf{v}}_{\mathrm{k}}[\tau] = \mathcal{P}(\boldsymbol{\Phi}_{\mathrm{k}}^{ss}[\tau]) \tag{6.15}$$

$$\mathbf{v}_{\mathrm{k}}[\tau] = \tilde{\mathbf{v}}_{\mathrm{k}}[\tau] / \tilde{v}_{\mathrm{k}0}[\tau] \tag{6.16}$$

where $\mathcal{P}(\cdot)$ extracts the principal component and $\mathbf{v}_{\mathrm{k}}[\tau] \in \mathbb{C}^{M}$ is the final steering vector.

We then use the steering vector to estimate a blocking matrix $\mathbf{B}_{\mathrm{k}}[\tau]$. The blocking matrix is orthogonal to the steering vector and can be constructed as

$$\mathbf{B}_{\mathrm{k}}[\tau] = \begin{bmatrix} -\frac{[v_{\mathrm{k}1}[\tau], \cdots, v_{\mathrm{k}M}[\tau]]^{\mathsf{H}}}{v_{\mathrm{k}0}[\tau]^{\mathsf{H}}} \\ \mathbf{I}_{M-1\times M-1} \end{bmatrix} \in \mathbb{C}^{M\times M-1}. \tag{6.17}$$

The distortionless constraint is then satisfied by applying the GSC beamformer as

$$\hat{s}_{\mathrm{k}}[\tau] = (\mathbf{v}_{\mathrm{k}}[\tau] - \mathbf{B}_{\mathrm{k}}[\tau]\mathbf{w}_{\mathrm{k}}[\tau])^{\mathsf{H}}\mathbf{u}_{\mathrm{k}}[\tau] \tag{6.18}$$

where $\mathbf{w}_{\mathrm{k}}[\tau] \in \mathbb{C}^{M-1}$ is the adaptive filter weight, and the desired response for the loss is $d_{\mathrm{k}}[\tau] = \mathbf{v}_{\mathrm{k}}[\tau]^{\mathsf{H}}\mathbf{u}_{\mathrm{k}}[\tau]$.

Our objective is to learn an optimizer g_{ϕ} that minimizes the AF ISE loss using this GSC filter implementation. By doing so, we learn an online, adaptive beamformer that listens in one direction and suppresses interferers from all others.

We measure performance using scale-invariant source-to-distortion ratio (SI-SDR) [161] and STOI. SI-SDR is computed as

$$\mathbf{a} = (\underline{\hat{\mathbf{s}}}^{\top}\underline{\mathbf{s}})/\|\underline{\mathbf{s}}\| \tag{6.19}$$

$$\text{SI-SDR}(\underline{\mathbf{s}}, \underline{\hat{\mathbf{s}}}) = 10 \cdot \log_{10}(\|\mathbf{a}\underline{\mathbf{s}}\|^2/\|\mathbf{a}\underline{\mathbf{s}} - \underline{\hat{\mathbf{s}}}\|^2), \tag{6.20}$$

where larger values indicate better performance. STOI is computed between the output and desired speech signal. We also compute the BSS eval metrics, source-to-distortion ratio (SDR), source-to-interference ratio (SIR), and source-to-artifact ratio (SAR) [173].

6.7.1 Prior Works

Finally, multi-microphone enhancement or beamforming (BF) can be formulated as an informed interference cancellation task and also has a breadth of associated BF approaches [32, 33, 34, 35, 36, 37, 38] and unmixing algorithms [39, 40, 41, 42, 43, 44].

6.7.2 Experimental Design

We compare our Meta-GSC beamformer to LMS, RMSProp, NLMS, and BD-RLS beamformers, in scenes with either diffuse or directional noise sources.

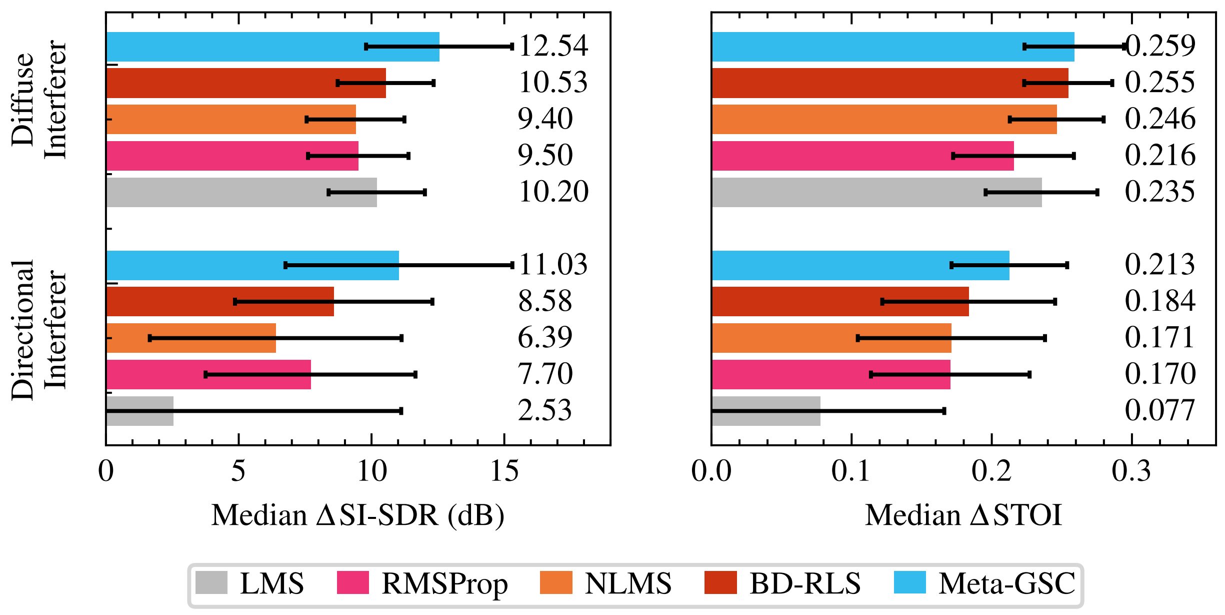


Figure 6.22: Performance comparison across interferers. The directional noise is the most challenging and diffuse is the easiest.

6.7.3 Evaluation

We use the CHIME-3 challenge proposed in [174, 175]. This dataset contains scenes with simulated speech and relatively diffuse noise sources in a multi-channel environment. The array is rectangular and has six microphones spaced around the edge of a smart tablet. There are $7,138$ training files, $1,640$ validation files, and $1,320$ test files. When running this dataset with directional sources, we mix spatialized speech from one mixture with the spatialized speech from a random other mixture. We do not mix speech across folds.

We find that Meta-GSC outperforms LMS, RMSProp, NLMS, and BD-RLS in median performance metrics as shown in Fig. 6.22 and Fig. 6.23 and in a qualitative analysis in Fig. 6.24.

Diffuse Interferers: The diffuse scenario tests the ability to suppress omnidirectional noise in a perceptually pleasant fashion. We show these comparisons in the "Diffuse Interferer" rows of Fig. 6.22 and Fig. 6.23. The median input STOI was 0.675 and the median input SI-SDR was -0.67. Meta-GSC (blue, solid) outperforms BD-RLS (red, dotted) in SI-SDR performance with Meta-GSC scoring 12.54 dB improvement and BD-RLS scoring 10.53 dB improvement. In STOI, Meta-GSC outperforms BD-RLS by 0.004. The BSS Eval metrics show that Meta-GSC provides 5.8 dB more interferer suppression (SIR) while simultaneously introducing 1.4 dB fewer artifacts (SAR) and 2.02 dB less distortion (SDR) than BD-RLS. Typically, enhancement algorithms trade improved interference suppression for additional artifacts. However, the meta-training scheme produces an optimizer that simultaneously

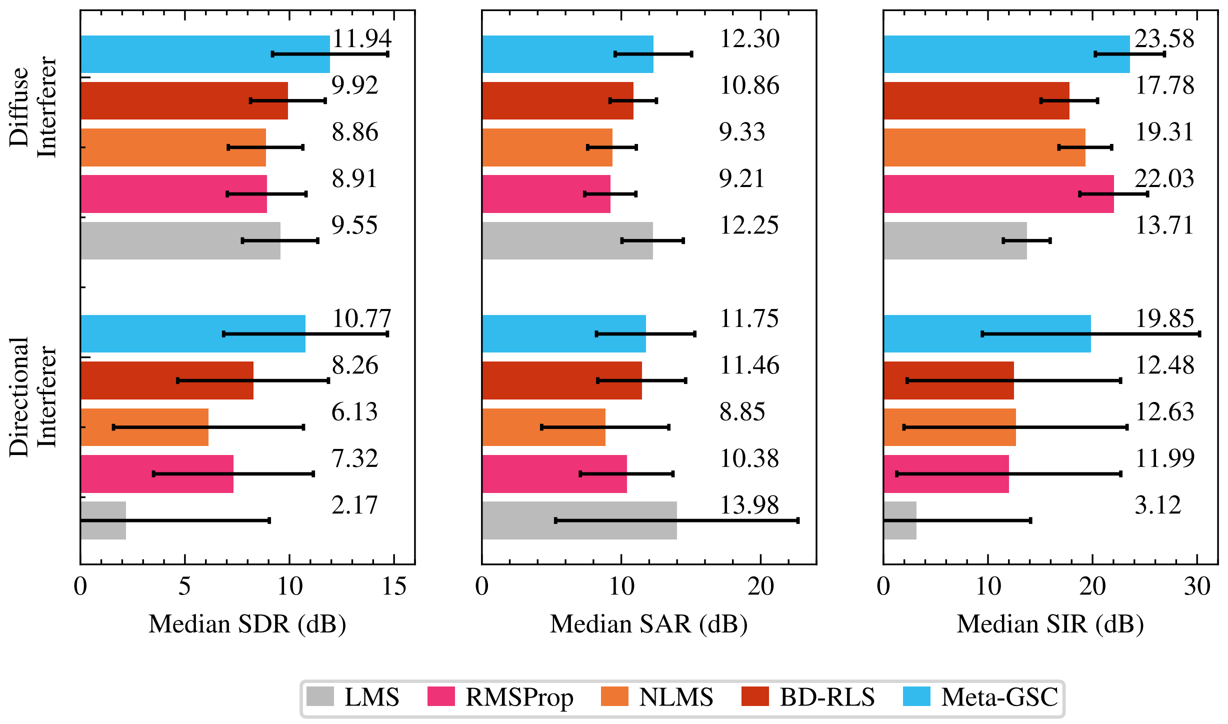


Figure 6.23: BSS eval comparison across interferers. Meta-GSC provides more suppression with less distortion and artifacts.

improves both.

Directional Interferers: The directional scenario tests the ability to suppress sources from one particular direction – typically achieved by steering nulls in the beam pattern. We show these comparisons in the "Directional Interferer" rows of Fig. 6.22 and Fig. 6.23. The median input STOI was 0.734 and the median input SI-SDR was -0.45. Meta-GSC scores 11.03 dB on SI-SDR improvement whereas BD-RLS scores 8.58 dB. STOI performance trends similarly with Meta-GSC outperforming BD-RLS by 0.029. The BSS-Eval metrics show a similar trend with Meta-GSC providing 7.37 dB more SIR while simultaneously introducing .29 dB fewer artifacts (SAR) and 2.51 dB less distortion (SDR) than BD-RLS. We hypothesize that Meta-GSC steers sharper nulls and learns an automatic VAD-like controller.

Beampattern Comparison: We compute beam plots for Meta-GSC and BD-RLS at ≈ 1 sec. and ≈ 2 sec. in a scene with a directional interferer. As expected, the models share the same look direction. However, our Meta-GSC method appears to steer more aggressive nulls as shown in Fig. 6.24

Computational Complexity: Our Meta-GSC method has a single CPU core RTF of ≈ 0.54, and 32 ms latency. The optimizer network alone has $\approx 14K$ complex-valued parameters and a single CPU core RTF of ≈ 0.25.

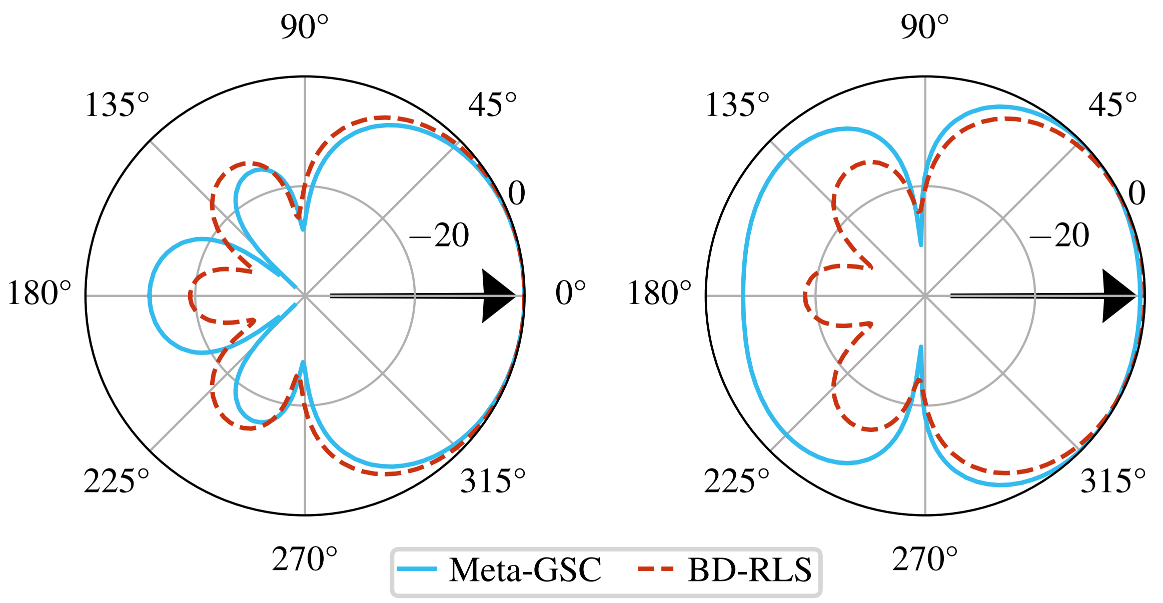


Figure 6.24: Spatial response plots at ≈ 1 KHz for a directional interferer at ≈ 1 sec. (left) and ≈ 2 sec. (right).

6.8 SCALING-UP BEAMFORMING

Multi-channel speech enhancement using beamforming is a core task for smart systems and researchers have been developing better solutions for decades. Typically, for better performance, researchers need to invent or otherwise develop novel approaches. In this section, we attempt to get better performance just by deploying more compute within an existing method. The goal of our experiments is to confirm the AEC results for SMS-AF and demonstrate how it scales across the task of beamforming. We take the same GSC setup from before and we again vary 1) optimizer model sizes with small (S), medium (M), and large (L) models 2) an unsupervised (U) or supervised (S) loss and 3) the number of predict (P) and update (U) steps per frame. We again label baseline methods when applicable.

6.8.1 Prior Works

We compare against NLMS, recursive-lease-squares (RLS), and Meta-AF. We again test multiple HO-Meta-AF model sizes as well as multi-step NLMS, RLS, and HO-Meta-AF. We used the CHIME-3 [174] dataset which as 7.1K training, 1.6K validation, and 1.3K test files. CHIME-3 uses a rectangular six-microphone array.

6.8.2 Evaluation

For speech enhancement, we use a single-block frequency-domain GSC beamformer with 64ms latency. The signal model at each of M microphones is $\underline{\mathbf{u}}_{\mathrm{m}}[\mathrm{t}] = \underline{\mathbf{r}}_{\mathrm{m}}[\mathrm{t}] * \underline{\mathbf{s}}[\mathrm{t}] + \underline{\mathbf{n}}_{\mathrm{m}}[\mathrm{t}]$, and $\underline{\mathbf{r}}_{\mathrm{m}}[\mathrm{t}]$ is the impulse response from source to mic m. The goal is to recover the clean speech $\underline{\mathbf{s}}$ given the input signal $\underline{\mathbf{u}}_{\mathrm{m}}[\mathrm{t}]$. This requires fitting a filter to remove the effects of noise, $\underline{\mathbf{n}}_{\mathrm{m}}$. We assume access to a steering vector. For a review, see Gannot et al. [38]

We use higher-order Meta-AF optimizers with banded coupling and a group size of 5. The training setup is identical to the AEC scaling section except that we use scale-invariant signal-to-distortion ratio (SI-SDR) as our loss.

Beamforming results are in Table 6.6. Notably, SMS-AF improvements apply without any modifications. Here, all models assume access to a steering vector, which can be challenging to estimate in practice. Again, supervision and multi-step optimization yield significant performance gains. Our S·S·P model outperforms all baselines including L·U·P across all metrics. Our model scales reliably with the L·S·P variant improving performance in all metrics. Scaling up the iterations to S·U·PUx2 yields larger gains across all metrics. Our largest and best model, L·S·PUx2 scores a remarkable 17.72 dB SI-SDR while still being real-time. Of note, the L·S·PU model has the same RTF as RLS·PU, even though RLS uses fewer operations. Thus, for a second task, we show that SMS-AF performance scales with both model capacity and optimization steps per frame.

Across this GSC and the AEC experiments, we have introduced a new method for neural network-based adaptive filter optimizers called supervised multi-step adaptive filters (SMS-AF). To do so, we extend recent meta-adaptive filtering methods with several critical advances that cohesively work together to reliably increase performance by simply leveraging more computation. We evaluate our proposed method on low latency, online AEC, and GSC AF tasks, compare against many baselines, and test on both synthetic and real data. Our method improves both subjective and objective metrics, achieving ≈ 5 dB ERLE/SI-SDR gains compared to prior work, and increases the performance ceiling across AEC and GSC. Furthermore, we relate our work to the Kalman filter and meta-AFs, giving insight into many other applications. We believe scaling up AFs is a promising research direction and hope our results encourage future work on scalable, general-purpose AFs.

6.9 BEAMFORMING, STEERING AND POST-FILTERING

Beamforming is an integral part of more comprehensive speech capture systems, often accompanied by a preceding steering or direction-of-arrival system and followed by an en-

Model	SI-SDR↑	SIR↑	SAR↑	STOI↑	MFLOPs↓	RTF↓
Mixture	-0.71	-	-	0.674	-	-
NLMS·P	8.60	16.21	9.78	0.905	0.43	0.36
NLMS·PU	8.84	16.54	10.00	0.910	0.47	0.47
RLS·P	9.84	16.70	9.70	0.919	0.53	0.50
RLS·PU	10.14	17.16	11.49	0.924	0.54	0.62
S·U·P	12.20	22.57	12.79	0.931	4.70	0.41
M·U·P	12.62	22.56	13.26	0.938	12.08	0.45
L·U·P	12.45	22.43	13.09	0.935	36.35	0.53
S·S·P	13.92	23.00	14.66	0.950	4.70	0.41
M·S·P	14.34	23.45	15.07	0.953	12.08	0.45
L·S·P	14.69	24.36	15.33	0.954	36.35	0.53
S·S·PU	15.46	25.69	16.09	0.956	4.74	0.51
M·S·PU	16.83	27.70	17.41	0.960	12.12	0.54
L·S·PU	17.22	28.37	17.80	0.962	36.39	0.62
S·S·PUx2	15.67	25.89	16.35	0.956	9.07	0.70
M·S·PUx2	17.06	28.42	17.63	0.961	23.83	0.76
L·S·PUx2	**17.72**	**29.52**	**18.25**	**0.964**	72.37	0.91

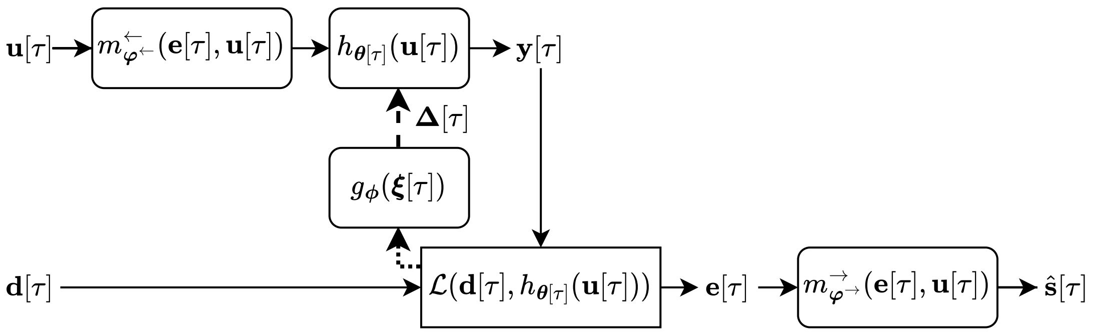
Table 6.6: GSC Performance vs. Computational Cost on the standardized Chime 3 dataset.

hancement module. In this context, we establish a complete capture system, treating each step as an individual neural network task. This approach results in a sequence of neural network modules that amalgamate spectral and spatial data over time, even in intricate scenarios. To gain a broader understanding of this setup, please refer to Fig. 6.25, in which a pre-processor gathers information that is subsequently directed to a Meta module, and finally processed by a post-processor.

For a more detailed view of this setup, see Fig. 6.26. This pipeline strings together an enhancer, a covariance estimator, and a beamformer. The estimator outputs are translated into spatial covariance matrices and fed to the beamformer. The convolutions (blue, solid edges) are applied across frequencies whereas the LSTM, Linear layers, and beamformer (yellow, dashed edges) are applied separately to each frequency. We train this pipeline using on various datasets in the next section.

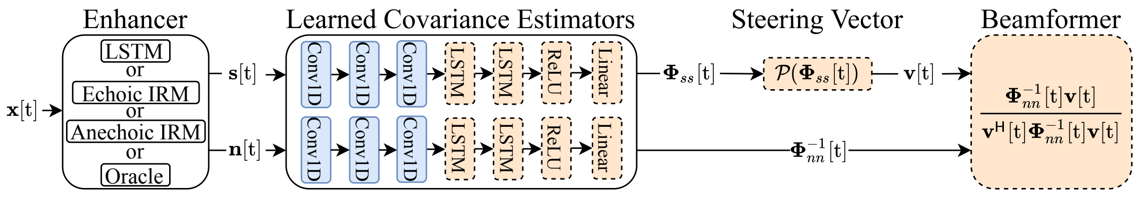
6.9.1 Prior Works

The growing success of deep neural networks (DNN) in single-channel spectral separation and speech enhancement has led to several works integrating single- and/or multi- channel



$\mathbf{u}[\tau]$
$m^{\leftarrow}_{\varphi^{\leftarrow}}(\mathbf{e}[\tau],\mathbf{u}[\tau])$
$h_{\theta[\tau]}(\mathbf{u}[\tau])$
$\mathbf{y}[\tau]$
$\boldsymbol{\Delta}[\tau]$
$g_{\phi}(\boldsymbol{\xi}[\tau])$
$\mathbf{d}[\tau]$
$\mathcal{L}(\mathbf{d}[\tau],h_{\theta[\tau]}(\mathbf{u}[\tau]))$
$\mathbf{e}[\tau]$
$m^{\rightarrow}_{\varphi^{\rightarrow}}(\mathbf{e}[\tau],\mathbf{u}[\tau])$
$\hat{\mathbf{s}}[\tau]$

Figure 6.25: Meta-AF for GSC with a DNN steering vector estimator and a DNN speech enhancer.



Enhancer
LSTM
or
Echoic IRM
or
Anechoic IRM
or
Oracle
$\mathbf{x}[t]$
$\mathbf{s}[t]$
$\mathbf{n}[t]$
Learned Covariance Estimators
Conv1D
Conv1D
Conv1D
LSTM
LSTM
ReLU
Linear
$\boldsymbol{\Phi}_{ss}[t]$
Steering Vector
$\mathcal{P}(\boldsymbol{\Phi}_{ss}[t])$
$\mathbf{v}[t]$
$\boldsymbol{\Phi}^{-1}_{nn}[t]$
Beamformer
$\frac{\boldsymbol{\Phi}^{-1}_{nn}[t]\mathbf{v}[t]}{\mathbf{v}^{\mathsf{H}}[t]\boldsymbol{\Phi}^{-1}_{nn}[t]\mathbf{v}[t]}$

Figure 6.26: A highly customized processing pipeline composed of an enhancer, covariance estimator, and a beamformer.

enhancers into beamforming pipelines for spatially stationary scenes [63, 64]. These works leverage a two-part procedure where a pre-trained DNN mask-estimation module is inserted into an existing beamforming pipeline and used for spectral estimation. Training modules of a beamforming pipeline in one-step or end-to-end has proved challenging due to the numerical instability of spatial covariance matrix inversion but has led to improved speech recognition performance [80]. Moreover, it is also possible to do away with the beamforming structure entirely and learn DNN-based multi-channel modules that learn to leverage spatial information implicitly [176, 177, 178]. However, some works have found that it is advantageous to incorporate explicit spectral and spatial estimation steps either in the form of explicit masking and beamforming [66, 179] or more abstractly [180].

6.9.2 Evaluation

We evaluate our approach in spatially stationary and dynamic scenes. For comparison, we construct several validation-set-tuned conventional estimators. We also evaluate the test-time modularity of our approach by modifying portions of the processing pipeline without retraining. In all, we find that Meta-AF model with covariance outputs can outperform their conventional counterparts by a significant amount in all metrics, are modular, robust, and computationally efficient.

We created a spatialized version of the DNS challenge dataset [181] using pyroomacoustics [182] for simulations. Each scene lasts 5 seconds, contains 1 to 3 noise sources, 1 speech source, and is randomly mixed at SNR between -5 dB and 5 dB. We simulate rooms with dimensions uniformly distributed between 4 m, and 8 m, and with a T60 between 0.25 s and 0.75 s. Within each room, we simulate a 7 cm diameter circular microphone array with 6 microphones. For the dynamic dataset, we simulate array rotation by computing fixed impulse responses at 250 rotations and perform a time-varying convolution. We generate rotation patterns according to the EasyCom dataset [183]. To prevent leakage we use separate room geometries, rotation patterns, and speech/noise files for training/validation/testing. We generated two separate datasets, one spatially static (no rotations) and one spatially dynamic (with rotations). The train, validation, and test set splits of each dataset contain $20,000$, $5,000$, and $5,000$ scenes respectively.

To evaluate the effect of enhancement quality and isolate analysis of the covariance estimator, we experiment with semi-oracle, learned, and oracle enhancers. For the semi-oracle enhancers, we use an IRM (ideal-ratio mask) computed using either the echoic (Echoic IRM) or anechoic (Anechoic IRM) clean speech. For the learned enhancer, we predict a single-channel real-valued mask using a 3-layer LSTM with a hidden size of either 256 or 512 trained with SI-SDR [161] loss w.r.t the anechoic clean speech. This is a simple but capable model. All masks are applied via $\mathbf{M} \odot \mathbf{x}_{\mathrm{m}}[\tau]$ where $\mathbf{M}$ is the mask and $\odot$ is the element-wise product. For the Oracle enhancer, we set $\hat{\mathbf{s}} = \mathbf{s}$ and $\hat{\mathbf{n}} = \mathbf{x} - \mathbf{s}$. The IRM and oracle enhancers use ground truth knowledge and only serve as comparison points. The LSTM serves as an example of a suitable model whose performance lies between the Echoic and Anechoic IRMs.

For beamforming, we use MVDR with a steering vector based on the estimated relative-transfer function. When running Meta-AF, we use the normalized first column of the target covariance matrix as a steering vector. We do this to reduce computational complexity during training. When testing all other covariance estimators, we use the normalized largest principal component of the target covariance matrix. We denote the extraction of this time-varying steering vector with the operation $\mathcal{P}(\cdot)$. We then compute MVDR weights as (frequency indices dropped for readability),

$$\mathbf{w}[\tau] = \frac{\mathbf{\Phi}_{nn}^{-1}[\tau]\mathbf{v}[\tau]}{\mathbf{v}^{\mathsf{H}}[\tau]\mathbf{\Phi}_{nn}^{-1}[\tau]\mathbf{v}[\tau]}, \tag{6.21}$$

where $\mathbf{v}[\tau] = \mathcal{P}(\mathbf{\Phi}_{ss}[\tau])$.

We experiment with two conventional covariance estimators as well as learned methods. On the conventional side, we experiment with offline Fixed and online Buffer estimates, which we pair with the enhancers discussed above. For the Fixed case, we compute a single non-causal estimate across the entire scene. For the Buffer case, we use a sliding buffer to compute an online time-varying estimate. In each experiment, we tune the buffer size with a grid search over $[5, 50]$ on the validation dataset. We tested exponentially smoothed updates but found they performed worse.

The learned estimators, $g_\theta(\cdots)$, are composed of a 2-layer $D = 128$ dimension hidden size unidirectional LSTM. We stack the real/imaginary components of each STFT bin as input. On the output, we predict the real and imaginary parameters independently using a set of linear layers. When experimenting with frequency information sharing we set $f_\psi(\cdots)$ to be a stack of three convolutional layers each with kernel size 3 and 64 channels. The convolutional layers run across frequency while the LSTMs run on each time/frequency independently effectively performing parameter sharing across all frequencies. This makes the model independent of the STFT frame/hop but constrained to a predetermined number of microphones. We implemented this framework in PyTorch.

Static Scenes: First, we study static scenes and evaluate all methods using the Scale-Invariant Signal to Distortion Ratio (SI-SDR), Short-Time Objective Intelligibility (STOI), and the Perceptual Evaluation of Speech Quality (PESQ) metric. For this initial analysis, we equip all learned estimator models with a semi-oracle Echoic IRM as the enhancer. These results are shown in Table 6.7. The first grouping (rows 0-3) displays the results for a reference microphone and three single-channel models, which do not estimate spatial statistics. The next grouping (rows 4-7) shows results for a variety of conventional estimators paired with a variety of enhancers. We evaluated all pairs of enhancers/estimators and chose to only show the best. The third grouping (rows 8-9) shows a Meta-AF estimator which predicts Rank-1 updates and how incorporating banded dependencies improves performance in all metrics. Based on this, we use learned frequency information sharing in all other Meta-AF models. The final grouping (rows 10-11) demonstrates that relaxing the enforced structure of the estimator and shifting all covariance processing within the learned module improves performance. Both the Cholesky and Arbitrary models outperform the conventional estimators equipped with perfect source separation knowledge. This is remarkable since the learned modules are causal and rely on worse speech/noise estimates.

The model in row 4 corresponds to [63, 64] where a DNN enhancer is combined with a conventional covariance estimator. Our proposed method is compatible with this approach as we demonstrate next where we study spatially dynamic scenes.

#	Model Attributes		Metrics		
	Enhancer	*Estimator*	*SI-SDR*	*STOI*	*PESQ*
0	N/A	N/A	-1.10	.738	1.17
1	Echoic IRM	N/A	-0.06	.813	1.51
2	LSTM (512)	N/A	2.32	.804	1.44
3	Anechoic IRM	N/A	3.28	.950	3.08
4	LSTM (512)	Fixed [63, 64]	3.26	.855	1.43
5	Anechoic IRM	Fixed	3.32	.866	1.50
6	Oracle	Fixed	5.12	.895	1.75
7	Oracle	Buffer	4.78	.898	1.75
8	Echoic IRM	Diag-Meta-AF Rank-1	5.78	.866	1.61
9	Echoic IRM	Banded-Meta-AF Rank-1.	6.57	.876	1.64
10	Echoic IRM	Banded-Meta-AF Cholesky	7.32	.918	**2.26**
11	Echoic IRM	Banded-Meta-AF Arbitrary	**7.60**	**.919**	2.24

Table 6.7: Results in spatially stationary scenes. Models are made of an enhancer and an estimator. The learned estimators in rows 8-11 outperform their fixed counterparts. The highest scores are in boldface.

Dynamic Scenes: Next, we study spatially dynamic scenes. These results are shown in Table 6.8 and can be compared to Table 6.7. The single-channel approaches in rows 0-3 are minimally impacted by spatial changes since they do not perform spatial processing. The next grouping (rows 4-7) shows results for several pairs of non-learned estimators and enhancers. Unsurprisingly, the fixed estimators shown in rows 4 and 5 drop in performance from static to dynamic scenes. This occurs since collected statistics are dependent on array pose. The buffer-based model shown in row 6 maintains performance as it was re-tuned for dynamic scenes. The second to last grouping (rows 7-9) shows Meta-AF estimators with Rank-1, Cholesky, and Arbitrary structure. While the Rank-1 estimates (row 7) do not outperform the models with oracle knowledge (row 6), the Cholesky (row 8) and Arbitrary (row 9) estimates do. However, they drop in performance when compared to their results in static scenes. This performance drop is the largest for the Rank-1 model. We hypothesize that the fully learned state of the Cholesky and Arbitrary models can adapt more rapidly. Though, sometimes additional structure is useful. The Cholesky model scores better in perceptual metrics than the Arbitrary model, indicating that the Cholesky structure may be more faithful to the MVDR objective.

In the final grouping (rows 10-11), we replace the Echoic IRM, a practically unrealizable

#	Model Attributes		Metrics		
	Enhancer	*Estimator*	*SI-SDR*	*STOI*	*PESQ*
0	N/A	N/A	-1.09	.738	1.17
1	Echoic IRM	N/A	-0.04	.812	1.51
2	LSTM (512)	N/A	2.31	.804	1.43
3	Anechoic IRM	N/A	3.34	.950	3.09
4	LSTM (512)	Fixed [63, 64]	1.43	.798	1.31
5	Oracle	Fixed	1.37	.820	1.42
6	Oracle	Buffer	4.04	.889	1.70
7	Echoic IRM	Banded-Meta-AF Rank-1	4.53	.856	1.61
8	Echoic IRM	Banded-Meta-AF Cholesky	6.60	**.916**	**2.26**
9	Echoic IRM	Banded-Meta-AF Arbitrary	6.80	.915	2.19
10	LSTM (256)	Banded-Meta-AF Arbitrary	6.58	.865	1.75
11	LSTM (256)	Banded-Meta-AF ($2 \times H$) Arbitrary	**7.13**	.873	1.84

Table 6.8: Results in spatially dynamic scenes. Pairing learned estimators and enhancers yields the best SI-SDR as shown in row 11.

enhancer, with an LSTM enhancer. In effect, we create a Meta-AF model with multiple learned components and extend [63, 64]. Though, instead of a two-part training procedure we train the enhancer and estimator jointly, in an end-to-end fashion. This model is entirely realizable as it is online and uses no ground truth knowledge. The only other comparable models are in rows 2 and 4. To push performance, we double the estimator state size from 128 to 256. This largest model achieves our best dynamic scene SI-SDR of 7.13 dB. We believe these models could perform better with a more exhaustive hyperparameter search and tuning, as they run to our max epochs limit of 50.

Test Time Adaptation: Our approach is modular which enhances its real-world usability. Since it is part of a pipeline, we can modify various modules at test time. We study modifications in dynamic scenes and show results in Table 6.9. Our initial model shown in row 0 (same as Table 6.8 row 9) is the banded Arbitrary model trained with a 512 window, 256 point hop, and Echoic IRM.

First, we experiment with doubling and halving the window and hop size at test time (rows 1-4). Effectively, we evaluate test-time latency and resolution adaption. Next, we experiment with changing the enhancer at test time. In row 5, the masker is changed from an Echoic IRM to an Anechoic IRM. This is a case of swapping in better enhancers at test time

#	Test-Time Modification	Metrics		
		SI-SDR	*STOI*	*PESQ*
0	No Modification	6.80	.915	2.19
1	Double Window	6.92	.923	2.26
2	Halve Hop	6.39	.910	2.05
3	Double Window and Hop	6.23	.911	2.16
4	Halve Window and Hop	5.11	.886	1.87
5	Anechoic IRM	4.15	.916	2.21
6	Oracle Separation	3.65	.909	2.14
7	Dead Microphone	0.13	.832	1.48

Table 6.9: Results evaluating test-time modularity. All adaptation is tested on the banded Arbitrary model using the Echoic-IRM.

without retraining. However, the enhancer modification can not be too large. In row 6, we provide oracle separation, and all metrics drop. We believe this stems from the model being trained with an IRM-based enhancer. Finally, we study robustness and simulate hardware failures by replacing a random microphone signal with white noise (row 7). As expected, all metrics drop, however, they remain higher than the Echoic-IRM (Table 6.8 row 1) and buffer estimator, which, in our experiments, errored out.

Computational Complexity: The estimators have 550 K parameters and use 2.5 MB of storage. They can run in real-time with a real-time factor of 0.05 on an Nvidia P100 GPU and 0.9 on a 2.4 GHz Xeon CPU using vanilla Python without any tuning. The computational budget is 80 MFLOP per frame, or 2 GFLOP per second.

Conclusion: We evaluated Meta-AF with covariance outputs, a learning-based method for estimating time-varying spatial covariance matrices, with application to joint speech enhancement and dereverberation. We applied these methods to a dataset containing both spatially -stationary and -dynamic scenes. Our empirical results showed that the proposed methods outperform a variety of conventional estimation approaches. Moreover, by incorporating the learned modules into a conventional beamforming pipeline, we can perform real-time inference, and apply test-time adaptation with respect to latency, resolution, and masking modification. The model is also robust to hardware failures. We believe our proposed method could also be a valuable tool for complex spatial processing in other domains. Further, we have empirical evidence of this model working in real-time on devices.

Chapter 7: Future Work And Conclusions

In this thesis, we present a novel approach for the development of adaptive filter (AF) systems that leverages the power of meta-learning and end-to-end learning techniques. The approach is based on the idea of learning control rules for AF tasks directly from data using some form of supervision. This framework enables the development of processing pipelines that surpass expert-designed systems for a variety of tasks. These techniques can scale from unsupervised to fully supervised to accommodate all available data and scale gracefully with available computational resources.

When we review the cumulative results of our approach, we note several interesting observations. First, the performance of our meta-learned AFs is strong and compares favorably to conventional optimizers across all tasks. Second, the performance gain achieved by meta-learned AFs over conventional AFs is larger for tasks that are traditionally more difficult to model by humans including AEC double-talk, AEC path changes, and directional interference cancellation. Third, we found that we could use general configurations of our method for many tasks, which significantly reduced our development time. This suggests that our learned AFs are a viable replacement for human-derived AFs for a variety of audio signal processing tasks and are most valuable for complex AF tasks that typically require more human engineering effort.

When we reflect on how our learned optimizers can achieve this success, we note two core reasons. First, and most obvious, our meta-learned AFs are *data-driven* and trained on a particular class of signals (e.g. speech, directional noise, etc). Thus, Meta-AF is limited by the capacity of our optimizer network and training data and not signal modeling skills. Second, by framing AF development as a learning problem, we effectively distill the knowledge of our loss into the AF loss and corresponding learned update rules, thus enabling us to learn AFs that optimize objectives that would be very difficult (e.g. our frame accumulated) or even impossible to develop manually (e.g. supervised losses, STOI, SI-SDR, KWS, etc).

7.1 FUTURE WORK

The field of meta-learning and meta-learned optimizers is young and has a bright future for signal processing applications. Future research directions include: enhancing training methods, investigating non-linear optimization problems, developing more efficient optimizer architectures, and refining optimizer/meta-loss functions. Particularly promising avenues for future work include identifying better optimization losses and better filter representations for Meta-AF style optimization. Overall, we are optimistic that our Meta-AF approach can benefit from both adaptive filtering advances as well as deep learning progress and will be an exciting research topic.

7.2 CONCLUSION

We introduce a general framework capable of automatically learning adaptive filter update rules through deep learning from scratch. Our approach provides substantial advantages by eliminating the need for extensive manual tuning and development. It is also the first task-agnostic method for learning AF update rules directly from data. Furthermore, our framework boasts the capability to smoothly adapt across a spectrum of scenarios, ranging from those with limitations such as no supervised training data and minimal computational resources to scenarios abundant in labeled data and ample computational power.

To demonstrate the power of our framework, we use a universal configuration trained on different datasets and test it on all four canonical adaptive filtering architectures and five unique tasks, including system identification, acoustic echo cancellation, equalization, dereverberation, GSC-based beamforming, and heart-rate estimation. We also assess downstream tasks, such as keyword spotting, as well as end-to-end systems like joint echo cancellation and noise reduction. Whenever feasible, we conduct testing using real-world datasets.

In all cases, we were able to train high-performing AFs, which surpassed conventional optimizers as well as certain state-of-the-art methods. We are excited about the future of adaptive filtering backed by deep learning and hope this thesis will stimulate further research and rapid progress.

Appendix A: Experiments on Non-Audio Tasks

A.1 PHOTOPLETHYSMOGRAPH HEART RATE ESTIMATION VIA MOTION ARTIFACT REMOVAL

The task of Photoplethysmograph (PPG) Heart Rate Estimation is critical in the field of physiological monitoring and health assessment. PPG is a non-invasive optical technique used to measure changes in blood volume within tissue, primarily at the fingertip or earlobe, which allows for the estimation of heart rate. However, in real-world scenarios, motion artifacts, such as body movements or ambient light interference, can corrupt PPG signals, leading to inaccurate heart rate readings [184]. This task involves developing sophisticated algorithms and signal processing techniques to extract the true heart rate signal from the noisy PPG data, effectively filtering out unwanted artifacts. Accurate heart rate estimation is essential for various applications, from wearable fitness trackers to clinical monitoring, and it plays a pivotal role in ensuring the reliability of health-related data collected through PPG sensors.

In this appendix section, we address the challenging problem of PPG-based heart rate estimation in the presence of motion artifacts. We consider this motion as a reference signal and employ informed interference cancellation techniques to minimize its adverse effects on heart rate measurement accuracy. This is an approach commonly taken when performing manual filter design [185, 186, 187]. To achieve this, we employ a straightforward yet effective full-block OLS filter, which we control using a diagonal single-layer Meta-AF model. This is the same model and training procedure outlined in the toy system identification task. We also implement a basic heart-rate recovery problem using cross-correlation and peak picking. We perform all evaluations on the TROIKA dataset [188]. This dataset contains PPG recordings from a smart ring from a subject performing physical tasks, like running on the treadmill. The dataset also contains ground truth estimated via standard heart rate estimation techniques.

When we adapt this approach to PPG estimation, we observe some performance gains. We randomly select a demo file and display it in Fig. A.1. This shows that Meta-AF improves tracking qualitatively, a surprising finding since Meta-AF was developed for a vastly different task. We observe a modest improvement in accuracy, resulting in an error reduction from 33.76 root-MSE to 31.74 root-MSE. This achievement is particularly noteworthy since PPG estimation involves different sampling rates, relies on diverse sensor modes (including accelerometers and PPG sensors), and represents a distinct task from traditional audio signal

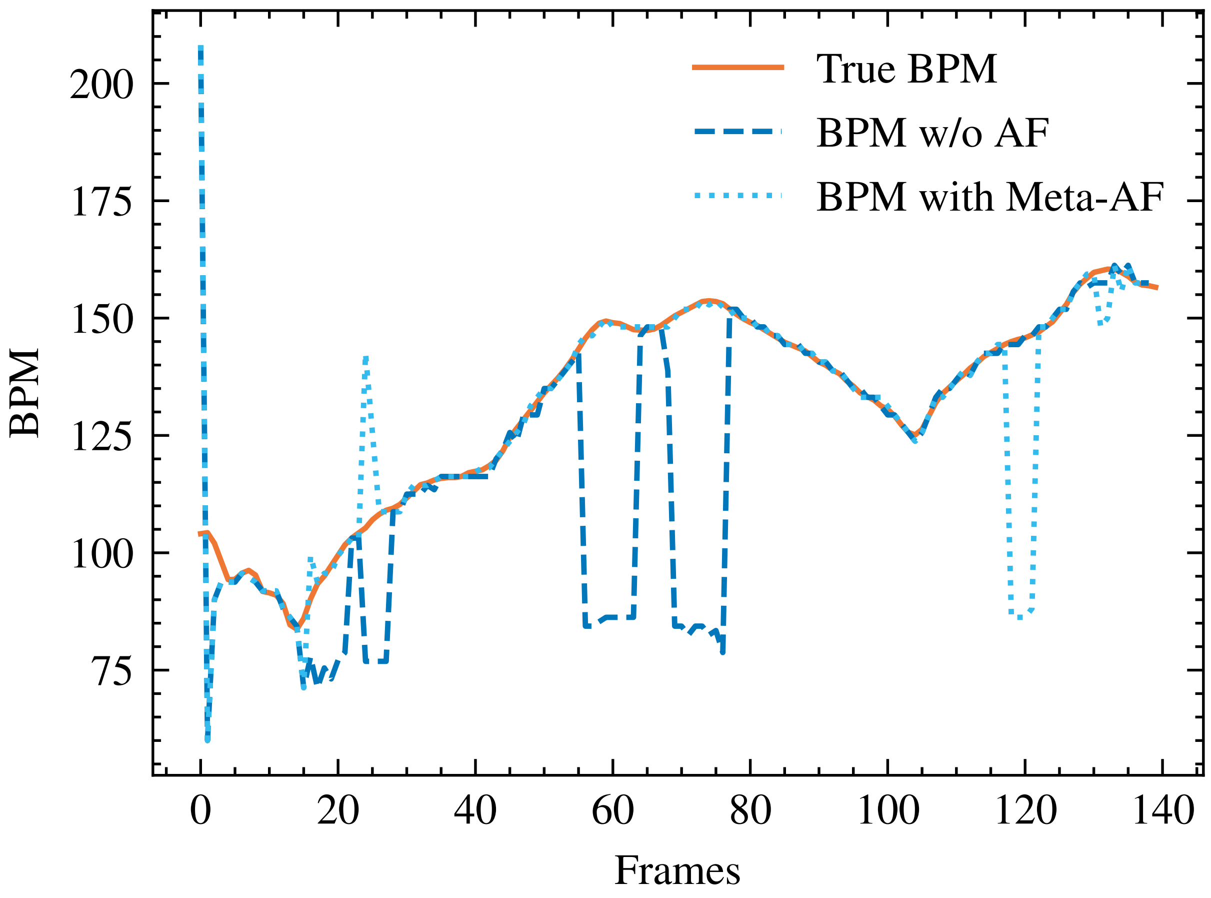


Figure A.1: Results of using Meta-AF to perform motion artifact removal during PPG estimation. Ground truth is shown in solid orange, results without Meta-AF are shown in dark blue dashed, and Meta-AF is light blue dotted. Meta-AF performed well despite never being developed on this task.

processing problems. Nevertheless, our approach seamlessly translates to the realm of PPG-based heart rate estimation, demonstrating its potential for broader applications beyond audio signal processing.

References

[1] B. Widrow and S. D. Stearns, *Adaptive Signal Processing.* Prentice-Hall, 1985.

[2] J. A. Apolinário, J. A. Apolinário, and R. Rautmann, *QRD-RLS adaptive filtering.* Springer, 2009.

[3] V. J. Mathews, "Circuits and systems tutorials: Adaptive polynomial filters," *IEEE SPM*, 1991.

[4] J.-S. Soo and K. K. Pang, "Multidelay block frequency domain adaptive filter," *IEEE TASSP*, 1990.

[5] S. S. Haykin, *Adaptive filter theory.* Pearson, 2008.

[6] L. R. Rabiner, B. Gold, and C. Yuen, *Theory and application of digital signal processing.* Prentice-Hall, 2016.

[7] N. N. Schraudolph, "Local gain adaptation in stochastic gradient descent," in *International Conference On Artificial Neural Networks (ICANN)*, 1999.

[8] S. L. Gay, "An efficient, fast converging adaptive filter for network echo cancellation," in *Asilomar Conference on Signals, Systems, and Computers.* IEEE, 1998.

[9] J. Benesty, T. Gänsler, D. R. Morgan, S. L. Gay, and M. M. Sondhi, *Advances in Network and Acoustic Echo Cancellation.* Springer, 2001.

[10] E. Hänsler and G. Schmidt, *Acoustic echo and noise control: a practical approach.* John Wiley & Sons, 2005.

[11] G. Enzner and P. Vary, "Frequency-domain adaptive Kalman filter for acoustic echo control in hands-free telephones," *Elsevier Signal Processing*, vol. 86, no. 6, 2006.

[12] J.-M. Valin, "On adjusting the learning rate in frequency domain echo cancellation with double-talk," *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, 2007.

[13] S. Malik and G. Enzner, "Model-based vs. traditional frequency-domain adaptive filtering in the presence of continuous double-talk and acoustic echo path variability," in *IEEE International Workshop on Acoustic Signal Enhancement (IWAENC)*, 2008.

[14] S. Malik and G. Enzner, "Online maximum-likelihood learning of time-varying dynamical models in block-frequency-domain," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2010.

[15] F. Kuech, E. Mabande, and G. Enzner, "State-space architecture of the partitioned-block-based acoustic echo controller," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2014.

[16] F. Yang, G. Enzner, and J. Yang, "Frequency-domain adaptive Kalman filter with fast recovery of abrupt echo-path changes," *IEEE Signal Processing Letters (SPL)*, vol. 24, no. 12, 2017.

[17] M. L. Valero and E. A. Habets, "Multi-microphone acoustic echo cancellation using relative echo transfer functions," in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA).* IEEE, 2017.

[18] T. Haubner, A. Brendel, M. Elminshawi, and W. Kellermann, "Noise-robust adaptation control for supervised acoustic system identification exploiting a noise dictionary," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2021.

[19] P. A. Nelson, H. Hamada, S. J. Elliott et al., "Adaptive inverse filters for stereophonic sound reproduction," *IEEE Transactions on Signal Processing (TSP)*, 1992.

[20] P. A. Nelson, F. Orduna-Bustamante, and H. Hamada, "Inverse filter design and equalization zones in multichannel sound reproduction," *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, vol. 3, no. 3, 1995.

[21] M. Bouchard and S. Quednau, "Multichannel recursive-least-square algorithms and fast-transversal-filter algorithms for active noise control and sound reproduction systems," *IEEE Transactions on Speech and Audio Processing (TSAP)*, vol. 8, no. 5, 2000.

[22] N. V. George and G. Panda, "Advances in active noise control: A survey, with emphasis on recent nonlinear techniques," *Elsevier Signal Processing*, vol. 93, no. 2, 2013.

[23] L. Lu, K.-L. Yin, R. C. de Lamare, Z. Zheng, Y. Yu, X. Yang, and B. Chen, "A survey on active noise control in the past decade—part i: Linear systems," *Elsevier Signal Processing*, vol. 183, 2021.

[24] L. Lu, K.-L. Yin, R. C. de Lamare, Z. Zheng, Y. Yu, X. Yang, and B. Chen, "A survey on active noise control in the past decade–part ii: Nonlinear systems," *Elsevier Signal Processing*, vol. 181, 2021.

[25] T. Yoshioka, H. Tachibana, T. Nakatani, and M. Miyoshi, "Adaptive dereverberation of speech signals with speaker-position change detection," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2009.

[26] T. Nakatani, T. Yoshioka, K. Kinoshita, M. Miyoshi, and B.-H. Juang, "Speech dereverberation based on variance-normalized delayed linear prediction," *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, vol. 18, no. 7, 2010.

[27] T. Yoshioka and T. Nakatani, "Generalization of multi-channel linear prediction methods for blind MIMO impulse response shortening," *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, vol. 20, no. 10, 2012.

[28] B. Li, T. N. Sainath, A. Narayanan, J. Caroselli, M. Bacchiani, A. Misra, I. Shafran, H. Sak, G. Pundak, K. K. Chin et al., "Acoustic modeling for google home." in *Interspeech*, 2017.

[29] J. Caroselli, I. Shafran, A. Narayanan, and R. Rose, "Adaptive multichannel dereverberation for automatic speech recognition." in *Interspeech*, 2017.

[30] L. Drude, J. Heymann, C. Boeddeker, and R. Haeb-Umbach, "NARA-WPE: A python package for weighted prediction error dereverberation in numpy and tensorflow for online and offline processing," in *ITG-Symposium on Speech Communication.* VDE, 2018.

[31] J. Wung, A. Jukić, S. Malik, M. Souden, R. Pichevar, J. Atkins, D. Naik, and A. Acero, "Robust multichannel linear prediction for online speech dereverberation using weighted householder least squares lattice adaptive filter," *IEEE Transactions on Singal Processing (TSP)*, vol. 68, 2020.

[32] L. Griffiths and C. Jim, "An alternative approach to linearly constrained adaptive beamforming," *IEEE Transactions on Audio Processing (TAP)*, vol. 30, no. 1, 1982.

[33] O. Hoshuyama, A. Sugiyama, and A. Hirano, "A robust adaptive beamformer for microphone arrays with a blocking matrix using constrained adaptive filters," *IEEE Transactions on Signal Processing (TSP)*, vol. 47, no. 10, 1999.

[34] J. Chen, J. Benesty, and Y. Huang, "A minimum distortion noise reduction algorithm with multiple microphones," *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, vol. 16, no. 3, 2008.

[35] M. Souden, J. Benesty, and S. Affes, "On optimal frequency-domain multichannel linear filtering for noise reduction," *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, vol. 18, no. 2, 2009.

[36] Y. A. Huang and J. Benesty, "A multi-frame approach to the frequency-domain single-channel noise reduction problem," *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, vol. 20, no. 4, 2011.

[37] S. Markovich-Golan, S. Gannot, and I. Cohen, "A sparse blocking matrix for multiple constraints GSC beamformer," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2012.

[38] S. Gannot, E. Vincent, S. Markovich-Golan, and A. Ozerov, "A consolidated perspective on multimicrophone speech enhancement and source separation," *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, vol. 25, no. 4, 2017.

[39] H. Buchner, R. Aichner, and W. Kellermann, "Trinicon: A versatile framework for multichannel blind signal processing," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2004.

[40] T. Kim, H. T. Attias, S.-Y. Lee, and T.-W. Lee, "Blind source separation exploiting higher-order frequency dependencies," *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, 2006.

[41] A. Ozerov and C. Févotte, "Multichannel Nonnegative Matrix Factorization in Convolutive Mixtures for Audio Source Separation," *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, vol. 18, no. 3, pp. 550–563, 2010.

[42] A. Ozerov, C. Févotte, R. Blouet, and J.-L. Durrieu, "Multichannel Nonnegative Tensor Factorization with Structured Constraints for User-Guided Audio Source Separation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2011, pp. 257–260.

[43] H. Sawada, H. Kameoka, S. Araki, and N. Ueda, "Efficient Algorithms for Multichannel Extensions of Itakura-Saito Nonnegative Matrix Factorization," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2012, pp. 261–264.

[44] N. Seichepine, S. Essid, C. Févotte, and O. Cappé, "Soft Nonnegative Matrix Co-Factorization," *IEEE Transactions on Signal Processing (TSP)*, vol. 62, no. 22, pp. 5940–5949, 2014.

[45] A. N. Birkett and R. A. Goubran, "Acoustic echo cancellation using NLMS-neural network structures," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 1995.

[46] A. B. Rabaa and R. Tourki, "Acoustic echo cancellation based on a recurrent neural network and a fast affine projection algorithm," in *IEEE IES*, 1998.

[47] J. Malek and Z. Koldovskỳ, "Hammerstein model-based nonlinear echo cancelation using a cascade of neural network and adaptive linear filter," in *IEEE International Workshop on Acoustic Signal Enhancement (IWAENC)*. IEEE, 2016.

[48] S. Zhang and W. X. Zheng, "Recursive adaptive sparse exponential functional link neural network for nonlinear AEC in impulsive noise environment," *IEEE TNNLS*, 2017.

[49] M. M. Halimeh, C. Huemmer, and W. Kellermann, "A neural network-based nonlinear acoustic echo canceller," *IEEE Signal Processing Letters (SPL)*, 2019.

[50] H. Zhang, K. Tan, and D. Wang, "Deep learning for joint acoustic echo and noise cancellation with nonlinear distortions." in *Interspeech*, 2019.

[51] L. Ma, H. Huang, P. Zhao, and T. Su, "Acoustic echo cancellation by combining adaptive digital filter and recurrent neural network," *arXiv:2005.09237*, 2020.

[52] A. Ivry, I. Cohen, and B. Berdugo, "Nonlinear acoustic echo cancellation with deep learning," in *Interspeech*, 2021.

[53] J.-M. Valin, S. Tenneti, K. Helwani, U. Isik, and A. Krishnaswamy, "Low-complexity, real-time joint neural echo control and speech enhancement based on PercepNet," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021.

[54] Y.-L. Zhou, Q.-Z. Zhang, X.-D. Li, and W.-S. Gan, "Analysis and DSP implementation of an ANC system using a filtered-error neural network," *Journal of Sound and Vibration*, vol. 285, no. 1-2, 2005.

[55] T. Krukowicz, "Active noise control algorithm based on a neural network and nonlinear input-output system identification model," *Archives of Acoustics*, vol. 35, no. 2, 2010.

[56] H. Zhang and D. Wang, "A deep learning approach to active noise control," in *Interspeech*, 2020.

[57] H. Zhang and D. Wang, "Deep anc: A deep learning approach to active noise control," *Neural Networks*, vol. 141, 2021.

[58] K. Kinoshita, M. Delcroix, H. Kwon, T. Mori, and T. Nakatani, "Neural network-based spectrum estimation for online WPE dereverberation." in *Interspeech*, 2017.

[59] J. Heymann, L. Drude, R. Haeb-Umbach, K. Kinoshita, and T. Nakatani, "Frame-online DNN-WPE dereverberation," in *IEEE International Workshop on Acoustic Signal Enhancement (IWAENC)*. IEEE, 2018.

[60] L. Drude, C. Boeddeker, J. Heymann, R. Haeb-Umbach, K. Kinoshita, M. Delcroix, and T. Nakatani, "Integrating neural network based beamforming and weighted prediction error dereverberation." in *Interspeech*, 2018.

[61] Z.-Q. Wang, G. Wichern, and J. Le Roux, "Convolutive prediction for reverberant speech separation," in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. IEEE, 2021.

[62] Z.-Q. Wang, G. Wichern, and J. Le Roux, "Convolutive prediction for monaural speech dereverberation and noisy-reverberant speaker separation," *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, vol. 29, 2021.

[63] H. Erdogan, J. R. Hershey, S. Watanabe, M. I. Mandel, and J. Le Roux, "Improved mvdr beamforming using single-channel mask prediction networks." in *Interspeech*, 2016.

[64] J. Heymann, L. Drude, and R. Haeb-Umbach, "Neural network based spectral mask estimation for acoustic beamforming," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016.

[65] K. Qian, Y. Zhang, S. Chang, X. Yang, D. Florencio, and M. Hasegawa-Johnson, "Deep learning based speech beamforming," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018.

[66] Z. Zhang, Y. Xu, M. Yu, S.-X. Zhang, L. Chen, and D. Yu, "ADL-MVDR: All deep learning MVDR beamformer for target speech separation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021.

[67] W. Herbordtt, S. Nakamura, and W. Kellermann, "Joint optimization of lcmv beamforming and acoustic echo cancellation for automatic speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 3. IEEE, 2005, pp. iii–77.

[68] T. Yoshioka, T. Nakatani, M. Miyoshi, and H. G. Okuno, "Blind separation and dereverberation of speech mixtures by joint optimization," *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, vol. 19, no. 1, pp. 69–84, 2010.

[69] H. Kagami, H. Kameoka, and M. Yukawa, "Joint separation and dereverberation of reverberant mixtures with determined multichannel non-negative matrix factorization," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 31–35.

[70] T. Nakatani and K. Kinoshita, "A unified convolutional beamformer for simultaneous denoising and dereverberation," *IEEE Signal Processing Letters (SPL)*, vol. 26, no. 6, pp. 903–907, 2019.

[71] T. Nakatani, C. Boeddeker, K. Kinoshita, R. Ikeshita, M. Delcroix, and R. Haeb-Umbach, "Jointly optimal denoising, dereverberation, and source separation," *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, vol. 28, pp. 2267–2282, 2020.

[72] S. Hashemgeloogerdi and S. Braun, "Joint beamforming and reverberation cancellation using a constrained kalman filter with multichannel linear prediction," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 481–485.

[73] M. L. Seltzer, B. Raj, and R. M. Stern, "Likelihood-maximizing beamforming for robust hands-free speech recognition," *IEEE Transactions on Speech and Audio Processing (TSAP)*, vol. 12, no. 5, pp. 489–498, 2004.

[74] M. L. Seltzer and R. M. Stern, "Subband likelihood-maximizing beamforming for speech recognition in reverberant environments," *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, vol. 14, no. 6, pp. 2109–2121, 2006.

[75] Y. Liu, P. Zhang, and T. Hain, "Using neural network front-ends on far field multiple microphones based speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 5542–5546.

[76] T. Ochiai, S. Watanabe, T. Hori, J. R. Hershey, and X. Xiao, "Unified architecture for multichannel end-to-end speech recognition with neural beamforming," *IEEE Journal of Selected Topics in Signal Processing (STSP)*, vol. 11, no. 8, pp. 1274–1288, 2017.

[77] K. Zmolikova, M. Delcroix, K. Kinoshita, T. Higuchi, T. Nakatani, and J. Černockỳ, "Optimization of speaker-aware multichannel speech extraction with asr criterion," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2018, pp. 6702–6706.

[78] J. Casebeer, J. Donley, D. Wong, B. Xu, and A. Kumar, "Nice-beam: Neural integrated covariance estimators for time-varying beamformers," *arXiv preprint arXiv:2112.04613*, 2022.

[79] J. Casebeer, V. Vale, U. Isik, J.-M. Valin, R. Giri, and A. Krishnaswamy, "Enhancing into the codec: Noise robust speech coding with vector-quantized autoencoders," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2021.

[80] W. Zhang, C. Boeddeker, S. Watanabe, T. Nakatani, M. Delcroix, K. Kinoshita, T. Ochiai, N. Kamo, R. Haeb-Umbach, and Y. Qian, "End-to-end dereverberation, beamforming, and speech recognition with improved numerical stability and advanced frontend," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2021, pp. 6898–6902.

[81] N. Howard, A. Park, T. Z. Shabestary, A. Gruenstein, and R. Prabhavalkar, "A neural acoustic echo canceller optimized using an automatic speech recognizer and large scale synthetic data," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2021, pp. 7128–7132.

[82] T. Haubner and W. Kellermann, "Deep learning-based joint control of acoustic echo cancellation, beamforming and postfiltering," in *European Signal Processing Conference (EUSIPCO).* IEEE, 2022, pp. 752–756.

[83] W. Zhang, X. Chang, C. Boeddeker, T. Nakatani, S. Watanabe, and Y. Qian, "End-to-end dereverberation, beamforming, and speech recognition in a cocktail party," *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, vol. 30, pp. 3173–3188, 2022.

[84] Y. Masuyama, X. Chang, S. Cornell, S. Watanabe, and N. Ono, "End-to-end integration of speech recognition, dereverberation, beamforming, and self-supervised learning representation," in *IEEE Spoken Language Technology Workshop (SLT).* IEEE, 2023, pp. 260–265.

[85] S. Cornell, T. Balestri, and T. Sénéchal, "Implicit acoustic echo cancellation for keyword spotting and device-directed speech detection," in *IEEE Spoken Language Technology Workshop (SLT).* IEEE, 2023, pp. 1052–1058.

[86] R. Scheibler and M. Togami, “Surrogate source model learning for determined source separation,” in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2021.

[87] T. Haubner, A. Brendel, and W. Kellermann, “End-to-end deep learning-based adaptation control for frequency-domain adaptive system identification,” in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2022.

[88] H. Zhang, S. Kandadai, H. Rao, M. Kim, T. Pruthi, and T. Kristjansson, “Deep adaptive AEC: Hybrid of deep learning and adaptive acoustic echo cancellation,” in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2022.

[89] N. Shlezinger, Y. C. Eldar, and S. P. Boyd, “Model-based deep learning: On the intersection of deep learning and optimization,” *IEEE Access*, vol. 10, pp. 115 384–115 398, 2022.

[90] N. Shlezinger, J. Whang, Y. C. Eldar, and A. G. Dimakis, “Model-based deep learning,” *Proceedings of the IEEE*, 2023.

[91] J. Casebeer, M. Colomb, and P. Smaragdis, “Deep tensor factorization for spatially-aware scene decomposition,” in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA).* IEEE, 2019, pp. 180–184.

[92] J. Casebeer, N. J. Bryan, and P. Smaragdis, “Auto-dsp: Learning to optimize acoustic echo cancellers,” in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2021.

[93] J. Casebeer, N. J. Bryan, and P. Smaragdis, “Meta-AF: Meta-learning for adaptive filters,” *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, 2022.

[94] J. Wu, J. Casebeer, N. J. Bryan, and P. Smaragdis, “Meta-learning for adaptive filters with higher-order frequency dependencies,” in *IEEE International Workshop on Acoustic Signal Enhancement (IWAENC)*, 2022.

[95] R. S. Sutton, “Adapting bias by gradient descent: An incremental version of delta-bar-delta,” in *AAAI Conference on Artificial Intelligence*, 1992.

[96] A. R. Mahmood, R. S. Sutton, T. Degris, and P. M. Pilarski, “Tuning-free step-size adaptation,” in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2012.

[97] J. Schmidhuber, “Learning to control fast-weight memories: An alternative to dynamic recurrent networks,” *Neural Computation*, 1992.

[98] I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le, “Neural optimizer search with reinforcement learning,” in *International Conference on Machine Learning (ICML)*, 2017.

[99] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," in *ICLR*, 2017.

[100] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning (ICML)*, 2017.

[101] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas, "Learning to learn by gradient descent by gradient descent," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2016.

[102] O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. Freitas, and J. Sohl-Dickstein, "Learned optimizers that scale and generalize," in *International Conference on Machine Learning (ICML)*, 2017.

[103] L. Metz, N. Maheswaranathan, J. Nixon, D. Freeman, and J. Sohl-Dickstein, "Understanding and correcting pathologies in the training of learned optimizers," in *International Conference on Machine Learning (ICML)*, 2019.

[104] T. Chen, W. Zhang, Z. Jingyang, S. Chang, S. Liu, L. Amini, and Z. Wang, "Training stronger baselines for learning to optimize," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[105] P. Vicol, L. Metz, and J. Sohl-Dickstein, "Unbiased gradient estimation in unrolled computation graphs with persistent evolution strategies," in *International Conference on Machine Learning*. PMLR, 2021, pp. 10 553–10 563.

[106] J. Harrison, L. Metz, and J. Sohl-Dickstein, "A closer look at learned optimization: Stability, robustness, and inductive biases," *arXiv preprint arXiv:2209.11208*, 2022.

[107] L. Metz, J. Harrison, C. D. Freeman, A. Merchant, L. Beyer, J. Bradbury, N. Agrawal, B. Poole, I. Mordatch, A. Roberts et al., "Velo: Training versatile learned optimizers by scaling up," *arXiv preprint arXiv:2211.09760*, 2022.

[108] J. Casebeer, J. Wu, and P. Smaragdis, "Meta-af echo cancellation for improved keyword spotting," *Under Review*, 2023.

[109] J. Casebeer, N. J. Bryan, and P. Smaragdis, "Scaling up adaptive filter optimizers," *Under Review*, 2023.

[110] S. Venkataramani, J. Casebeer, and P. Smaragdis, "Adaptive front-ends for end-to-end source separation," in *Workshop for Audio Signal Processing, NIPS*, 2017.

[111] S. Venkataramani, J. Casebeer, and P. Smaragdis, "End-to-end source separation with adaptive front-ends," in *Asilomar Conference on Signals, Systems, and Computers*, 2018.

[112] J. Casebeer, B. Luc, and P. Smaragdis, "Multi-view networks for denoising of arbitrary numbers of channels," in *IEEE International Workshop on Acoustic Signal Enhancement (IWAENC)*, 2018.

[113] J. Casebeer, H. Sarker, M. Dhuliawala, N. Fay, M. Pietrowicz, and A. Das, "Verbal protest recognition in children with autism," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

[114] Y.-J. Liu, J. Casebeer, and I. Dokmanić, "Cocktails, but no party: multipath-enabled private audio," in *IEEE International Workshop on Acoustic Signal Enhancement*, 2018.

[115] J. Casebeer, Z. Wang, and P. Smaragdis, "Multi-view networks for multi-channel audio classification," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019.

[116] A. Chaman, Y.-J. Liu, J. Casebeer, and I. Dokmanić, "Multipath-enabled private audio with noise," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019.

[117] M. Pietrowicz, C. Agurto, J. Casebeer, M. Hasegawa-Johnson, K. Karahalios, and G. Cecchi, "Dimensional analysis of laughter in female conversational speech," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2019, pp. 6600–6604.

[118] J. Casebeer, U. Isik, S. Venkataramani, and A. Krishnaswamy, "Efficient trainable front-ends for neural speech enhancement," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2020.

[119] J. Casebeer, J. Kaikaus, and P. Smaragdis, "Communication-cost aware microphone selection for neural speech enhancement with ad-hoc microphone arrays," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2021.

[120] E. Tzinis, J. Casebeer, Z. Wang, and P. Smaragdis, "Separate but together: Unsupervised federated learning for speech enhancement from non-iid data," in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2021.

[121] Z. Wang, J. Casebeer, A. Clemmitt, E. Tzinis, and P. Smaragdis, "Sound event detection with adaptive frequency selection," in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2021.

[122] A. V. Oppenheim and R. W. Schafer, *Digital signal processing.* Prentice-Hall, 1975.

[123] Q. Li, W.-G. Chen, C. He, and H. S. Malvar, "Design of oversampled dft modulated filter banks optimized for acoustic echo cancellation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2009, pp. 197–200.

[124] S. Malik, J. Wung, J. Atkins, and D. Naik, "Double-talk robust multichannel acoustic echo cancellation using least-squares mimo adaptive filtering: Transversal, array, and lattice forms," *IEEE Transactions on Signal Processing (TSP)*, vol. 68, pp. 4887–4902, 2020.

[125] M. Portnoff, “Implementation of the digital phase vocoder using the fast fourier transform,” *IEEE Transactions on Acoustics, Speech, and Signal Processing (TASSP)*, vol. 24, no. 3, pp. 243–248, 1976.

[126] T. N. Sainath, B. Kingsbury, A.-r. Mohamed, and B. Ramabhadran, “Learning filter banks within a deep neural network framework,” in *2013 IEEE workshop on automatic speech recognition and understanding.* IEEE, 2013, pp. 297–302.

[127] G. Hinton, N. Srivastava, and K. Swersky, “Neural networks for machine learning lecture 6a overview of mini-batch gradient descent,” *Cited on*, 2012.

[128] S. T. Alexander and A. L. Ghimikar, “A method for recursive least squares filtering based upon an inverse qr decomposition,” *IEEE Transactions on Signal Processing*, 1993.

[129] P. Strobach, “Low-rank adaptive filters,” *IEEE Transactions on Signal Processing*, 1996.

[130] D. P. Mandic, S. Kanna, and A. G. Constantinides, “On the intrinsic relationship between the least mean square and kalman filters [lecture notes],” *IEEE Signal Processing Magazine*, vol. 32, no. 6, pp. 117–122, 2015.

[131] F. Yang, G. Enzner, and J. Yang, “On the convergence behavior of partitioned-block frequency-domain adaptive filters,” *IEEE Transactions on Signal Procesing (TSP)*, vol. 69, pp. 4906–4920, 2021.

[132] Z. Wang, Y. Na, Z. Liu, B. Tian, and Q. Fu, “Weighted recursive least square filter and neural network based residual echo suppression for the aec-challenge,” in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2021.

[133] “WebRTC acoustic Echo Cancellation v3,” https://webrtc.googlesource.com/src, accessed: 2022-08-10.

[134] D. Yang, F. Jiang, W. Wu, X. Fang, and M. Cao, “Low-complexity acoustic echo cancellation with neural kalman filtering,” in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.

[135] “Acoustic Echo Cancellation Challenge – ICASSP 2021,” https://www.microsoft.com/en-us/research/academic-program/acoustic-echo-cancellation-challenge-icassp-2021/results/, accessed: 2022-08-10.

[136] V. Manohar, S.-J. Chen, Z. Wang, Y. Fujita, S. Watanabe, and S. Khudanpur, “Acoustic modeling for overlapping speech recognition: Jhu chime-5 challenge system,” in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2019.

[137] T. Chen, X. Chen, W. Chen, Z. Wang, H. Heaton, J. Liu, and W. Yin, "Learning to optimize: A primer and a benchmark," *The Journal of Machine Learning Research*, vol. 23, no. 1, pp. 8562–8620, 2022.

[138] X. Wang, S. Yuan, C. Wu, and R. Ge, "Guarantees for tuning the step size using a learning-to-learn approach," in *International Conference on Machine Learning.* PMLR, 2021, pp. 10 981–10 990.

[139] H. W. Heaton, *Learning to Optimize with Guarantees.* University of California, Los Angeles, 2021.

[140] J. Le Roux, N. Ono, and S. Sagayama, "Explicit consistency constraints for STFT spectrograms and their application to phase reconstruction," in *Interspeech*, 2008.

[141] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, and S. Wanderman-Milne, "JAX: composable transformations of Python+ NumPy programs, 2018," *URL http://github. com/google/jax*, 2020.

[142] T. Hennigan, T. Cai, T. Norman, and I. Babuschkin, "Haiku: Sonnet for JAX," 2020. [Online]. Available: http://github.com/deepmind/dm-haiku

[143] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *IEEE*, 1990.

[144] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2014.

[145] M. Wolter and A. Yao, "Complex gated recurrent neural networks," in *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[146] J. Arenas-Garcia, L. A. Azpicueta-Ruiz, M. T. Silva, V. H. Nascimento, and A. H. Sayed, "Combinations of adaptive filters: performance and convergence properties," *IEEE Signal Processing Magazine (SPM)*, vol. 33, no. 1, pp. 120–140, 2015.

[147] D. Kari, A. H. Mirza, F. Khan, H. Ozkan, and S. S. Kozat, "Boosted adaptive filters," *Digital Signal Processing*, vol. 81, pp. 61–78, 2018.

[148] Y. Avargel and I. Cohen, "System identification in the short-time fourier transform domain with crossband filtering," *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, 2007.

[149] M. L. Valero, E. Mabande, and E. A. Habets, "A state-space partitioned-block adaptive filter for echo cancellation using inter-band correlations in the kalman gain computation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015.

[150] B. Widrow and M. E. Hoff, "Adaptive switching circuits," Stanford University, Tech. Rep., 1960.

[151] E. Caballero, K. Gupta, I. Rish, and D. Krueger, "Broken neural scaling laws," in *International Conference on Learning Representations (ICLR)*, 2022.

[152] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark et al., "Training compute-optimal large language models," *arXiv preprint arXiv:2203.15556*, 2022.

[153] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A ConvNet for the 2020s," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[154] M. Kang, J.-Y. Zhu, R. Zhang, J. Park, E. Shechtman, S. Paris, and T. Park, "Scaling up gans for text-to-image synthesis," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[155] T. Haubner, A. Brendel, and W. Kellermann, "End-to-end deep learning-based adaptation control for frequency-domain adaptive system identification," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2022, pp. 766–770.

[156] S. Braun and M. L. Valero, "Task splitting for dnn-based acoustic echo and noise removal," in *IEEE International Workshop on Acoustic Signal Enhancement (IWAENC).* IEEE, 2022, pp. 1–5.

[157] K. Sridhar, R. Cutler, A. Saabas, T. Parnamaa, M. Loide, H. Gamper, S. Braun, R. Aichner, and S. Srinivasan, "ICASSP 2021 acoustic echo cancellation challenge: Datasets, testing framework, and results," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2021.

[158] T. Ko, V. Peddinti, D. Povey, M. L. Seltzer, and S. Khudanpur, "A study on data augmentation of reverberant speech for robust speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2017.

[159] G. Enzner, H. Buchner, A. Favrot, and F. Kuech, "Acoustic echo control," in *Academic press library in signal processing.* Elsevier, 2014, vol. 4.

[160] C. H. Taal, R. C. Hendriks, R. Heusdens, and J. Jensen, "An algorithm for intelligibility prediction of time–frequency weighted noisy speech," *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, vol. 19, no. 7, 2011.

[161] J. Le Roux, S. Wisdom, H. Erdogan, and J. R. Hershey, "SDR–half-baked or well done?" in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2019.

[162] R. Haeb-Umbach, S. Watanabe, T. Nakatani, M. Bacchiani, B. Hoffmeister, M. L. Seltzer, H. Zen, and M. Souden, "Speech processing for digital home assistants: Combining signal processing with deep-learning techniques," *IEEE Signal Processing Magazine (SPM)*, vol. 36, no. 6, pp. 111–124, 2019.

[163] A. Ivry, I. Cohen, and B. Berdugo, "Deep adaptation control for acoustic echo cancellation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 741–745.

[164] G. Shi, P. Aarabi, and H. Jiang, "Phase-based dual-microphone speech enhancement using a prior speech model," *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, vol. 15, no. 1, pp. 109–118, 2006.

[165] X. Zhao and Z. Ou, "Closely coupled array processing and model-based compensation for microphone array speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, vol. 15, no. 3, pp. 1114–1122, 2007.

[166] J. Heymann, L. Drude, C. Boeddeker, P. Hanebrink, and R. Haeb-Umbach, "Beamnet: End-to-end training of a beamformer-supported multi-channel asr system," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 5325–5329.

[167] S. H. Mallidi, R. Maas, K. Goehner, A. Rastrow, S. Matsoukas, and B. Hoffmeister, "Device-directed utterance detection," *Interspeech*, pp. 1225–1228, 2018.

[168] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv:1804.03209*, 2018.

[169] B. Soleimani, H. Schepker, and M. Mirbagheri, "Neural-afc: Learning-based step-size control for adaptive feedback cancellation with closed-loop model training," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.

[170] G. 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)*, vol. 22, no. 8, 2014.

[171] K. Kinoshita, M. Delcroix, S. Gannot, E. A. Habets, R. Haeb-Umbach, W. Kellermann, V. Leutnant, R. Maas, T. Nakatani, B. Raj et al., "A summary of the REVERB challenge: state-of-the-art and remaining challenges in reverberant speech processing research," *EURASIP Journal on Advances in Signal Processing*, vol. 2016, no. 1, 2016.

[172] A. Jukić, T. van Waterschoot, and S. Doclo, "Adaptive speech dereverberation using constrained sparse multichannel linear prediction," *IEEE Signal Processing Letters (SPL)*, vol. 24, no. 1, 2016.

[173] E. Vincent, R. Gribonval, and C. Févotte, "Performance measurement in blind audio source separation," *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, vol. 14, no. 4, 2006.

[174] J. Barker, R. Marxer, E. Vincent, and S. Watanabe, "The third CHiME speech separation and recognition challenge: Dataset, task and baselines," in *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE, 2015.

[175] J. Barker, R. Marxer, E. Vincent, and S. Watanabe, "The third CHiME speech separation and recognition challenge: Analysis and outcomes," *Computer Speech & Language*, 2017.

[176] Y. Luo, C. Han, N. Mesgarani, E. Ceolini, and S.-C. Liu, "Fasnet: Low-latency adaptive beamforming for multi-microphone audio processing," in *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE, 2019, pp. 260–267.

[177] B. Tolooshams, R. Giri, A. H. Song, U. Isik, and A. Krishnaswamy, "Channel-attention dense u-net for multichannel speech enhancement," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 836–840.

[178] Y. Koyama and B. Raj, "Exploring optimal dnn architecture for end-to-end beamformers based on time-frequency references," *arXiv preprint arXiv:2005.12683*, 2020.

[179] Y. Xu, Z. Zhang, M. Yu, S.-X. Zhang, and D. Yu, "Generalized spatio-temporal rnn beamformer for target speech separation," *arXiv preprint arXiv:2101.01280*, 2021.

[180] Z.-Q. Wang, H. Erdogan, S. Wisdom, K. Wilson, D. Raj, S. Watanabe, Z. Chen, and J. R. Hershey, "Sequential multi-frame neural beamforming for speech separation and enhancement," in *IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2021, pp. 905–911.

[181] C. K. Reddy, H. Dubey, K. Koishida, A. Nair, V. Gopal, R. Cutler, S. Braun, H. Gamper, R. Aichner, and S. Srinivasan, "Interspeech 2021 deep noise suppression challenge," in *Interspeech*, 2021.

[182] R. Scheibler, E. Bezzam, and I. Dokmanić, "Pyroomacoustics: A python package for audio room simulation and array processing algorithms," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 351–355.

[183] J. Donley, V. Tourbabin, J.-S. Lee, M. Broyles, H. Jiang, J. Shen, M. Pantic, V. K. Ithapu, and R. Mehra, "Easycom: An augmented reality dataset to support algorithms for easy communication in noisy environments," *arXiv preprint arXiv:2107.04174*, 2021.

[184] D. Castaneda, A. Esparza, M. Ghamari, C. Soltanpur, and H. Nazeran, "A review on wearable photoplethysmography sensors and their potential future applications in health care," *International journal of biosensors & bioelectronics*, vol. 4, no. 4, p. 195, 2018.

[185] H. H. Asada, H.-H. Jiang, and P. Gibbs, "Active noise cancellation using mems accelerometers for motion-tolerant wearable bio-sensors," in *IEEE Engineering in Medicine and Biology Society*, vol. 1. IEEE, 2004, pp. 2157–2160.

[186] M. R. Ram, K. V. Madhav, E. H. Krishna, N. R. Komalla, and K. A. Reddy, "A novel approach for motion artifact reduction in ppg signals based on as-lms adaptive filter," *IEEE Transactions on Instrumentation and Measurement*, vol. 61, no. 5, pp. 1445–1457, 2011.

[187] S. Nabavi and S. Bhadra, "A robust fusion method for motion artifacts reduction in photoplethysmography signal," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 12, pp. 9599–9608, 2020.

[188] Z. Zhang, Z. Pi, and B. Liu, "Troika: A general framework for heart rate monitoring using wrist-type photoplethysmographic signals during intensive physical exercise," *IEEE Transactions on biomedical engineering*, vol. 62, no. 2, pp. 522–531, 2014.