PADERBORN UNIVERSITY

FACULTY OF
COMPUTER SCIENCE,
ELECTRICAL ENGINEERING
AND MATHEMATICS

Extending Speech Separation and Evaluation Measures for Meeting Transcription

From the Faculty of Electrical Engineering, Computer Science and Mathematics of Paderborn University

to obtain the academic degree

Doctor of Engineering (Dr.-Ing.)

approved dissertation
by

M. Sc. Thilo von Neumann

First Reviewer: Prof. Dr. Reinhold Häb-Umbach
Second Reviewer: Priv. Doz. Dr. rer .nat. Ralf Schlüter

Date of the dissertation defense: 24.02.2026

Paderborn 2026

Diss. EIM-E/397

Abstract

Meeting transcription has become an important topic in the field of speech processing. As the task of answering the question who said what and when for long recordings of partially overlapping speech, it encompasses speech transcription (what was said?), diarization (who spoke when?), and, either explicitly or implicitly, speech separation (recovering the speech signal of each speaker). Solving the combination of these problems on arbitrarily long recordings poses challenges that have not been present for the sub-tasks alone. This work specifically discusses how to solve the so-called assignment problem, i.e., finding a plausible assignment or mapping between the ground-truth annotations and the output of a meeting transcription system, or one of its sub-systems. Such an assignment is required when computing a loss function for the training of meeting transcription system or when computing a performance measure. For performance assessment, the Word Error Rate (WER) is extended from the single-utterance evaluation case to the meeting scenario. A temporal constraint is incorporated into the WER such that the obtained alignments stay physically plausible also for long transcriptions. Multiple ways are presented to solve the assignment problem when computing the WER for different system architectures and analysis purposes. After that, it is shown how to solve the assignment problem for the training of a speech separation system by extending the utterance-level Permutation-Invariant Training (uPIT) scheme initially proposed for short fully overlapped speech separation to the meeting scenario. The resulting training technique, named Graph-PIT, enables processing of long recordings without a sliding window approach. Finally, a separation-first and diarization-last pipeline for meeting transcription is built that exploits information about word positions from the speech recognition stage in the diarization. It is shown that this pipeline is competitive with other, often more complex, pipelines.

Zusammenfassung

Die Transkribierung von Gesprächssituationen ist zu einem wichtigen Thema im Bereich der Sprachverarbeitung geworden. Als Aufgabe, die Frage zu beantworten, wer was wann in langen Aufnahmen mit teilweise überlappender Sprache gesagt hat, umfasst sie die Transkription von Sprache (was wurde gesagt?), die Diarisierung (wer hat wann gesprochen?) und, entweder explizit oder implizit, die Sprachseparierung (Wiederherstellung des Sprachsignals jedes Sprechers). Das Lösen der Kombination dieser Unterprobleme bei langen Aufnahmen stellt Herausforderungen dar, die bei den Teilaufgaben allein nicht auftreten. Diese Arbeit befasst sich speziell mit der Lösung des sogenannten Zuordnungsproblems (*Assignment Problem*), d. h. der Suche nach einer plausiblen Zuordnung oder Abbildung zwischen den wahren Annotationen und der Ausgabe eines Besprechungstranskriptionssystems oder eines seiner Teilsysteme. Eine solche Zuordnung ist notwendig sowohl dann, wenn eine Kostenfunktion im Training berechnet werden soll, als auch, wenn die Leistung eines Transkriptionssystems bewertet werden soll. Um Systeme für Gesprächssituationen zu bewerten wird die die Wortfehlerrate (*Word Error Rate, WER*) von der Bewertung einzelner Aussagen auf Besprechungen erweitert. In die Wortfehlerrate wird eine zeitliche Beschränkung integriert, die sicherstellt, dass Wörter, die als korrekt erkannt werden, auch physikalisch plausibel vom selben akustischen Event stammen können. Außerdem werden Verfahren vorgestellt, wie das Zuordnungsproblem während der Berechnung der Wortfehlerrate für verschiedene Systemarchitekturen and Analysezwecke gelöst werden kann. Anschließend wird hergeleitet, wie das Zuordnungsproblem für das Training eines Neuronalen Netzes zur Sprechertrennung gelöst werden kann. Dies geschieht als Erweiterung des häufig verwendeten Aussagenweisen Permutations-invarianten Trainings (*Utterance-level Permutation Invariant Training, uPIT*) auf ganze Meetingaufnahmen. Das daraus resultierende Trainingsschema, Graph-PIT, erlaubt es, lange Aufnahmen zu verarbeiten, ohne auf eine Fensterung zurückgreifen zu müssen. Abschließend wird ein volles System zur Transkription von Gesprächssituationen vorgestellt, das basierend auf der Sprecherseparierung erst Spracherkennung und dann Diarizsierung durchführt. Dabei werden in der Diarisierung Informationen aus der Spracherkennung wiederverwendet, um die zeitliche Auflösung und damit auch die Sprechererkennung zu verbessern. Das vorgestellte System zeigt eine vergleichbare Transkriptionsqualität wie komplexere Systeme aus der Literatur.

Acknowledgements

I would like to thank everyone who supported me throughout the process of writing this thesis. First and foremost, I am deeply grateful to my supervisor, Prof. Dr. Reinhold Häb-Umbach, for his guidance, encouragement and trust. He gave me the freedom to explore new ideas while providing the necessary pressure to transform them into publications. A big thank-you also goes to the research team at the NTT Inc., Communication Science Labs, in particular to Marc Delcroix and Keisuke Kinoshita, for the fruitful collaboration on many publications and for countless insightful discussions. I also would like to thank my colleagues at the Department of Communications Engineering at Paderborn University who created an enjoyable working environment. I especially thank my office colleagues: Christoph Boeddeker, whose ideas and comments often pushed me into the right direction, and Tobias Cord-Landwehr, for the many discussions and coffee breaks we shared. Finally, I owe my deepest gratitude to my wife and family for their constant support during my studies. My two sons, Dante and Tamino, born during my time at the University, brought me joy and the perfect, and sometimes necessary, distraction from my research.

Contents

1 Introduction

Meetings are a central form of human communication used for collaboration and decision making. Automatic meeting transcription — generating a transcript of who said what and when in a recording of a meeting — enables a wide range of applications, from searchable meeting archives to accessibility services and productivity tools.

Recent progress in speech recognition and signal processing has renewed interest in this task. Several community challenges, such as the NOTSOFAR-1 [1] and 7th and 8th Computational Hearing in Multi-source Environments (CHiME) challenges [2, 3], have driven progress in this area. Along these, corresponding datasets have been released, and commercial interest has grown with the rise of meeting transcription devices and services.

Meeting transcription goes beyond classical speech recognition. Unlike the classical single-speaker single-utterance scenarios, meetings involve long recordings, multiple speakers with spontaneous and overlapping speech and complex interactions. These characteristics make meeting transcription significantly harder than the classical tasks like single-utterance Automatic Speech Recognition (ASR), fully overlapped speech recognition or overlap-free diarization, which have traditionally been studied in isolation.

The task can be fundamentally decomposed into answering two sub-questions: Who spoke when (diarization) and what was said (speech recognition). When multiple participants speak simultaneously (i.e., speech overlaps), a third sub-problem arises: recovering the individual speaker signals (speech separation) from the mixture. Each sub-problem has been studied extensively and evaluated with dedicated evaluation metrics, but mostly in isolation and simplified conditions, e.g., fully overlapped speech separation [4–7], single-utterance speech recognition [8–10] or mostly overlap-free diarization [11, 12]. As a result, practical meeting transcription systems often rely on complex pipelines that break up the meeting transcription problem into these well-studied sub-problems in different orders.

Despite active development in the field, no consensus exists on the optimal pipeline design, i.e., in which order the sub-problems should be solved [13]. For example, speech separation can be performed as the first step in the pipeline [14, OC1], resulting in a separation-first pipeline. The conventional approach to this assumes for the separator fully overlapped speech with a fixed number of speakers in each mixture that is then applied to long recordings using a moving-window scheme [14]. This has two drawbacks: Training data with only short and mostly fully overlapped mixtures does not reflect realistic meeting conditions, and classical separation losses are unstable with silent target signals or a high variability in speaker activity. The moving-window approach further creates computational overhead because overlapping windows are required for a consistent signal reconstruction.

Other approaches may solve diarization first [2, 13, 15–17, OC15] or solve the sub-tasks jointly with an end-to-end model [18, 19]. Each strategy, however, comes with strengths and weaknesses, and their evaluation is complicated by the diversity of outputs they produce. A pipeline may, for example, only estimate speaker changes without attributing utterances to speakers.

This diversity complicates fair model evaluation and comparison. While performance measures for the sub-systems are well established, e.g., Signal-to-Distortion Ratio (SDR) for speech separation, Diarization Error Rate (DER) for overlap-free diarization and Word Error Rate (WER) for single-utterance speech recognition, there is no widely accepted way to evaluate meeting transcription as a whole that works for all system architectures. The only widely accepted metric, the Concatenated minimum-Permutation WER (cpWER) [20], requires systems to estimate speaker labels and leaves out the timing component (who spoke *when*?) from the diarization.

This thesis addresses the outlined challenges on two levels: It first provides an evaluation framework that extends the classical WER to the meeting scenario, and then adapts and extends neural speech separation techniques, resulting in a separation-first pipeline for meeting transcription.

For evaluation, the classical WER is extended to long recordings utilizing a temporal constraint in order to improve the sequence alignment and metric interpretability, resulting in a *time-constrained* WER formulation. To handle cases where speaker information is missing or where speaker attribution performance is desired to be excluded from the evaluation, the WER is modified to find the mapping of reference transcripts to estimated hypothesis transcripts that minimizes the WER. The resulting speaker-agnostic metrics, the ORC-WER, Multiple Input Multiple Output WER (MIMO-WER) and Diarization Invariant cpWER (DI-cpWER), enable the evaluation of systems that was previously infeasible. We also propose visualization tools that reveal system errors at the word level, enabling detailed analysis beyond aggregate statistics.

For separation we adopt the SDR loss to remain stable in the presence of silence and imbalanced speaker activity and extend the classical Utterance-level Permutation Invariant Training (uPIT) training scheme for short mixtures to long recordings through the proposed *Graph-PIT* training scheme. The combined modifications improve the separation quality and reduce the computational overhead when processing long recordings.

Based on this framework, a separation-first and diarization-last pipeline is built. By performing separation and ASR before diarization, the pipeline exploits information from ASR to improve diarization. The resulting pipeline is competitive with other larger and more complex pipelines from the literature.

The remainder of this thesis is structured as follows. Chapter 2 introduces the meeting transcription tasks and surveys common computational approaches to its sub-problems. Section 2.8 and Chapter 3 build and analyze the evaluation framework. Specifically, Section 2.8 describes common performance measures for each sub-problem and how they can be applied to the meeting transcription problem. Chapter 3 describes the extensions required to compute a WER in the meeting scenario. Algorithms are designed for efficiently computing the WERs for these situations and the impact of different errors is analyzed in detail. Chapter 4 and Chapter 5 develop training methods for robust speech separation in meetings by first robustifying the training loss function and then introducing Graph-PIT. Chapter 6 integrates these methods into a full separation-first pipeline for meeting transcription and evaluates it in the context of the current literature. Chapter 7 finally concludes with a summary of contributions and future work.

The main contributions of this work are highlighted at the beginning of each chapter, and related own publications are listed in boxes titled “Related own publications”.

2 Meeting Transcription Overview

This chapter introduces the meeting transcription task as an important application for speech processing systems. It first describes the meeting scenario in Section 2.1 and the signal model in Section 2.2. It then describes the meeting transcription task and its sub-tasks in Section 2.3. The assignment problem, i.e., the problem of finding a mapping between ground-truth annotations and an estimated transcript, is discussed in Section 2.4 as a fundamental problem when developing and evaluating meeting transcription systems. It gives an overview of existing datasets and approaches to meeting transcription in Section 2.5, Section 2.6 and Section 2.7. Finally, it introduces existing evaluation metrics for the meeting scenario in Section 2.8.

2.1 The Meeting Scenario

Meetings are a central mode of communication in a professional context. They bring together multiple participants to exchange information and discuss topics. Unlike controlled laboratory speech recordings, meetings are spontaneous, interactive and highly variable. This makes them particularly challenging for an automatic transcription system.

In a typical meeting scenario, as sketched in Fig. 2.1, multiple participants ("speakers", here four, indicated with different colors) are seated around a table while having a discussion. During the discussion, the participants take turns while reacting to each other, which leads to a typically sparse activity for each individual speaker including long pauses between two utterances of the same speaker. Since the interactions are spontaneous, speech overlaps occur, which is indicated by the overlapping utterances of each speaker in Fig. 2.1. Depending on the setup, the conversation could be recorded with a single microphone or a microphone array located on the table, or even multiple distant devices. We here limit ourselves to the single-microphone case. An example single-channel recording is depicted at the bottom of the figure, where different colors represent different speakers.

Overall, the meeting scenario provides a rich but complex environment for automatic transcription. The combination of the acoustic conditions, complex speaking patterns, and long recordings set it apart from other transcription tasks and motivates the development of techniques for robust meeting transcript.

The following assumptions are made about the meeting characteristics to develop transcription systems and evaluation methods for the meeting scenario. They exclude some of the exotic cases that may be observed in real meeting situations, but simplify modelling of the scenario.

Speech activity patterns are sparse. We assume that for each speaker, phases of speaking and silence alternate. The indicated speech activity in the recording of Fig. 2.1, where different colors mark different speakers, shows such a situation: the blue speaker speaks

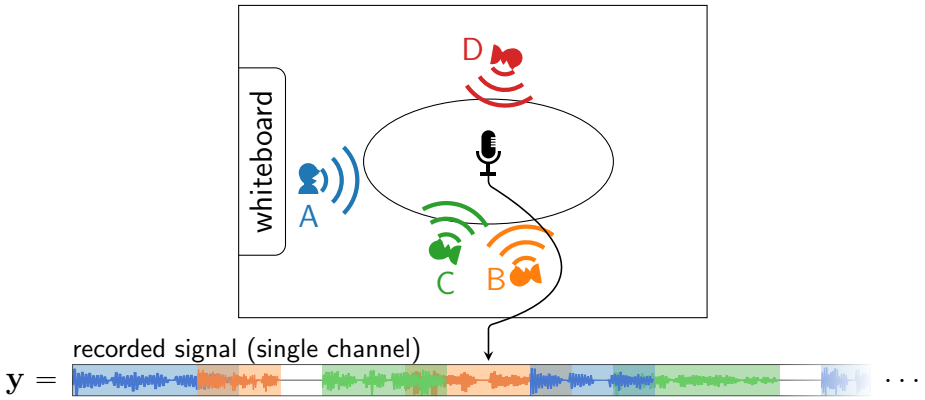


Figure 2.1: An example sketch of a meeting room with speaker and microphone positions. Participants B to D are seated around the table while participant A is standing at the whiteboard. The distances of the speakers to the microphone vary. Speaker D is present but inactive in the depicted extract of the recording at the bottom of the figure.

in the beginning, is silent while the orange and green speakers speak, and then becomes active again. The time that a speaker is active or silent typically varies dramatically between speakers and meetings.

We also assume that most of the time, only a single speaker is active and that speech overlaps are limited to two active speakers at a time. Even though most of the recording is made up of single-speaker speech, speech overlaps impact the transcription quality significantly as formal meetings contain overlap 5 % to 10 % of the time and in informal get-togethers, the overlap can exceed 20 % [OC17]. Simultaneous speech activity of more than two speakers is disregarded because it happens rarely and if it does, these regions often contain no significant speech content.

The acoustic environment is static. We assume static speaker and microphone positions that do not change over the course of a recording. This assumption is mainly taken to simplify the signal model and data simulation process, it is not strictly required for the signal processing methods presented in later chapters. For the acoustic environment, we assume moderate noise levels and reverberation. We assume recording with a single microphone and a distance between the speakers and the microphone in the order of a few tens of cm to a few meters.

Linguistic information can be ignored While linguistic information plays an important role in meeting analysis and tasks such as meeting summarization, we here assume that its role is only minor for meeting transcription. Linguistic context beyond the boundaries of a single utterance are not exploited.

2.2 Signal Model

Following the assumptions above, the scenario can be described as a multi-source single-channel mixture with varying source activity and speech overlaps in the presence of reverberation and background noise. In the following, first the speech activity pattern is modelled,

followed by a description of the acoustic properties of the recorded signals.

Let us first describe the speech activity pattern as a set of utterances $u \in \mathcal{U}$, where each utterance is a continuous region of speech activity of a single speaker. There are in general multiple utterances for each speaker in a meeting and the total number of speakers is denoted by K. The total length of the meeting is denoted by T. Every utterance has a begin time $b_u \in [0, T]$, an end time $e_u \in [b_u, T]$, from the interval between 0 and T so that the end time is not smaller than the begin time, and a speaker label $\ell_u \in \{1, ..., K\}$. The start and end times may also be more detailed than on an utterance-level, e.g., on a word-level if a transcript is provided, in which case they are represented as vectors $\mathbf{b} = [b_{u,1}, b_{u,2}, ...]$ and $\mathbf{e} = [e_{u,1}, e_{u,2}, ...]$. The length of an utterance is the distance between its start and end time, $e_u - b_u$. When a transcript is provided along the utterances, then the timestamps can also be available for each word instead of for each utterance. The temporal order of the utterances is captured by the relation $<_\mathrm{u}$, where $u <_\mathrm{u} u'$ if u begins before u'. It is in general assumed that the utterance indices are sorted by $<_\mathrm{u}$. Two utterances are defined to overlap if they share a common time interval. The overlap function

$$o(u_1, u_2) = \begin{cases} 1 & \text{if } e_{u_1} > b_{u_2} \text{ and } e_{u_2} > b_{u_1} \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

is 1 when the two utterances u_1 and u_2 overlap and 0 otherwise. Among others, it fulfills the following properties:

1. $\ell_{u_1} = \ell_{u_2} \Rightarrow o(u_1, u_2) = 0$. Two utterances of the same speaker cannot overlap.
2. $b_{u_2} < b_{u_1} \wedge b_{u_3} < b_{u_1} \wedge o(u_1, u_2) = 1 \wedge o(u_1, u_3) = 1 \Rightarrow o(u_2, u_3) = 1$. If two utterances overlap with a third utterance and the third utterance begins later than both, then all three utterances overlap.

These properties are required in the remainder of this thesis.

For modelling the speech signals and transcripts, each utterance is associated with a (possibly empty) transcript $\mathbf{t}_u = [t_1, t_2, ...]$ which is represented as a row vector of words $t_1, t_2, ... \in \Sigma$ drawn form the vocabulary Σ. Each utterance further has an associated (clean) speech source speech signal $\mathbf{s}_u \in \mathbb{R}^{e_u - b_u}$, whose length is determined by the utterance duration $e_u - b_u$. By clean speech we mean the signal as it leaves the mouth of the speaker, unaffected by environmental distortions.

All signals are single-channel and modeled as column vectors of samples. To align the utterance signals in the time-frame of the overall meeting, we define a padded version of the source signal

$$\breve{\mathbf{s}}_u = [\mathbf{0}_{b_u}^\mathsf{T}, \mathbf{s}_u^\mathsf{T}, \mathbf{0}_{T-e_u}^\mathsf{T}]^\mathsf{T} \in \mathbb{R}^T, \tag{2.2}$$

where T is the full meeting length in samples, $(\cdot)^\mathsf{T}$ denotes vector transpose and $[\mathbf{a}, \mathbf{b}]$ indicates concatenation vectors.

The observed recording also contains noise $\mathbf{n}$, which encompasses both environmental sounds, e.g., paper rustling or typing, and microphone noise. Assuming a static acoustic environment, the acoustic transfer function between a speaker and the microphone is modeled as a linear time-invariant system, characterized by a convolution with a Room Impulse Response (RIR). Since speaker and microphone positions are fixed, each speaker k is associated with a unique RIR, denoted as $\mathbf{h}_k$.

The overall signal model of a mixture is therefore given by

$$\mathbf{y} = \sum_{u \in \mathcal{U}} \breve{\mathbf{s}}_u * \mathbf{h}_{\ell_u} + \mathbf{n}, \tag{2.3}$$

where $*$ denotes discrete convolution and $\dim(\mathbf{y}) = T$ is the total length of the meeting.

The RIR can be decomposed into two components, an early part $\mathbf{h}^{(\text{early})}$ that consists of the direct path and early reflections, and a late part $\mathbf{h}^{(\text{late})}$ that consists of the diffuse reverberation tail [21]. For convenience, we define the corresponding utterance contribution as

$$\breve{\mathbf{s}}_u^{(\text{early})} = \breve{\mathbf{s}}_u * \mathbf{h}_{\ell_u}^{(\text{early})}, \tag{2.4}$$

$$\breve{\mathbf{s}}_u^{(\text{late})} = \breve{\mathbf{s}}_u * \mathbf{h}_{\ell_u}^{(\text{late})}. \tag{2.5}$$

$$\tag{2.6}$$

With this notation, the mixture signal can be written as

$$\mathbf{y} = \sum_{u \in \mathcal{U}} (\breve{\mathbf{s}}_u^{(\text{early})} + \breve{\mathbf{s}}_u^{(\text{late})}) + \mathbf{n}. \tag{2.7}$$

When training an Neural Network (NN)-based source separator (see Section 2.6) in practice, the late reverberation tail can be regarded as an additional noise source and excluded from the target signal, whereas the direct speech and its early reflections are considered beneficial over using the clean source alone [22]. This decomposition is mainly relevant when selecting which target signal to use for training a separation network. In the remainder of this thesis, the distinction between early and late components will be omitted whenever it is clear from the context or unimportant.

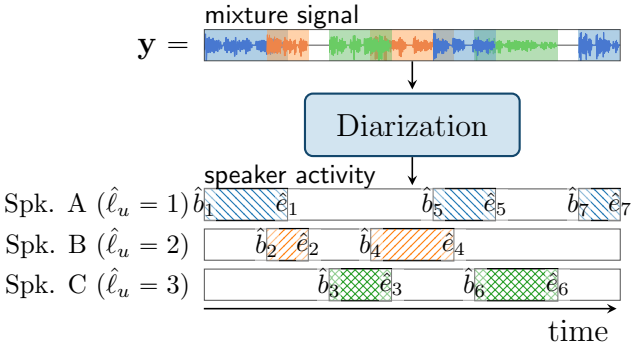
2.2.1 Anechoic Case

Some experiments in this thesis are conducted in an anechoic environment where no reverberation is present. This is modeled by setting all room impulse responses to the causal impulse $\mathbf{h}_k = (1, 0, 0, ...)^{\mathsf{T}}$.

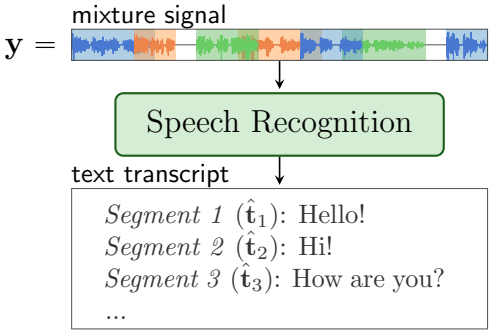
2.3 The Meeting Transcription Task

Given a meeting recording $\mathbf{y}$, the goal of meeting transcription is to produce a transcript that specifies who spoke when and what. Formally, this involves recovering, for each utterance, its transcript $\mathbf{t}_u$, speaker label ℓ_u, begin time b_u and end time e_u, and in some cases also $\mathbf{s}_u$.

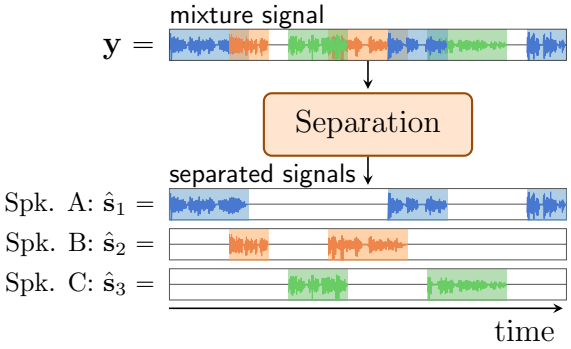
In practice, however, reconstructing the utterances exactly is infeasible because the segmentation is often ambiguous and multiple valid segmentations of a meeting exist. The task is therefore to estimate a set of segments $\hat{\mathcal{U}}$ that resembles the ground-truth utterances $\mathcal{U}$ as closely as possible, while allowing for different segmentation decisions. Each estimated segment $\hat{u} \in \hat{\mathcal{U}}$ again has a transcript $\hat{\mathbf{t}}_{\hat{u}}$, a speaker label $\hat{\ell}_{\hat{u}}$, a begin time $\hat{b}_{\hat{u}}$ and an end time $\hat{e}_{\hat{u}}$ and an estimated speech source signal $\hat{\mathbf{s}}_{\hat{u}}$. This mismatch introduces an assignment problem between system outputs and ground-truth references, discussed further in Section 2.4. For notational simplicity, we will often write u to index both the reference and estimated segments.



(a) Diarization: Speaker activity is estimated. Each box represents a region of speech activity. Time flows from left to right (following the recording) and each row represents a speaker.



(b) Speech Recognition: Words are estimated, no speaker attribution.



(c) Speech Separation: Separated speaker signals are estimated. Systems can have varying output formats. Most systems have a fixed number of output streams.

Figure 2.2: The three sub-problems of meeting transcription.

2.3.1 Sub-problems of Meeting Transcription

Meeting transcription is typically divided into several sub-problems, illustrated in Fig. 2.2 and described below.

Speech separation Speech separation aims to recover the individual clean speech signals $\mathbf{s}_u$ from the mixture signal $\mathbf{y}$. While reconstructing $\mathbf{s}_u$ is not a primary goal of meeting transcription, separation is often used as preprocessing to handle overlapped speech.

NN-based speech separators usually output a fixed number of S signals (streams) $\hat{\mathbf{s}}_1, ..., \hat{\mathbf{s}}_S$, each spanning the full recording. Since the number of utterances $|\mathcal{U}|$ does not generally equal S, multiple utterances may be mapped onto the same stream. If utterances from different speakers are mapped onto each of the streams so that more than S speakers can be separated, it is called Continuous Speech Separation (CSS). An example for this is depicted later in Fig. 2.7. A follow-up diarization step is then required to identify individual utterances, their temporal boundaries and their speaker labels from the separated streams.

Although separation is not strictly necessary if downstream models are overlap-aware, it remains a widely used component and is discussed in more detail in Section 2.6.

Diarization addresses the question "Who spoke when?" (see Fig. 2.2a). It is commonly split into a segmentation and a speaker attribution problem. The segmentation step determines continuous temporal regions containing speech for a single speaker (i.e., estimating the number of utterances $|\mathcal{U}|$ and b_u and e_u for each utterance u). Speaker attribution determines which segment belongs to which speaker (i.e., assigning a speaker label ℓ_u to each utterance u).

Due to ambiguities in the segmentation, the estimated number of utterances can differ from the number of ground-truth utterances. A single utterance can be split into two or multiple utterances merged into one. This is a natural effect because pauses can be interpreted differently. Traditional diarization methods assume non-overlapping speech, which does not hold in meetings. Here, overlap-aware diarization or preprocessing by source-separation is required. The conventional clustering-based diarization pipeline is reviewed in Section 2.7.

Automatic speech recognition [8] addresses the question "What was said?" by estimating transcripts $\mathbf{t}_u$ from speech segments, either from $\hat{\mathbf{s}}_u$ or directly from $\mathbf{y}$ and additional diarization information. Traditional ASR systems assume short and clean single-speaker signals and thus require separation and diarization in advance.

Target-speaker ASR systems use speaker identity information to only transcribe one speaker from a speech mixture. In this thesis, however, ASR systems are treated as black boxes. We rely on pre-trained recognizers without modifications.

2.3.2 Processing Order

Different processing orders of separation, diarization, and ASR have been proposed, with no universally optimal pipeline [13, 14, 20]. Fig. 2.3 illustrates the concepts of the most common strategies.

Each pipeline has system-specific trade-offs. Early stages can provide useful cues for later stages, but errors made early often propagate downstream. Speech separation, for example, improves with diarization information (GSS [23]) while diarization and speech recognition improve when competing speakers are removed with speech separation [24, 25]. It here makes sense putting stronger sub-systems in the target scenario earlier in the pipeline so that all following stages can profit from it, or even to cycle through the same stage multiple times.

We briefly review the major pipeline variants found in the literature.

Diarization-first A diarization-first pipeline, which seems to be the preferred pipeline for processing real-world data[1], performs diarization before separation and speech recognition. Speech separation in this pipeline can use estimated speaker activity regions (e.g., Guided Source Separation (GSS) [23]) or speaker identity information (e.g., SpeakerBeam [32], VoiceFilter [33]) from diarization to improve performance, as done in the CHiME 6 Pipeline [20]. The separation and recognition stages can here also be merged into a target-speaker speech recognizer, resulting in the target-speaker ASR pipeline [26] that extracts only the speech of a requested speaker, represented by a speaker embedding vector obtained from diarization. Such pipelines are limited by the capability of the diarization system, which are often limited to a maximum number of speakers or break with large amounts of overlaps.

[1] Most of the submissions to the CHiME-7 challenge employed an initial diarization phase [2, 15–17, OC15, 27–31]

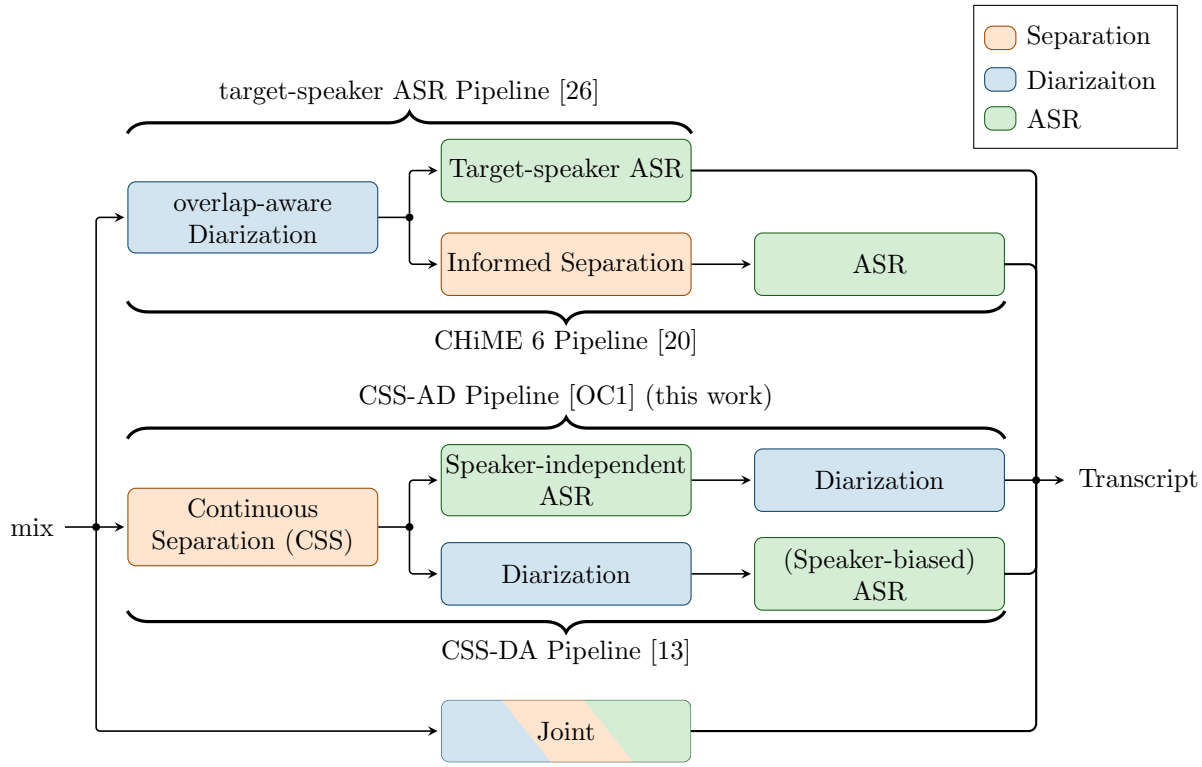


Figure 2.3: An overview of pipeline architectures found in the literature. *(Inspired by [13] Fig. 1)*

Separation-first A separation-first pipeline performs speech separation first to simplify the follow-up diarization and speech recognition stages. The CSS-DA (Separation → Diarization → ASR) [14, 34] or CSS-AD (Separation → ASR → Diarization) [OC1] pipelines are examples for this. In the CSS-DA pipeline, diarization is performed on overlap-free signals followed by speaker-dependent or speaker-independent ASR. Speaker-dependent ASR here differs slightly from target-speaker ASR since it receives already separated speech, so speaker information is only used to boost separation quality, it is not necessarily required. The CSS-DA pipeline was first mentioned in [13] and implemented in [OC1]. It applies a speaker-independent ASR system to the continuously separated streams followed by diarization as the last step, where the diarization can use information from the ASR. Such a pipeline has been implemented in [OC1] and is described in detail in Chapter 6. When separation is performed first, the pipelines are typically either limited by a maximum number of speakers in the total recording, within a limited temporal context or active at the same point in time.

Multi-pass pipelines There are also approaches that go beyond the pipelines displayed in Fig. 2.3 and solve the same sub-problem multiple times to refine it. For example, TS-VAD employs multiple diarization stages [35], the best TS-SEP configuration employs diarization followed by joint separation and diarization and then performs separation and diarization again [36]. Even more complex pipelines exist that circle multiple times through different diarization and separation methods [27].

Joint models The discussed ambiguity in the order, and the different limitations that are apparent in each pipeline, indicate that jointly solving all sub-problems may be a better

approach. There are attempts to directly train an ASR system on long recordings containing multiple speakers, e.g., with Serialized Output Training (SOT), to output speaker change tokens along with the transcript, either on a single stream [37] or on multiple streams [38]. Since ASR systems are not necessarily bound by the temporal properties of the physical signals, they may output transcripts grouped by speaker (one after another) or in temporal order. These ASR systems also provide rough timing information along with the predicted words, but usually only predict speaker changes and no speaker identities, so that the diarization is only partially solved. As will be shown in Section 6.4.4, their performance is not on par with modular pipelines yet.

2.3.3 Difficulties and Difference to Other Tasks

Meeting transcription differs from traditional single-speaker ASR, utterance-level separation and overlap-free diarization in several key aspects:

- *Long recordings:* Meetings span tens of minutes to hours which requires systems that can handle long recordings and scale efficiently in time and memory demands. Often, block-wise processing is used to achieve this.
- *Multiple interacting speakers:* The interaction between speakers creates more complex overlap patterns, regions with varying numbers of speakers and overlap ratios. The total number of speakers in a recording is often unknown, although it is assumed to be known in this work.
- In realistic meetings, *data is imbalanced* even within a single recording. It is common to have regions with different properties, e.g., regarding the number of speakers, overlap ratio, noise level, etc., in a single recording. A typical structure for a meeting is a single participant giving a presentation followed by a discussion by the whole group. There can also be speakers that only utter a few sentences during the meeting while others are active for long amounts of time. The meeting transcription system must be able to adopt to these different regions and imbalanced data automatically.
- Realistic meetings contain *spontaneous speech*, which often contains *filler words, interruptions* and *speech fragments*, and also *laughter*. Compared to other scenarios, the speech is less well-articulated, sometimes interrupted and more difficult to recognize.

A closely related but even harder problem is the *cocktail party scenario* [39, 40], where multiple conversations occur simultaneously in a noisy, unconstrained environment. Unlike the relatively structured meeting setting, cocktail parties feature more speakers, moving participants, and significantly more challenging acoustic conditions. Recordings of actual cocktail parties are rare, with CHiME-6 being a notable example described in more detail in Section 2.5. The cocktail party scenario is here regarded as future work since it introduces much more sub-problems, multi-array multi-channel processing and handling of speakers moving far distances.

2.4 The Assignment Problem

As discussed in Section 2.3, a meeting transcription system, or its sub-systems, often do not directly reconstruct the utterances $\mathbf{s}_u$ from the mixture $\mathbf{y}$, but an arbitrary number of streams

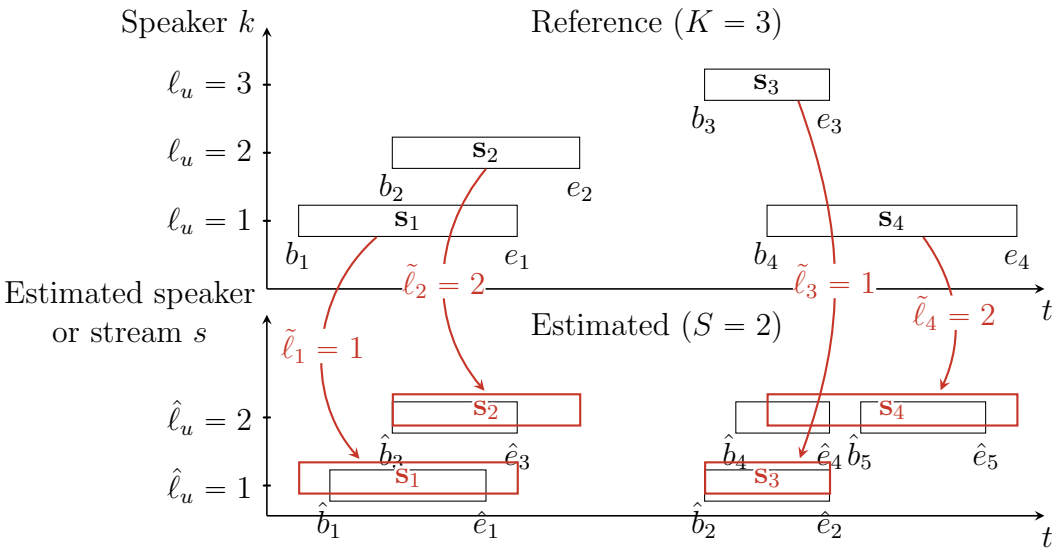


Figure 2.4: An example assignment problem with two estimated streams and three ground-truth speakers. Time flows from left to right, boxes represent ground truth utterances or estimated segments and rows represent ground truth speakers or estimated streams. The assignment problem is to find a mapping between the estimated segments and the ground-truth utterances, here visualized with arrows. The assignment can either be done from the reference to the estimated segments (as shown) or from the estimated segments to the reference utterances.

$\hat{\mathbf{s}}_{\hat{u}}$ or segments that in general do not match the ground-truth utterance boundaries. Speech separation systems often output a fixed number of streams S that each contain multiple utterances from either a single or multiple speakers. Follow-up systems often follow the same structure. For both training and evaluation, thus a mapping must be found between the estimated streams and the ground-truth utterances to compute a loss or a metric. This mapping should be in some way optimal, i.e., regarding the timestamps (as indicated in Fig. 2.4), the word content of the utterances (as done in the WER in Chapter 3) or the speech signal quality (as done for training a source separator in Chapter 5). We call finding this mapping the *assignment problem* since it describes how to assign the reference utterances to the estimated streams (or vice versa). The assignment problem is visualized in Fig. 2.4, where the ground truth utterances and estimated segments are represented as boxes and the optimal assignment is indicated with red arrows and boxes.

The assignment is either represented as a set of modified labels $\{\ell'_u\}$ that describe which utterance is assigned to which estimated stream or as an assignment matrix

$$\mathbf{C} = \begin{bmatrix} \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_{|\mathcal{U}|} \end{bmatrix} = \begin{bmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,S} \\ c_{2,1} & c_{2,2} & \dots & c_{2,S} \\ \vdots & \vdots & \ddots & \vdots \\ c_{|\mathcal{U}|,1} & c_{|\mathcal{U}|,2} & \dots & c_{|\mathcal{U}|,S} \end{bmatrix} \in \{0,1\}^{|\mathcal{U}| \times S}, \tag{2.8}$$

where every row $\mathbf{c}_u = [c_{u,1}, ..., c_{u,S}] = \mathbf{e}^{(S)}_{\ell'_u}$ of $\mathbf{C}$ is a one-hot vector of length S that indicates which estimated stream the u-th utterance is assigned to. The elements of the matrix $c_{u,s}$ are 1 if the u-th utterance is assigned to the s-th stream and 0 otherwise.

The approach to solving the assignment problem depends on the application; solving the

problem based on the word content is different from solving it based on the speech signal. In Chapter 3, different approaches for solving the assignment problem are discussed for computing the WER metric for meeting transcription systems. Here, different approaches arise from the different output formats of a meeting transcription system, i.e., whether estimated utterances are grouped by speaker or not and whether speaker attribution errors should be penalized. In Chapter 5 it is discussed how to solve the assignment problem for training a source separator, while respecting the overlap graph, i.e., making sure that no two reference utterances overlap in the training target signals.

Sometimes, the assignment problem can be simplified significantly by generating data with certain properties, i.e., fully overlapped speech (see uPIT in Section 2.6.1) or specific activity constellations with exactly S possible assignments (see Group-PIT [41], also reviewed briefly in Section 5.5.1).

2.5 Datasets

When developing meeting transcription systems, it is important to have proper datasets for training and evaluation. Several datasets exist that contain recordings of meetings with different qualities and characteristics. We here only consider English datasets at a sample rate of 8 kHz or 16 kHz. In the following, the datasets most relevant for this thesis are described, which include single-speaker datasets as a basis for simulated mixtures and real recordings of conversations.

2.5.1 Simulated Datasets

Since real meeting recordings are scarce and the clean speech signals are not available for real recorded data, simulated datasets are used. Especially speech separation requires the clean speech signals as targets for training. Simulated mixtures are always based on datasets of clean recordings. We here introduce the WSJ and LibriSpeech datasets as the base for simulated fully overlapped and meeting-style mixtures.

2.5.1.1 WSJ

The WSJ dataset [42] contains clean recordings of read speech from articles from the Wall Street Journal. It has been used as a benchmark dataset for single-utterance speech recognition, for example in the Kaldi speech recognition toolkit [9]. This dataset is used with a sample rate of 8 kHz.

2.5.1.2 LibriSpeech

LibriSpeech [43] is a collection of audio snippets from freely available audio books. It contains in total 960 h of training recordings and 10.5 h and 10.7 h of test and development data, respectively. The train set contains in total 2338 different speakers, and the test and development datasets contain 73 speakers each. The recordings are relatively clean and some of them contain significant amounts of silence at the beginning and end. This dataset is used as base for simulated long-form datasets at a sample rate of 16 kHz.

Table 2.1: Overlap ratios of LibriCSS sub-sets in % as total overlap time divided by the length of the recording. Determined based on ground-truth annotations.

subset	0L	0S	OV10	OV20	OV30	OV40
no speech	28.535	1.680	2.028	1.455	1.343	0.885
1 speaker	71.465	98.320	92.795	87.428	81.376	74.367
2 speakers	0.000	0.000	5.177	11.111	17.119	24.162
3 speakers	0.000	0.000	0.000	0.006	0.162	0.586

2.5.1.3 WSJ0-mix

The WSJ0-mix datasets (WSJ0-2mix and WSJ0-3mix) [4] are artificially generated fully overlapped speech mixtures based on the WSJ0 subset of the WSJ dataset. A mixture is generated by sampling two recordings from different speakers from the WSJ0 set, mixing them with a signal-to-interference ratio between −5 dB to 5 dB and cutting the mixture to the length of the shorter recording[2]. This dataset is commonly used as a benchmark for fully overlapped speech separation [4, 5, 44], and also in Section 4.4.1.

2.5.1.4 LibriCSS

The LibriCSS dataset [14] was created by re-recording utterances from the LibriSpeech dataset [43] in a simulated meeting scenario. Eight loudspeakers were placed around a table, one for each meeting participant, and a circular seven-channel microphone array was placed in the center of the table. Speech recordings from the LibriSpeech dataset were played back through the loudspeakers and recorded with the microphone array. The speaker activity was simulated with the CSS task in mind, where speech of many speakers is separated onto a fixed number of output streams S, typically two, so that only a negligible amount of three-speaker overlapped speech was generated.

The dataset contains 10 sessions of 1 hour each. Every session is sub-divided into six mini-sessions named OV40, OV30, OV20, OV10, 0L and 0S of 10 minutes length. These 1h-long sessions are called LibriCSS-1h and the mini-sessions are called LibriCSS-raw in the remainder of this thesis. The mini-sessions were generated to target 40 %, 30 %, 20 %, 10 % of overlap and two sub-sets with no overlap. The 0L sub-set contains long pauses between speaker turns and 0S contains short pauses, where short pauses are in general considered harder, especially for clustering-based diarization systems. Data was generated for utterance-wise evaluation, so the overlap was measured on an utterance-level, i.e., how many samples contain overlaps when cutting the meeting into the ground-truth utterances. This effectively counts overlaps twice compared to measuring the overlap relative to the recording time. The breakdown of the amounts of speech activity per number of speaker in Table 2.1 relative to the meeting duration shows that the total overlap time is only roughly half the target overlap. All datasets are dominated by single-speaker speech. Even though not intended, there is a small amount of three-way overlap.

The LibriCSS-raw recordings are further sub-segmented into the LibriCSS-segments subset,

[2] The datasets also contain a `max` version which is not truncated, but we here only consider the truncated `min` version.

where the recordings are split into shorter recordings of a length of approximately 120 s at points where no speech is active. This sub-set was created to allow the application of systems that are unable to process the longer LibriCSS-raw recordings in one piece.

There is no official split into validation and test sets, but it was recommended to use `session0` as the development set [14]. We adopt this split. Along with the dataset, the CSS task was introduced, which is discussed in Section 2.6.2.

2.5.1.5 Simulated Meetings: MMS-MSG

MMS-MSG[3] [OC2] is a toolkit for simulating speech mixtures and meetings. Along others, it provides functionality to generate speech activity patterns, noise and reverberation from any source datasets for clean speech, background noise and RIRs. The speech activity pattern generation is flexible with configurable probabilities for silence, speech activity and speaker changes, and configurable ranges for the amount of desired overlap. Every aspect of the generation is sampled independently and deterministically, such that the data generation is reproducible and changing one parameter, e.g., the reverberation time, does not affect the other parameters, e.g., the speech activity pattern. It also allows for efficient dynamic mixing, where training speech mixtures are generated on the fly during the training process to increase the data variability in training for better model performance [45].

This simulation tool is used in the remainder of the thesis for simulating any speech mixtures. It especially allows for keeping the training and evaluation data identical except for reverberation in Section 5.4.1 and Section 5.4.2.

2.5.2 Real Recordings

This thesis only considers the Dinner Party Corpus (DiPCo) [46] and CHiME-6 [10] datasets of real recordings that have been part of the 7-th CHiME challenge. Other datasets exist that more closely resemble the meeting scenario, such as AMI [47] and NOTSOFAR-1 [1], but they are not considered in this thesis.

2.5.2.1 DiPCo

The DiPCo dataset [46] contains audio recordings of ten dinner parties, each with four participants. The participants were instructed to have a natural conversation over dinner. The session lengths range from 15 min to 45 min. The DiPCo data is used later for analysis of the WER metrics in Chapter 3.

2.5.2.2 CHiME-6

The CHiME-6 dataset was introduced as part of the 6th CHiME challenge [20]. Similar to DiPCo, it contains audio recordings of twenty dinner parties each with four participants. While this scenario contains more noise and spontaneous speech than a typical meeting, it still shares many properties with the meeting scenario. The challenge contained tracks for speech recognition only and for speech recognition with joint diarization, and was re-used in the 7th CHiME challenge [2] for evaluating meeting transcription systems along with the

[3] https://github.com/fgnt/mms_msg

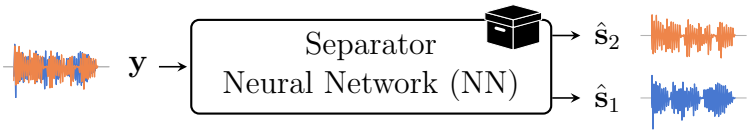


Figure 2.5: Neural-network-based speech separation. A separator takes a mixture signal $\mathbf{y}$ as input and outputs S streams $\hat{\mathbf{s}}_1, ..., \hat{\mathbf{s}}_S$ (here $S = 2$). The network is treated as a black box.

DiPCo dataset and the Mixer 6 dataset[4]. The CHiME-6 data is used for analysis of the presented WER metrics later in Chapter 3.

2.6 Neural-Network-based Speech Separation

Today speech separation is typically based on NNs. The basic procedure for NN-based speech separation is visualized in Fig. 2.5. In this thesis, an NN is treated as a black box that takes a mixture signal $\mathbf{y}$ as input and outputs S streams $\hat{\mathbf{s}}_1, ..., \hat{\mathbf{s}}_S$. Its parameters are optimized in a supervised way with a gradient-based optimizer, such as Stochastic Gradient Descent (SGD) or Adam [48], based on a loss function that compares the estimated signals to the ground-truth signals and adjusts the parameters to reduce the loss. In the remainder of this thesis, different network architectures are used. Their details are outlined in Section A.1. Below, the uPIT training scheme for utterance-wise speech separation and the stitching approach are described.

2.6.1 Utterance-wise Speech Separation

Before discussing speech separation specifically designed for meeting scenarios, this section first introduces the concept of utterance-level, or fully overlapped, speech separation. The goal is to train a NN such that it maps a mixture signal of K speakers to a fixed number of $S \geqslant K$ output streams such that each stream exclusively contains speech of a single speaker (*speaker-exclusive* streams). While multiple approaches exist, e.g., Deep Casa [49], Deep Clustering [4] or uPIT [50], only uPIT has gained wide acceptance in the community as a standard way to train utterance-level speech separation networks.

Only the separation performance is important and not the output order, so the model is allowed to output the speakers in any order. In training, a permutation problem arises when computing a loss between the estimated outputs and the ground truth source signals. It is solved by only performing parameter updates for the permutation of reference signals that best matches the estimated signals. Assuming for now that every source signal $\mathbf{s}_u$ exclusively contains speech of a single speaker and that all signals have the same length, the uPIT loss is [50]:

$$\mathcal{L}^{(\text{uPIT})}(\mathbf{s}_1, ..., \mathbf{s}_K, \hat{\mathbf{s}}_1, ..., \hat{\mathbf{s}}_K) = \sum_{k=1}^{K} \mathcal{L}^{(\text{single})}(\mathbf{s}_{\pi_k^*}, \hat{\mathbf{s}}_k), \tag{2.9}$$

[4] Mixer6 is not used in this thesis and not further detailed due to licensing restrictions.

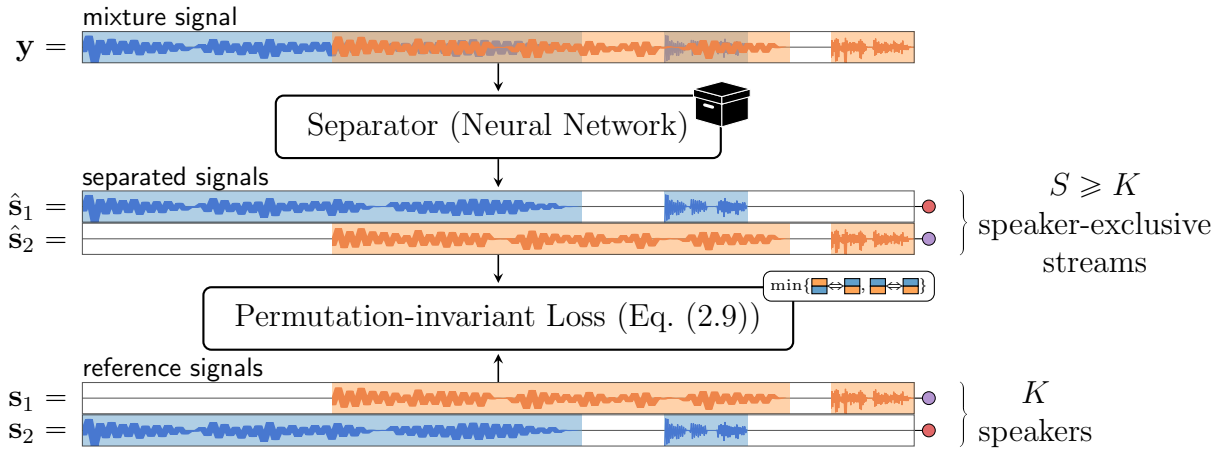


Figure 2.6: Utterance-wise speech separation. Every output stream of the separator should contain the speech of a single speaker exclusively. The number of output streams S must not be smaller than the number of speakers K. The loss computes the minimum over all permutations (here two). *(Modified from [OC3])*

where $\boldsymbol{\pi}^* = [\pi_1^*, ..., \pi_K^*]$ is the permutation vector that minimizes the loss. It is determined as

$$\boldsymbol{\pi}^* = \underset{\boldsymbol{\pi} \in \mathcal{P}(K)}{\arg\min} \sum_{k=1}^{K} \mathcal{L}^{(\text{single})}(\mathbf{s}_{\pi_k}, \hat{\mathbf{s}}_k) \tag{2.10}$$

where $\boldsymbol{\pi} = [\pi_1, ..., \pi_K] \in \{1, ..., K\}^K$, such that $\forall i, j : \pi_i \neq \pi_j$, is a permutation vector taken from the set $\mathcal{P}(K)$ of all $K!$ permutations of length K. $\mathcal{L}^{(\text{single})}$ is an arbitrary signal-level loss function that operates on a single pair of signals. The best permutation $\boldsymbol{\pi}^*$ is a solution of the assignment problem (Section 2.4) where all utterances from the same speaker are assigned to the same output stream.

Since the model output in the beginning of a training is essentially random, the permutations will also be random. Over the course of the training, however, the model will learn to output the signals in a consistent order, and keeping this order fixed can improve the final performance [51]. This order may not correspond to any known properties of the source signals.

Eq. (2.9) can be generalized for signal-level loss functions that allow interactions between the streams [OC4]. Let $\mathcal{L}$ operate on matrices of stacked signals $\mathbf{S} = [\mathbf{s}_1, ..., \mathbf{s}_K] \in \mathbb{R}^{T \times K}$ and $\hat{\mathbf{S}} = [\hat{\mathbf{s}}_1, ..., \hat{\mathbf{s}}_K] \in \mathbb{R}^{T \times K}$, where $[\mathbf{s}_1, ..., \mathbf{s}_K]$ denotes the stacking of K column vectors into a matrix. The permutation $\boldsymbol{\pi}$ is represented as a permutation matrix $\mathbf{P} = [\mathbf{e}_{\pi_1}, ..., \mathbf{e}_{\pi_K}] \in \{0, 1\}^{K \times K}$ which has exactly one 1 in every column (and row), at the position $\pi(i)$, and zeros elsewhere. The matrix $\mathbf{SP}$ is then a permutation of other source signals in $\mathbf{S}$ across the speaker dimension, such that Eq. (2.9) becomes

$$\mathcal{L}^{(\text{uPIT})}(\mathbf{S}, \hat{\mathbf{S}}) = \min_{\mathbf{P} \in \mathcal{P}(K)} \mathcal{L}(\mathbf{SP}, \hat{\mathbf{S}}). \tag{2.11}$$

We again denote with $\mathbf{P}^*$ the permutation matrix that corresponds to the minimum in Eq. (2.11). Finding $\mathbf{P}$ in Eq. (2.11) naively needs factorial time $\mathcal{O}(K!)$ since there are $K!$ permutation matrices for K speakers. A way to speed up finding the optimal permutation is described later in Section 5.2. The permutation matrix $\mathbf{P}$ is a valid assignment matrix if each source signal exclusively contains speech of a single speaker.

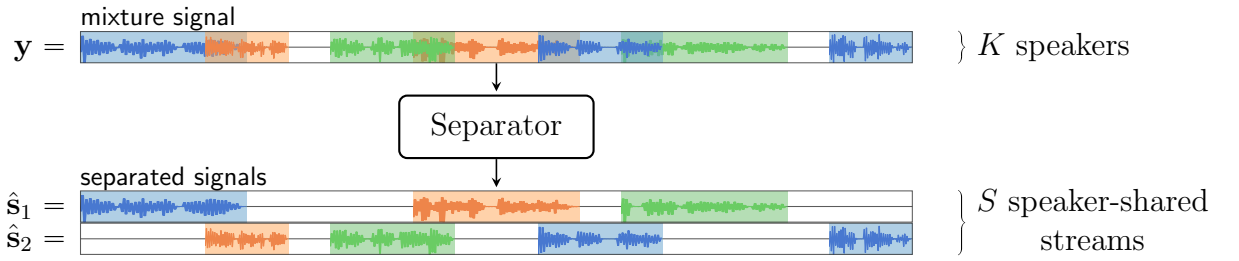


Figure 2.7: Continuous Speech Separation (CSS) with a speaker-shared stream assignment: The meeting recording of arbitrary length and with an arbitrary number of speakers is separated into a small number S (here two) of overlap-free streams. Different colors represent different speakers. Individual utterances from the recording can be placed on any output stream, independent of the speaker label, as long as they do not overlap on any output stream. The number of simultaneously overlapping speakers is limited by S. *(Taken from [OC3])*

2.6.2 Continuous Speech Separation (CSS)

In the meeting scenario, speech separation is often approached with CSS [14, 34]. It separates $\mathbf{y}$ into a set of S overlap-free output streams, in general fewer than there are speakers, where each stream may contain speech of multiple speakers. This general procedure is visualized in Fig. 2.7. The input recording is separated such that there is no speech overlap on any output stream, but the utterances of a single speaker can be placed on different streams and utterances from different speakers can be placed on the same stream. The assignment results in *speaker shared* streams. A follow-up diarization system (Section 2.7) can then extract the speaker labels ℓ_u, begin times b_u and end times e_u for each utterance u and extract $\mathbf{s}_u$ from the speaker-shared streams.

To ensure that CSS-style separation is possible, the number of output streams S must be at least as large as the number of concurrently active speakers $S \geqslant K^{(\text{sim})}$ at any point in the recording. Separation is impossible where this assumption is violated. The number of output streams S is typically chosen smaller than the number of speakers $S < K$ to reduce the computational complexity of the separation system.

2.6.2.1 Applying Utterance-level Separators to Meeting-level Separation With Stitching

An utterance-level separator trained with uPIT can be applied to meeting-level separation with the stitching processing scheme [14, 34]. Assuming that the number of speakers $K^{(\text{seg})}$ in a small temporal window of the recording is small, such that a speaker-exclusive separator can separate the speech within a single window. The separated signals are then aligned across speakers and concatenated into continuous streams.

As visualized in Fig. 2.8, the input mixture is first split into overlapping windows with a future context T_f, a payload window T_c and a history context T_h such that the payloads of adjacent windows are touching, the future context overlaps with the payload of the next window and the history context with the payload of the previous window. Each window is then processed with the separation network, which outputs S signals. Since the windows are processed independently and the set of active speakers can change between adjacent windows, there is no guarantee that the order of speakers is preserved. Thus, the streams in

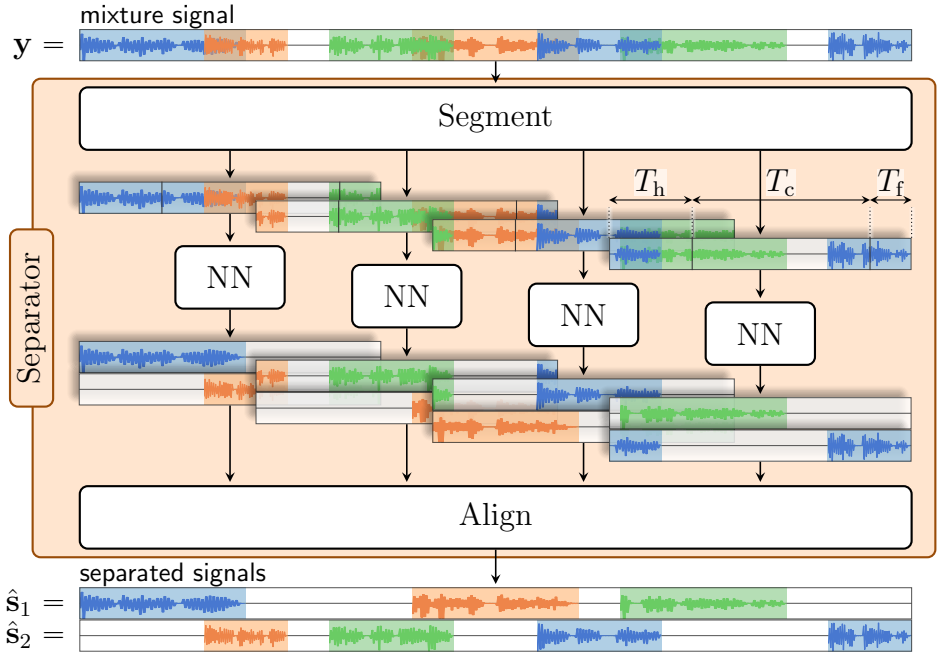


Figure 2.8: The stitching procedure. The input signal is segmented into equally sized overlapping windows. Each window is processed by a separator network. The separated signals are aligned and concatenated, where the history and future contexts are discarded, except for the first and last window. *(Modified from [OC3])*

each window must be reordered in order to obtain continuous and consistent outputs. There are different methods to track speakers across windows, e.g., by using a similarity between the overlapping parts of the windows [14, 34, OC5], using speaker embeddings [52], with an additional tracking network [41], or, for ASR, based on estimated transcripts [53]. In this thesis, the similarity-based stitching is used. After solving the window permutation problem, the future and history contexts are discarded, and the payload windows are concatenated.

The future and history context sizes control the overlap between adjacent windows and the latency of the system. The overlap has to be large enough to obtain reliable permutation estimates. The overall window size must be chosen small enough to ensure that the number of active speakers is less than or equal to the number of output streams of the separator ($K^{(\text{seg})} \leqslant S$), but large enough to ensure a good performance of the separator. This trade-off depends on the data and parameters must be chosen with care according to the specific scenario. In the literature, the window sizes are often chosen small (below 2.5 s [14, 34]) and, to avoid alignment errors, the overlap $(T_\text{h} + T_\text{f})/(T_\text{h} + T_\text{f} + T_\text{c})$ is chosen larger than 50 %.

While this technique is straightforward to apply, it significantly increases the computational complexity of the separation since the windows typically overlap by 50 % or more to get consistent output streams. In Chapter 5, a more efficient approach for CSS is introduced named Graph-PIT which drops the assumption that the number of speakers in a window must not exceed the number of output streams.

2.7 Clustering-based Diarization

While there are many different approaches to diarization, this section only describes the basic clustering-based approach which is used later in Chapter 6. This approach assumes in general no speech overlaps, or a pre-processing with speech separation. It consists in general of three steps [11]: Segmentation, embedding extraction and clustering. Many pipelines contain additional processing steps like front-end preprocessing or post-processing to improve the overall diarization performance, but rely on these three steps as the main diarization procedure.

Segmentation Since no speaker label should be assigned to non-speech regions, the first step is Voice Activity Detection (VAD)[5] to detect continuous regions of speech activity. The complexity of the VAD can range from a simple threshold on the energy to sophisticated algorithms and NNs. After active speech regions have been identified, the signals are split into smaller segments with the goal that each segment contains speech from a single speaker only. The most basic approach are fixed-length overlapping segments, which is the preferred approach for i-vector or d-vector-based embedding extractors. The segmentation can also be based on speaker-change point detection, where the left and right context of a change point candidate are compared to determine whether a speaker change has occurred. This in general leads to larger segments, but a varying segment length, which can be harmful to the speaker embedding extraction step. Errors in the speaker-change point detection also influence the following steps, potentially leading to severe outliers in the clustering step.

Speaker embedding extraction To identify the speakers in each (ideally) speaker-homogeneous segment, a speaker embedding vector is extracted for each segment. The embedding vector is meant to represent the identity of the speaker and to exclude environmental factors. The speaker embedding extractor can be a statistical model (e.g., i-vector [54]) or a neural-network-based model (e.g., d-vector [55]).

Clustering When a speaker embedding has been extracted for each segment, these vectors are clustered to group segments that belong to the same speaker. Here, often k-means [56], spectral clustering [57], or agglomerative hierarchical clustering (AHC) [58] is used. For k-means and spectral clustering, the number of clusters must be known beforehand, which is often assumed.

2.8 Evaluation Metrics

To assess the performance of meeting transcription systems, suitable performance measures have to be selected. A variety of measures has been established for the evaluation for utterance-level systems in the literature, such as the SDR, STOI and PESQ for speech separation systems or the utterance-level WER for speech recognition systems. The evaluation of meeting transcription systems, however, poses challenges not present in the evaluation of utterance-level systems so that these utterance-level metrics cannot be directly applied to

[5] This is often also called Speech Activity Detection (SAD)

meeting scenarios. Specifically, the assignment problem (Section 2.4) must be solved in order to compute a measure for the overall system performance.

As discussed earlier, many meeting transcription systems follow a modular approach where different components are optimized with different objectives and thus can be evaluated independently and should be evaluated with different measures. While it is sufficient for global system ranking to assess the overall performance with a single metric applicable to all systems, such as the WER described in Section 2.8.6 and Chapter 3, it is often insightful to evaluate the individual components with metrics that are specific to the task of the component. For example, the evaluation of a speech separation system can be done with a signal quality metric such as the SDR described in Section 2.8.3, while the evaluation of a diarization sub-system can be done with the DER, described in Section 2.8.5. The evaluation of the individual components can help to identify the bottlenecks in the system and to understand how the different components interact with each other.

Before introducing individual metrics, general properties of performance measures are shortly discussed.

"Foolability" We call a performance measure *foolable* if it can be improved by trivially modifying the hypothesis without improving the actual system performance as perceived by a human. One example for such a trivial modification is splitting estimated segments into smaller segments, which improves a measure if it favors smaller segments, while it objectively has no positive impact on the transcript quality. A foolable metric can often be converted into a non-foolable metric by always applying the fooling technique for all systems. This, however, often leads to a metric that is not very informative anymore and it may incur significant computational costs.

System ranking vs. system analysis Metrics can be used for different purposes: two of them are system ranking and system analysis. For system ranking, it is important that the metric correlates with the system performance and that there is no simple way to improve the metric without improving the actual system performance (the metric should be "unfoolable"). For (detailed) system analysis, on the other hand, it is less important whether the metric can be fooled, but it should reflect specific properties of the system quality with high precision. For example, how frequently does the system miss a speaker or how often does it hallucinate false speakers. Here, metrics are used to pinpoint issues of the system that need improvement for further development and the developer should understand the inner workings of the system and how they are reflected in the metric.

In the following, we will discuss metrics that are used in the context of meeting processing. This section focuses on general remarks regarding metrics for meeting processing, signal quality metrics and speaker diarization quality metrics. Transcript quality metrics are discussed in detail in Chapter 3.

2.8.1 Errors in Meeting Transcription

When solving the meeting transcription tasks, or one of its sub-tasks, the system makes errors that are specific for the task it is intended to solve, i.e., a speech recognizer makes recognition errors, speech separation creates artifacts that impact the signal quality and

diarization systems make timestamp precision or speaker attribution errors. In addition to these sub-task-specific errors, any meeting transcription system is in general subject to the same error types that emerge from the meeting scenario. These error types comprise speech activity errors, speaker attribution errors, and segmentation errors.

Speech activity errors occur when the predicted speech activity does not match the ground-truth speech activity, e.g., a segment is predicted in a region where the ground truth is silent (false-alarm) or nothing is predicted where speech is active (missed speech). These errors are mainly important for the diarization quality. Special cases of false-alarm are leakage (sometimes called *over-separation* [59]), where a segment is duplicated on a different output stream, and segment-length over-estimation, where a segment is estimated longer than it actually is. Segment-length under-estimation is a special case of missed speech.

Speaker attribution errors include any wrongly assigned speaker labels. Special cases are a single wrongly assigned speaker label or a speaker label swap between two segments.

Segmentation errors describe a mismatch between the segmentation, i.e., the grouping of speech activity words (in the final output), in the ground-truth and the estimation. Utterance merge errors are errors where two ground-truth utterances are merged. Utterance split errors are errors where one ground-truth utterance is split into two segments. Segmentation errors alone are typically not reflected in a performance measure because the segmentation is somewhat arbitrary. It is, for example, up to the annotator if a segment is split at a short pause or not. When combined with a speaker attribution error, however, they are penalized by many performance measures.

Classifying errors algorithmically is difficult The errors outlined above are not mutually exclusive and often occur in combination. An utterance split, for example, often occurs together with a speaker confusion and it is impossible to reliably discriminate between a speaker confusion and a combined missing speech and false alarm error. We here thus refrain from classifying individual error types in the evaluation and focus on the overall sub-system performance.

2.8.2 Utterance-wise Evaluation

Conventional performance measures are designed for the utterance-level scenario and therefore do not directly transfer to the meeting scenario. The main difficulty is the assignment problem, i.e., deciding which (parts of the) system output corresponds to which ground-truth utterance. A common way to address this is through *utterance-wise evaluation* [14, OC5]. In this procedure, the meeting recording and the system output (streams or transcripts) are segmented according to the ground-truth utterance boundaries b_u and e_u. The metric is computed for each ground-truth utterance against all system outputs and the best-matching output is selected. The final score is obtained by averaging across utterances.

The segments can either be extracted on the mixture $\mathbf{y}$ before processing, as in evaluations based on LibriCSS [14], or after processing, as in [OC3, OC5]. In this work, we always process the entire meeting recording first and then cut it into utterances after processing, which better reflects a realistic system application.

Utterance-wise evaluation, however, has clear limitations. Any regions in the system output where no ground truth utterance is present are ignored. This mostly neglects poor reconstruction of silence and leakage of speech of one speaker onto multiple output streams.

Despite its shortcomings, utterance-wise evaluation remains valuable. It provides a meaningful impression of the performance in the speech and overlap regions, as it allows computation of well-established metrics that would usually be inapplicable to the meeting scenario.

2.8.3 Signal-to-Distortion Ratio (SDR)

The SDR [60, 61] is a widely used metric to assess the signal quality at the output of an (utterance-level) speech separator. It measures the ratio of the energy of the target signal to the energy of the distortion. The distortion is here the difference between the target signal and the estimated signal. The SDR commonly used for evaluation of speech separation [60] is *convolution-invariant*, i.e., it allows for linear distortions of the source signal. This is motivated by the argument that separation quality should be evaluated independently of reverberation, and reverberation in a static environment can be approximated by a linear filter. In many settings, such as the LibriCSS dataset, only the ground truth signals without reverberation are available while the separation is performed on the reverberant mixture. We name this metric the Convolution-Invariant SDR (CI-SDR)[6] to avoid confusion with the non-convolutional-invariant SDR used as a loss function in Chapter 4. Convolution invariance is achieved by introducing a linear filter $\mathbf{a}$ of a fixed length applied to the source signal before computing the SDR such that the distortions are minimized. For a pair of reference signal $\mathbf{s}$ and estimated signal $\hat{\mathbf{s}}$, neglecting the segment subscripts, the CI-SDR is defined as

$$\text{CI-SDR} = 10 \log_{10} \frac{\|\mathbf{s} * \mathbf{a}\|^2}{\|\mathbf{s} * \mathbf{a} - \hat{\mathbf{s}}\|^2}, \tag{2.12}$$

where

$$\mathbf{a} = \arg\min_{\mathbf{a}'} \|\mathbf{s} * \mathbf{a}' - \hat{\mathbf{s}}\|^2 \tag{2.13}$$

is a least-squares approximation of the RIR that it can also hide linear distortions in $\hat{\mathbf{s}}$. Since the goal is to approximate $\mathbf{s} * \mathbf{a}$ as close as possible with $\hat{\mathbf{s}}$, a larger CI-SDR is better.

It was discussed in [62] that the CI-SDR is *foolable* since the filter $\mathbf{a}$ can compensate for some distortions in the signal. This can be exploited to exclude frequency bands from the evaluation without affecting the CI-SDR much. It is nevertheless a widely accepted metric for speech separation. The foolability is a small price to pay for the ability to evaluate the signal quality in the presence of reverberation.

The CI-SDR is extended to a meeting-level metric later in Section 5.4.1.2.

2.8.4 Attenuation Ratio

To evaluate the impact of false alarms on the signal level, we can measure how much energy is present in the estimated signal where no ground-truth utterance is active. This can be done with the *attenuation ratio* [OC5], defined as

$$\text{attenutation-ratio} = 10 \log_{10} \frac{\left\|\mathbf{y}^{(\text{sil})}\right\|^2}{\left\|\hat{\mathbf{s}}^{(\text{sil})}\right\|^2}, \tag{2.14}$$

[6] It is commonly also referred to as BSSEval-SDR because it was introduced with the BSSEval toolbox [61].We here refrain from this name to avoid being bound to the toolbox name.

where $\mathbf{y}^{(\text{sil})}$ is the mixture signal in the silence regions and $\hat{\mathbf{s}}^{(\text{sil})}$ is the estimated signal in the silence regions. These silent regions are determined after the assignment problem is solved, e.g., by using oracle utterance boundaries and the assignment from an utterance-wise evaluation. The goal for a system is to push $\hat{\mathbf{s}}^{(\text{sil})}$ to $\mathbf{0}$, so that a larger attenuation ratio is better. The attenutation ratio can be used in combination with an utterance-wise measure to evaluate the regions that are ignored by the utterance-wise evaluation.

2.8.5 Diarization Error Rate

The Diarization Error Rate (DER), as described in [63, 64], is used to quantify diarization errors, i.e., the difference between the estimated speaker segmentation and the ground-truth speaker segmentation. It is defined as

$$\text{DER} = \frac{T^{(\text{FA})} + T^{(\text{MD})} + T^{(\text{SC})}}{T^{(\text{SPK})}}, \tag{2.15}$$

where $T^{(\text{FA})}$ is the time of false alarms, $T^{(\text{MD})}$ is the time of missed detections, $T^{(\text{SC})}$ is the time of speaker confusion, and $T^{(\text{SPK})}$ is the total time of the recording.

$T^{(\text{FA})}$ and $T^{(\text{MD})}$ are computed by measuring the time where the number of speakers differs between the ground truth and the estimated segmentation. Any temporal region where the number of speakers is larger than the number of speakers in the ground truth is counted for $T^{(\text{FA})}$, and any region where it is smaller for $T^{(\text{MD})}$. $T^{(\text{SC})}$ is computed by measuring the time where the speaker identity differs between the ground truth and the estimated segmentation, up to a (global) permutation of the speaker labels.

Bounds The DER is bound to a minimum of 0 % (perfect recognition) and maximum of 100 % for non-overlapping speech. Speech overlaps in the reference are counted for $T^{(\text{SPK})}$ so that the DER is still bound by 100 % when speech overlaps appear in the reference. It can, however, exceed 100 % by an arbitrary amount when speech overlaps are present in the hypothesis.

Collar The DER typically employs a *collar*, which describes a region around the boundaries of ground-truth utterances that is excluded from the scoring. A collar of $c = 0.25\,\text{s}$, for example, would exclude all errors the intervals $[b_u - c, b_u + c]$ and $[e_u - c, e_u + c]$ for each ground-truth utterance $u \in \mathcal{U}$ from the scoring. This is done to account for small temporal misalignments between the ground truth and the estimated segmentation. They inevitably appear already due to imprecise annotations and are usually unimportant for follow-up tasks.

DER with speech overlaps The DER was designed for overlap-free scenarios [63] and only looks at speech activity. One effect of this is that speaker swaps in overlapping regions cannot be detected and that the speaker label permutation cannot be reliably estimated for high-overlap scenarios. Additionally, the way that the collar is typically implemented (compare `md_eval_22.pl`[7]) is not well suited for speech overlaps in the reference. Errors around one segment are not only ignored for this segment or speaker, but for the whole recording. For overlapped speech, this means that the collar applied to one segment can mask

[7] https://github.com/nryant/dscore/blob/master/scorelib/md-eval-22.pl

errors that happen in a different (but overlapping) segment. Recent diarization challenges have thus used a collar of 0 for the evaluation of overlapping speech [65, 66]. The DER should therefore be interpreted with care under the presence of speech overlaps.

DER in the absence of (hypothesis) speaker labels The DER cannot be evaluated without speaker labels, so that it is not applicable to CSS-style systems that predict utterance boundaries but no speaker labels. A modification of the DER that does not require speaker labels in the hypothesis was proposed in [OC6] as Voice Activity Error Rate (VAER). As the name suggests, it only looks at the speech activity and completely ignores speaker confusions.

2.8.6 Utterance-wise Word Error Rate

The transcript quality of ASR is commonly assessed with the WER. The (conventional) WER is defined between a pair of a (ground truth) reference transcript $\mathbf{t}$ and a (system output) hypothesis transcript $\hat{\mathbf{t}}$. It counts the number of wrongly recognized words in the hypothesis divided by the total number of words in the reference. A wrongly recognized word can either be deleted, inserted or substituted. The number of these *edit operations* is determined by the Levenshtein distance [67] lev.

The utterance-wise WER is computed for a set of utterances $\mathcal{U}$ as

$$\mathrm{WER}(\mathcal{U}) = \frac{\sum_{u\in\mathcal{U}} \mathrm{lev}(\mathbf{t}_u, \hat{\mathbf{t}}_u)}{\sum_{u\in\mathcal{U}} \dim(\mathbf{t}_u)} = \frac{\sum_{u\in\mathcal{U}} I_u + D_u + S_u}{\sum_{u\in\mathcal{U}} C_u + D_u + S_u}, \tag{2.16}$$

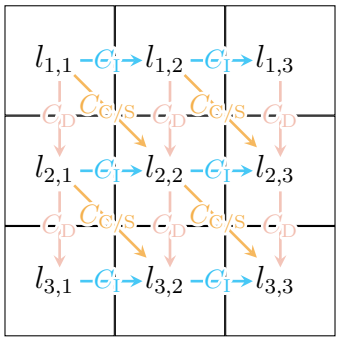
where there is exactly one reference transcript $\mathbf{t}_u$ and one estimated transcript $\hat{\mathbf{t}}_u$ for each utterance, $\dim(\cdot)$ is the number of words in a transcript and lev (see below in Eq. (2.19)) computes the Levenshtein distance between two word sequences. The decomposition into the number of insertions I_u, deletions D_u, substitutions S_u and correct matches C_u is shown in the second half of Eq. (2.16). Note how the overall error rate over a set of transcripts in Eq. (2.16) is not the mean of the error rates, but the sum of all Levenshtein distances across all examples divided by the total number of words across all reference utterances.

This conventional definition of the WER requires exactly one reference and one hypothesis transcript for every utterance. This is usually not given in meeting scenarios due to the assignment problem: the segmentation can differ between reference and hypothesis and the mapping of reference to hypothesis segments can be ambiguous. Approaches to solve this problem are discussed in Section 3.2.

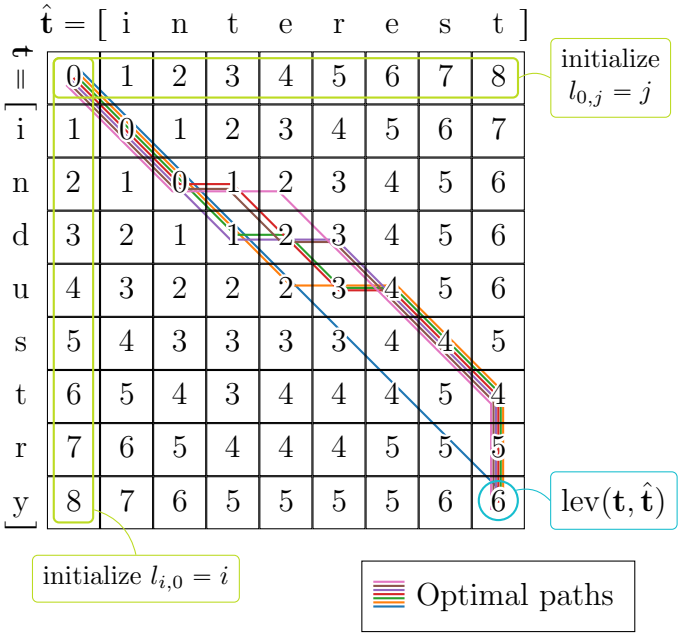
The WER naturally detects (low-level) word-level errors. For system analysis, however, it is often desired to analyze the higher-level errors described in Section 2.8.1. While any isolated high-level error ideally leads to insertions on one stream and deletions on another stream, the WER merges insertions and deletions into substitutions across multiple high-level errors. It is therefore often impossible to discriminate high-level errors from the word-level errors delivered by the WER.

2.8.6.1 Levenshtein Distance

The Levenshtein distance $\mathrm{lev}(\mathbf{t}, \hat{\mathbf{t}})$ [67] between two word sequences $\mathbf{t} = [t_1, ...]$ and $\hat{\mathbf{t}} = [\hat{t}_1, ...]$ is the minimum number of single-word edit operations to transform the word sequence $\mathbf{t}$ into $\hat{\mathbf{t}}$. The utterance index $(\cdot)_u$ is omitted here since the distance can only be computed between



(a) The transitions computed by Eq. (2.19) in an excerpt of the Levenshtein matrix $\mathbf{L}$. Movements to the right are insertions, movements down are deletions and diagonal movements are correct matches or substitutions. The minimum over the received values is computed in each cell.



(b) An example filled Levenshtein matrix $\mathbf{L}$ with alternative optimal match paths highlighted. Characters are treated as words for illustration. The matching is computed between $\mathbf{t}$ = [`i n d u s t r y`] and $\hat{\mathbf{t}}$ = [`i n t e r e s t`].

Figure 2.9: A visualization of the Wagner-Fischer algorithm.

a single pair of transcripts. Edit operations are insertion (a word has to be inserted into $\mathbf{t}$), deletion (a word has to be removed from $\mathbf{t}$) and substitution (a word has to be replaced by another word in $\mathbf{t}$). Treating characters as words for illustration, it takes for example two substitutions and one deletion to turn $\mathbf{t}$ = [`s i t t i n g`] into $\hat{\mathbf{t}}$ = [`k i t t e n`] or five insertions to turn $\mathbf{t}$ = [`a n t`] into $\hat{\mathbf{t}}$ = [`e l e p h a n t`]. In the context of WER, the edit operations are always named relative to the ground-truth transcript (what operations are required to get from the ground-truth reference to the system output?) and describe what erroneous edit operations the system performed while transcribing the speech.

The decomposition of the Levenshtein distance $\mathrm{lev}(\mathbf{t}, \hat{\mathbf{t}})$ into the number of insertions I, deletions D and substitutions S is in general not unique. For example, $\mathbf{t}$ = [`a b`] can be transformed into $\hat{\mathbf{t}}$ = [`b a`] either by two substitutions (`a` → `b` and `b` → `a`) or by one deletion (`a` → ∅) and one insertion (∅ → `b`), where ∅ denotes the empty sequence.

2.8.6.2 Wagner-Fischer Algorithm

The Levenshtein distance can be computed recursively with the Wagner-Fischer algorithm [68], which makes use of the property that $\mathrm{lev}(\mathbf{t}, \hat{\mathbf{t}})$ can be computed from the Levenshtein distances of sub-sequences of $\mathbf{t}$ and $\hat{\mathbf{t}}$ as

$$\mathrm{lev}(\mathbf{t}, \hat{\mathbf{t}}) = \begin{cases} \mathrm{lev}(\mathrm{head}(\mathbf{t}), \hat{\mathbf{t}}) + C_{\mathrm{I}}, \\ \mathrm{lev}(\mathbf{t}, \mathrm{head}(\hat{\mathbf{t}})) + C_{\mathrm{D}}, \\ \mathrm{lev}(\mathrm{head}(\mathbf{t}), \mathrm{head}(\hat{\mathbf{t}})) + C_{\mathrm{C/S}}(\mathrm{tail}(\mathbf{t}), \mathrm{tail}(\hat{\mathbf{t}})). \end{cases} \tag{2.17}$$

The notation $\text{head}(\mathbf{t}) = [t_1, ..., t_{\dim(\mathbf{t})-1}]$ denotes the sub-sequence of $\mathbf{t}$ that consists of all but the last word in $\mathbf{t}$ and $\text{tail}(\mathbf{t}) = t_{\dim(\mathbf{t})}$ is the last word in $\mathbf{t}$, where $\dim(\mathbf{t})$ is here the number of words in $\mathbf{t}$. For the recursion to terminate, we have to define the Levenshtein distance compared to an empty sequence $\varnothing$ as $\text{lev}(\mathbf{t}, \varnothing) = \dim(\mathbf{t})$ and $\text{lev}(\varnothing, \hat{\mathbf{t}}) = \dim(\hat{\mathbf{t}})$ as the length of the non-empty word sequence. The costs for the different edit operations are $C_\text{D} = 1$ for an insertion, $C_\text{I} = 1$ for a deletion, and the cost for a substitution or correct match,

$$C_\text{C/S}(t, \hat{t}) = \begin{cases} C_\text{C} & \text{if } t = \hat{t} \\ C_\text{S} & \text{otherwise,} \end{cases} \tag{2.18}$$

is the substitution cost $C_\text{S} = 1$ if the words in the reference and hypothesis mismatch at this position, and the cost for a correct match $C_\text{C} = 0$ otherwise.[8] We append the costs to the parameter list of lev when they differ from the default, e.g., $\text{lev}(\mathbf{t}, \hat{\mathbf{t}}, C_\text{S} = 2)$.

Eq. (2.17) can be formulated in matrix notation, where the Levenshtein matrix $\mathbf{L} = [l_{ij}] = [\text{lev}(\mathbf{t}_{\leqslant i}, \hat{\mathbf{t}}_{\leqslant j})]$ is recursively filled. The entries at the borders are initialized with $l_{0,j} = j$ and $l_{i,0} = i$ and the remaining entries are computed with

$$l_{i,j} = \min \begin{cases} l_{i-1,j} + C_\text{D} \\ l_{i,j-1} + C_\text{I} \\ l_{i-1,j-1} + C_\text{C/S}(t_i, \hat{t}_j). \end{cases} \tag{2.19}$$

Here, the notation $\mathbf{t}_{\leqslant i} = (t_1, ..., t_i)$ means the first i words of $\mathbf{t}$. The Levenshtein distance is then $\text{lev}(\mathbf{t}, \hat{\mathbf{t}}) = l_{\dim(\mathbf{t}),\dim(\hat{\mathbf{t}})}$.

The update Eq. (2.19) is visualized in Fig. 2.9a, where the transitions and the corresponding costs are indicated between the cells of the Levenshtein matrix. Movements to the right correspond to insertions and add the insertion cost C_I. Movements to the cell below are deletions and add the deletion cost C_D. Diagonal movements are either correct matches or substitutions and add the correct costs $C_\text{C} = 0$ or the substitution cost C_S, depending on the word contents. A full Levenshtein matrix $\mathbf{L}$ is displayed for an example of matching $\mathbf{t} = [\texttt{i n d u s t r y}]$ against $\hat{\mathbf{t}} = [\texttt{i n t e r e s t}]$ in Fig. 2.9b. The first row and first column are initialized with an increasing sequence, representing a matching with only deletions or only insertions, respectively. The remaining matrix is filled with Eq. (2.19), until the cell in the lower right corner is found as the value of the Levenshtein distance.

2.8.6.3 Retrieving the Alignment

An *alignment*, i.e., the sequence of edit operations is retrieved from the Levenshtein matrix $\mathbf{L}$ by tracing back the path of lowest cost, also called optimal path, in the reverse order as it was computed, i.e., starting at $l_{\dim(\mathbf{t}),\dim(\hat{\mathbf{t}})}$ and ending in $l_{0,0}$. When the path goes left in the matrix ($l_{i,j} \rightarrow l_{i-1,j}$), prepend a deletion, when it goes up ($l_{i,j-j} \rightarrow l_{i,j}$), prepend an insertion and if it goes diagonally ($l_{i,j} \rightarrow l_{i-1,j-1}$), insert a correct match or a substitution if the words at this position match or differ. When we denote with S substitution, C correct match, D deletion, and I insertion, the resulting alignment for transforming $\mathbf{t} = [\texttt{s i t t i n g}]$ into

[8] Other choices are possible for the costs, e.g., $C_\text{D} = C_\text{I} = 3$ and $C_\text{S} = 4$ [69], so that substitutions are preferred over insertions and deletions, which may lead to a more plausible matchings. However, Eq. (2.17) only resembles the Levenshtein distance as the absolute number of edit operations for $C_\text{D} = C_\text{I} = C_\text{S} = 1$.

$\hat{\mathbf{t}} = [\underline{\texttt{k}}\ \texttt{i}\ \texttt{t}\ \texttt{t}\ \underline{\texttt{e}}\ \texttt{n}]$ is (S C C C S C D). To turn $\mathbf{t} = [\texttt{a}\ \texttt{n}\ \texttt{t}]$ into $\hat{\mathbf{t}} = [\underline{\texttt{e}\ \texttt{l}\ \texttt{e}\ \texttt{p}\ \texttt{h}}\ \texttt{a}\ \texttt{n}\ \texttt{t}]$ it is [I I I I I C C C].

The optimal path through $\mathbf{L}$ is in general not unique; there can be multiple alignments that result in the same smallest distance. In Fig. 2.9b, seven optimal paths are marked through the Levenshtein matrix that lead to the same total cost, marked with lines with different colors. The diagonal path consist of only substitutions and correct matches. The other paths trade correct matches in the upper right half of $\mathbf{L}$ for insertions and deletions. Since two alignments cannot be meaningfully compared, there is no general notion of a best alignment.

2.8.6.4 Other Algorithms for Computing the Levenshtein Distance

It can be shown that the complexity for computing the Levenshtein distance is in general quadratic [70], here $\mathcal{O}(\dim(\mathbf{t}) \cdot \dim(\hat{\mathbf{t}}))$. Although it is not always necessary to compare all words form the reference to all words from the hypothesis, the required number of comparisons still grows linearly with the length of both sequences.

There are, nevertheless, more efficient algorithms than the Wagner-Fischer algorithm that either reduce the constant factor or improve the average-case complexity in practice. The bit-parallel approach [71] by Myers makes use of the fact that the state update only depends on the difference of the adjacent cells and that these differences are limited to -1, 0, or 1. Two bits per cell of $\mathbf{L}$ are enough to store this difference and an update can be computed with bit-wise operations, which generally leads to a speedup of at least half the CPU's bit width.

Another approach by Ukkonen [72] makes use of the observations that values of cells in the Levenshtein matrix $\mathbf{L}$ only depend on cells with smaller or equal values and that the values along diagonals are monotonically increasing. This insight allows for computing the cells in the Levenshtein distance in a different order (by increasing value rather than row by row or column by column) that more quickly reaches the final result in the lower right corner. By cleverly storing only the indices along the diagonals at which the cost exceeds the currently largest cost in the matrix, the algorithm can be sped up and uses only linear memory. The complexity $\mathcal{O}(\text{lev}(\mathbf{t}, \hat{\mathbf{t}}) \cdot \min(\dim(\mathbf{t}), \dim(\hat{\mathbf{t}})))$ of this algorithm depends on the Levenshtein distance $\text{lev}(\mathbf{t}, \hat{\mathbf{t}})$ between the two sequences and is smaller the closer the two sequences are to each other. In the worst case, however, the complexity is still $\mathcal{O}(\dim(\mathbf{t}) \cdot \dim(\hat{\mathbf{t}}))$ since $\text{lev}(\mathbf{t}, \hat{\mathbf{t}}) = \max(\dim(\mathbf{t}), \dim(\hat{\mathbf{t}}))$ in the worst case. In [73], the algorithm is extended to reduce the search space even further by bounds on the allowed optimal paths through the Levenshtein matrix.

All algorithms presented for computing the Levenshtein distance in the remainder of this thesis are based on the Wagner-Fischer algorithm since it is the easiest to extend. Extensions of the faster bit-parallel algorithms are unknown.

2.9 Summary

This chapter introduced the meeting scenario and the meeting transcription task. The assignment problem was introduced as a fundamental problem that arises when processing or evaluating meetings that form the basis for the discussion about meeting-level WERs in Chapter 3 and separation in Chapter 5. Along with the task description, the datasets used in this thesis were introduced. Classical approaches to the diarization and speech separation

sub-tasks were discussed, while the details of speech recognition were left out since ASR is treated as a pre-trained black-box model in the remainder of this thesis. Finally, evaluation metrics used in the literature for classical utterance-wise and meeting-level evaluation were introduced.

3 Word Error Rates for Meeting Scenarios

The most common metric for evaluating the quality of a (general) speech recognition system is the WER [9, 10, 20, 74, 75]. It was initially defined, as described in Section 2.8.6 for utterance-level speech recognition, for processing short recordings of a single speaker only. When systems become more complex and process more complex data, containing multiple speakers or spanning more than a single sentence, the utterance-level WER definition is no longer applicable due to causing implausible alignments for long recordings and due to the assignment problem.

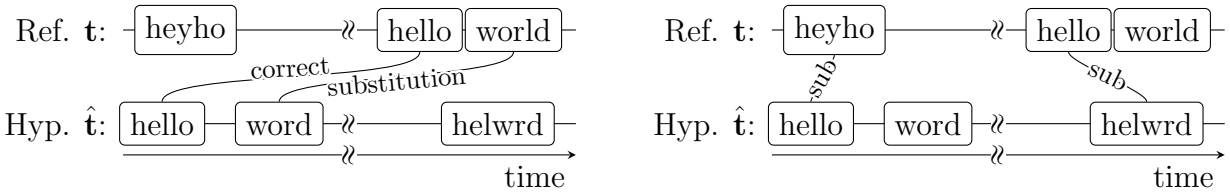
Related own publications

The performance measures presented in this chapter were developed in [OC7, OC16, OC8]. The discussions about the time-constrained WER in Section 3.1 are extended with additional experiments over [OC16] to gain a deeper insight into the effect of the time constraint on the matching. All algorithms are implemented in the open-source MeetEval toolkit[1].

Approaches have been proposed for solving the assignment problem. The cpWER [20] and the ORC-WER [38], and with tools in the NIST Scoring Toolkit (SCTK)[1]. The cpWER assumes speaker-attributed transcripts, i.e., that speaker labels $\hat{\ell}$ are estimated along with the transcript and that these labels represent actual speaker identities. The ORC-WER was proposed for the evaluation of speaker-agnostic CSS-style systems and works without speaker labels or with estimated labels that are uninformative w.r.t. the true speaker identity (e.g., the system output stream index in a CSS system). The `asclite` tool from the `SCTK` toolkit computes a word-level alignment across transcripts of multiple speakers, which leads to an in general over-optimistic speaker-agnostic measure. However, it incorporates a few heuristics and requires word-level timestamps, which makes it difficult to apply in practice. We here introduce an efficient algorithm for computing the ORC-WER and show a generalization of the ORC-WER for non-CSS-style systems that are allowed to change the utterance order, named MIMO-WER. We also introduce the DI-cpWER as a speaker-agnostic WER that is compatible to the cpWER in the sense that it is a lower bound on the cpWER assuming oracle speaker attribution.

The problem of implausible temporal alignments, where words are matched as correctly recognized across long temporal distances, is, however, not addressed in most WER measures. It has been discussed in the past [64, 76] (also in the `SCTK` toolkit), but the solutions were mostly discarded due to complexity or because it did not yield significant improvements for the analysis of the problem at hand. These approaches are discussed in more detail later in Section 3.1.5. We introduce with the time-constrained WER an approach that is simpler to apply and we show that it significantly improves the alignment quality and the interpretability of the resulting metric for meeting transcription tasks.

[1] https://github.com/usnistgov/SCTK



(a) Non-time-constrained matching. Here, it is cheaper (lower number of errors) to choose one correct match and one substitution across a long temporal distance instead of a physically more plausible matching of two substitutions.

(b) Time-constrained matching. The time constraint forbids matches across long temporal distances so that matchings appear locally.

Figure 3.1: Example matchings for a time-constrained and non-time-constrained WER. A long temporal gap is indicated by the broken time axis.

This chapter is structured as follows. It first shows how to improve the alignment quality for long sequences by a time constraint in Section 3.1, which forbids words to be matched across large temporal distances. A detailed analysis of the behavior of the time constraint and its only hyperparameter, the collar, is given. It extends slightly over the analysis presented in [OC16] and [OC8]. In Section 3.2, it is explained how the assignment problem can be solved for computing the WER in the meeting scenario, introducing the ORC-WER [38, OC7, 77], the MIMO-WER [OC7] and the DI-cpWER [36, OC8]. Efficient algorithms for these WER definitions are then developed in Section 3.3, where the derivation of the efficient ORC-WER algorithm extends over the discussions in [OC7, OC8]. The WERs are analyzed w.r.t. different error types, relations between WERs and execution time in Section 3.5. This analysis is mostly taken from [OC7, OC8] and extended slightly. Before, in Section 3.4, a visualization scheme and toolkit is described for detailed system analysis and development.

In this chapter, the estimated transcript at a system output is called hypothesis and the ground-truth transcription is called reference.

3.1 Time-constrained Word Error Rate

The standard Levenshtein distance defined in Eq. (2.19) produces reasonable alignments for short sequences. But, since there is no constraint on the temporal relation of the words in the reference and interference, the alignment can become unreasonable when the sequences become longer. Words can be matched as correct or substituted that are physically impossible to stem from the same acoustic event, as indicated in Fig. 3.1a by the broken time axis.

As a solution to this problem, we introduce a constraint on the maximum time difference, called a *collar* c, such that two words are only allowed to match (correct match or substitution) when their distance in time does not exceed c. This leads to the time-constrained Levenshtein distance tclev [OC16] as a drop-in replacement for the Levenshtein distance from Eq. (2.19) for the WER. It is described in detail below.

3.1.1 Time-constrained Levenshtein Distance

The time-constrained Levenshtein distance $\text{tclev}(\mathbf{t}, \hat{\mathbf{t}}, \mathbf{b}, \mathbf{e}, \hat{\mathbf{b}}, \hat{\mathbf{e}})$ [OC16] requires the begin and end timestamps for each ground truth word in $\mathbf{b} = [b_1, b_2, ...]$ and $\mathbf{e} = [e_1, e_2, ...]$ and for each estimated word in $\hat{\mathbf{b}} = [\hat{b}_1, \hat{b}_2, ...]$ and $\hat{\mathbf{e}} = [\hat{e}_1, \hat{e}_2, ...]$. The utterance index $(\cdot)_u$ is again omitted since the algorithm only operates on a single word sequence from a single utterance. The timestamps are neglected from the argument list of tclev in the following. It is always assumed that corresponding timestamps are available for the transcripts.

The time-constrained Levenshtein distance tclev is computed similarly to the standard Levenshtein distance in Eq. (2.19), but with an additional constraint on the substitution cost $C_{\text{C/S}}$. The corresponding Levenshtein distance matrix $\mathbf{L}^{(\text{tc})} = [l_{ij}^{(\text{tc})}] = [\text{tclev}(\mathbf{t}_{\leqslant i}, \hat{\mathbf{t}}_{\leqslant j})]$ is computed with the following recursion (compare Eq. (2.19)):

$$l_{i,j}^{(\text{tc})} = \min \begin{cases} l_{i-1,j} + C_{\text{D}} \\ l_{i,j-1} + C_{\text{I}} \\ l_{i-1,j-1} + C_{\text{C/S}}^{(\text{tc})}(t_i, \hat{t}_j, b_i, \hat{b}_j, e_i, \hat{e}_j), \end{cases} \tag{3.1}$$

where $C_{\text{D}} = 1$ and $C_{\text{I}} = 1$ are defined identically to the standard Levenshtein distance and

$$C_{\text{C/S}}^{(\text{tc})}(t, \hat{t}, b_i, \hat{b}_j, e_i, \hat{e}_j) = \begin{cases} \infty & \text{if } b_i - \hat{e}_j \geqslant c \text{ or } \hat{b}_j - e_i \geqslant c, \\ C_{\text{C}} & \text{if otherwise } t_i = \hat{t}_j, \\ C_{\text{S}} & \text{otherwise,} \end{cases} \tag{3.2}$$

is the substitution cost that is set to infinity if the timestamps indicate a gap between the words larger than the collar c and otherwise equal to the substitution cost $C_{\text{C/S}}(t_i, \hat{t}_j)$ from the Levenshtein distance. b_i, e_i and $\hat{b}_i, \hat{e}_i$ are the begin and end times of the i-th word in $\mathbf{t}$ and $\hat{\mathbf{t}}$, respectively.

This standard Levenshtein distance can be replaced with this time-constrained variant everywhere where the standard Levenshtein distance is applied and word-level timestamps are available. Any WER definition can thus be made time-constrained by replacing the Levenshtein distance definition.

3.1.1.1 Speedup by Pruning the Search Space

The computation of the time-constrained Levenshtein distance can be sped up compared to the standard Levenshtein distance by skipping unnecessary computations without compromising optimality. Since no diagonal transitions are allowed where $C_{\text{C/S}}^{(\text{tc})} = \infty$, the best cost that can be achieved by a path exclusively through that region is the Manhattan distance (the sum of the vertical and horizontal distance between the cells), and all paths that connect two cells exclusively though a region where $C_{\text{C/S}}^{(\text{tc})} = \infty$ cause the same cost. This Manhattan distance is also the highest possible cost for a path connecting two cells through a non-time-constrained region where $C_{\text{C/S}}^{(\text{tc})} < \infty$. From this we can conclude that, if such a path exists, a path connecting two cells through a region where $C_{\text{C/S}}^{(\text{tc})} < \infty$ cannot have a worse cost than a path through $C_{\text{C/S}}^{(\text{tc})} = \infty$. Computations of the Levenshtein matrix can thus be skipped in continuous regions where $C_{\text{C/S}}^{(\text{tc})} = \infty$.

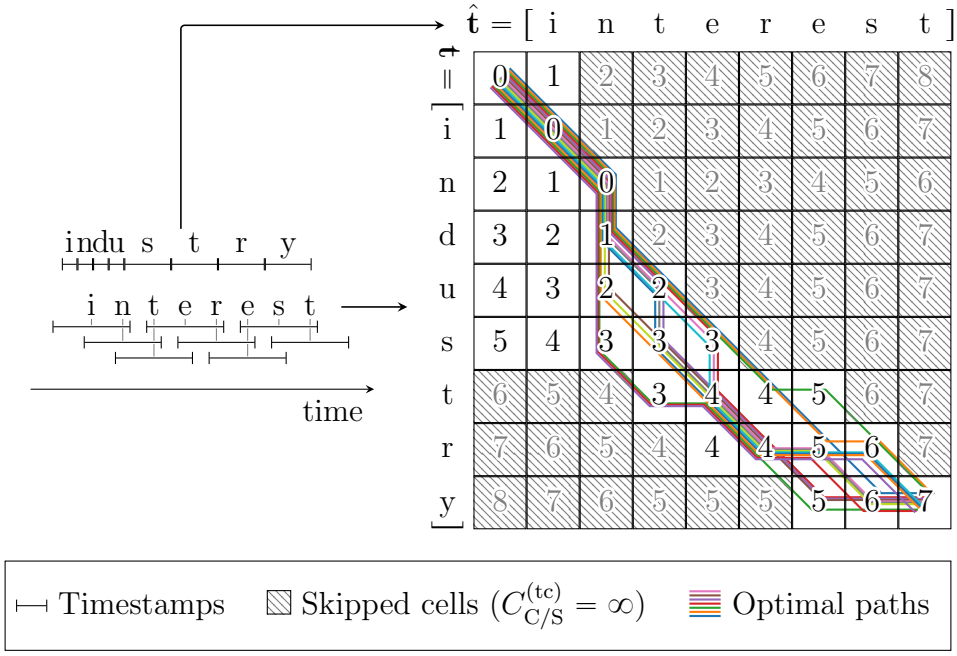


Figure 3.2: Pruning for the time-constrained Levenshtein distance. Characters are treated as words for illustration. The matching is computed between $\mathbf{t}$ = [`i n d u s t r y`] and $\hat{\mathbf{t}}$ = [`i n t e r e s t`] and the temporal overlaps between words are indicated by the vertical bars on the left. The computation of the hatched-out cells can be skipped. Only a sub-set of the optimal paths is displayed that go through the non-skipped region. The formerly optimal paths (without a time constraint, see Fig. 2.9) are disallowed by the time constraint.

When both start and end times are increasing for both sequences, then the regions that can be skipped form continuous regions at the upper right and lower left corners of the matrix, as visualized in Fig. 3.2. If the words are either not sorted by their begin time or if their length differences lead to their end times being unsorted, then there can be more disconnected regions spread throughout the matrix that can be skipped. But since these regions are typically small (low expected speedup by skipping them) and detecting them is computationally expensive, we only skip the large connected triangular regions at the corners of the matrix.

To simplify the implementation further, we first compute cumulative word-level timestamps to guarantee that the begin and end times used for pruning are monotonically increasing. We set the cumulative begin time to the earliest begin time that comes after this word $b_i^{(\text{cum})} = \min_{j \geqslant i} b_j$ and the cumulative end time to the latest end time that comes before this word $e_i^{(\text{cum})} = \max_{j \leqslant i} e_j$. These timestamps are always sorted (both begin and end) and used to detect the skipped regions. The above pruning still guarantees that the optimal path will be found.

Fig. 3.2 shows a pruned Levenshtein matrix for an example time-constrained matching. The temporal alignment is indicated on the left, where characters with overlapping bars overlap temporally. The Levenshtein matrix for the time-constrained matching is shown on the right, where cells with $C^{(tc)}_{C/S} = \infty$ are hatched out. Note that the costs in these cells do not resemble the Manhattan distance to any specific other cell, but the minimum cost required to reach each cell. In this example, no optimal path goes through the hatched-out

Table 3.1: Speedup of the time-constrained Levenshtein distance computation by pruning. The runtime was averaged over 1000 runs over the full LibriCSS-raw dataset.

Dataset	Cells skipped	Timings (ms)		Speedup
		without pruning	with pruning	
LibriCSS	87.28 %	0.342	0.210	1.63

region. The computation of all hatched-out cells can be skipped without influencing the computed distance.

Impact of pruning on the runtime To evaluate the impact of the pruning on the runtime, we conducted an experiment on the LibriCSS-raw dataset. Using a transcript produced by a model from a later chapter[2], we compute the number of cells that were skipped in the pruned computation and also track the time it took to compute the Levenshtein distance with and without pruning. The results are shown in Table 3.1. The runtime is measured on a single core of an Intel Xeon E5-1620 CPU at 3.7 GHz and averaged over 1000 runs.

Even though a significant amount of cells is skipped, the speedup was only 1.63. This indicates that the pruning only slightly outweighs the more cache-friendly memory access patterns of the unpruned version and the logic for determining which cells to skip. For the more complex ORC-WER and MIMO-WER that will be presented later in Section 3.2 and whose runtime will be analyzed in Section 3.5.7, however, the pruning has a larger impact on the practical execution time.

3.1.2 Pseudo-word-level Timestamps

The time-constrained Levenshtein distance in Eq. (3.1) requires word-level timestamps, which are often not available. Obtaining them for a reference dataset is expensive (as it requires manual annotation), error-prone and often not deterministic as there is some room for interpretation. Many widely-used datasets do not provide word-level timestamps, e.g., LibriSpeech [43] and WSJ [42]. For the hypothesis transcript, the timestamps are sometimes available, but more often not since modern attention-based end-to-end architectures do not provide a direct connection between the output tokens and the input frames. Obtaining a forced alignment from modern speech recognizers often requires additional effort, such as dedicated modelling or an additional forward pass through the network [78].

Segment-level timestamps, however, are usually available independently of the ASR system through a VAD or diarization stage and strategies exist for estimating *pseudo*-word-level timestamps from them [OC16]. These are visualized in Fig. 3.3. The simplest strategy is to assign each word the *(1) full segment*-level annotation, which does not represent the true word-level timestamps well. The true word boundaries can be better approximated using *(2) equally sized intervals* for the words, equally distributed within the segment. The most accurate estimation is achieved by using *(3) character-based* timestamps, where the length of the segment is divided among the words proportional to their length in number of characters. This assumes that the word length is a good approximation for the time it takes

[2] Details are unimportant here.

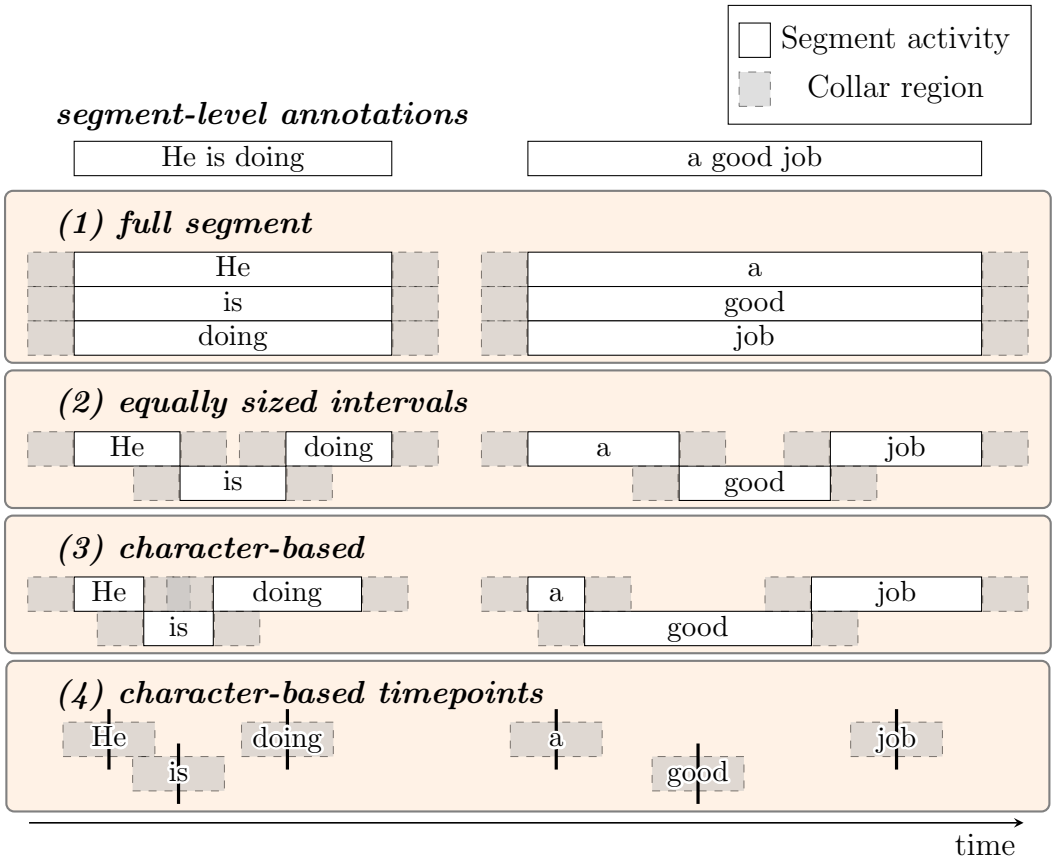


Figure 3.3: Different strategies for estimating word-level timestamps from segment-level timestamps. *(Taken from [OC16])*

to pronounce a word, which we confirmed for the LibriSpeech and TIMIT datasets containing read English speech. The difference between the (true) average word length and the word length estimated with the character-based approach never exceeds 100 ms, including a few outliers with extreme annotation errors. Note that the approximation becomes less accurate for longer segments and when the segments contain speech pauses.

3.1.2.1 Fooling the Metric With the Wrong Choice of Pseudo-word-level Timestamps

While the character-based estimation of the word-level timestamps is the most accurate, it can be exploited to fool the metric, i.e., improve the performance w.r.t. the metric without actually improving the system. When the estimated word length depends on the hypothesis segment length, the metric can be improved by increasing the segment length to cover a larger area. This increases the probability of correct matches and substitutions and decreases the probability of insertions and deletions since more words overlap with the enlarged segment.

It is here suggested to reduce the length of the estimated word-level timestamps to 0, resulting in *(4) character-based points* instead of intervals independent of the segment length. By doing this, a system is penalized for overly long segments since the precision of the time-point estimation decreases with larger segment sizes.

3.1.3 Choosing the Collar Value

A collar $c > 0$ is required to compensate inaccuracies in the timestamps for both reference and hypothesis and to allow for some degree of freedom for the system where exact timestamp positions are ambiguous. Both the hypothesis and the reference may contain significant errors in the temporal annotations.

Annotation errors in the reference The reference transcript is usually manually annotated and thus contains errors. Humans tend to over-estimate the length of speech activity to ensure that all speech is covered. In popular datasets, e.g., LibriSpeech and TIMIT, the human annotations extend over annotations created by a powerful VAD[3] by on average 660 ms and 400 ms, respectively. Re-recorded datasets, such as LibriCSS [14], can contain additional issues introduced by the re-recording process. These include offsets caused by the playback hardware and sound propagation delays and varying offsets caused by sample rate offsets, which can vary between speakers.

System errors As already seen from the reference timestamps, the exact timestamp position is often ambiguous such that a system is very unlikely to produce the exact same timestamps as the reference. Speech activity estimates are often smoothed [36] to avoid very short pauses and speech segments, which can introduce both over-estimation and under-estimation of the segment length. While minor noise in the timestamps should be mitigated by the collar, larger issues, such as systematic shifts, should be counted as errors.

3.1.3.1 Compensating for the Errors Introduced by the Pseudo-word-level Annotation Errors

As a lower bound on its value, the collar should be chosen large enough that it compensates all errors produced by the approximation of the word-level timestamps from Section 3.1.2. This depends on the dataset and the quality of the segment-level timestamps: If the segments are longer or the segment length is over-estimated, then the collar should be chosen larger to accommodate the additional uncertainty in the word-level timestamps.

For LibriSpeech and TIMIT, the gap sizes between the estimated word-level timestamps and the true word-level timestamps, determined through a forced alignment, is shown in Fig. 3.4 as a case study. For most words, the estimated pseudo-word-level timestamps overlap with the true word-level timestamps, indicated by the peak at 0 in the density plot. Most gaps for non-overlapping words are small, i.e., below 1 s and 2 s, respectively, and rapidly decreasing, but the collar should account for all of these errors, including the largest outliers. The largest gaps are found around 2.7 s and 3.6 s for LibriSpeech and TIMIT, respectively, providing a lower bound for the collar of at least 3.6 s. As can be seen from this small case study, the collar required to compensate all errors in the word-level timestamps varies widely between datasets, so it is recommended to choose the collar significantly larger than the lower bound found here.

[3] VAD taken from the respective KALDI [9] recipes

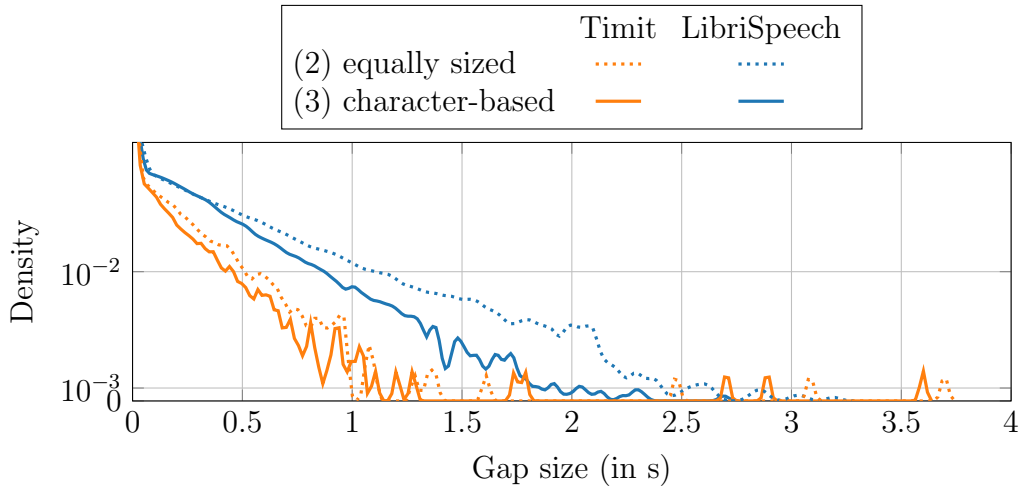


Figure 3.4: The gap between the estimated pseudo-word-level timestamps and the forced-aligned word-level timestamps for LibriSpeech and TIMIT. *(Taken from [OC16])*

3.1.3.2 Approaching the "desired" WER

As another case study, we look at the recognition result of a simple meeting transcription system, composed of a speaker-attributed TS-SEP separator [36] and the Whisper ASR system [79][4], on the LibriCSS-raw dataset. Fig. 3.5 shows the behavior of the time-constrained WER[5] over a varying collar. The time-constrained WER approaches the non-time-constrained WER for larger collars and both become equal for $c \to \infty$.[6] To find the optimal value for the collar, we first define a "desired" WER as a target that should be approximated with the time-constrained WER and then determine which value for c achieves this best.

"Desired" WER For this experiment, we define the desired WER as the time-constrained WER with a collar of 0 and the true word-level boundaries. The "true" timestamps for the reference are determined with a forced alignment using a system from KALDI [9] and the hypothesis word-level timestamps are obtained from the Whisper recognizer along with the transcript. Whisper is applied to the estimated speech signals at the output of the TS-SEP system directly. This is the physically most plausible WER: two words can only match as correct or substituted if they correspond to the same acoustic event, which means that they must appear at exactly the same point in time.

Optimal collar value for approximating the desired WER In Fig. 3.5, the time-constrained WER is close to the desired WER for a collar value between 3 s to 5 s, where the collar successfully compensates errors from the pseudo-word-level annotation estimation but still disallows temporally implausible matchings. The bottom of Fig. 3.5 shows the number of words that are matched in the desired WER but that would be disallowed by the collar (note that the WER with the collar is computed using estimated pseudo-word-level time-stamps

[4] Whisper is used in the `large-v2` configuration.
[5] More specifically the tcpWER, described later in Section 3.2.1. The details of how the assignment problem is solved are not important for this case study.
[6] Setting the collar to the the length of the recording is enough.

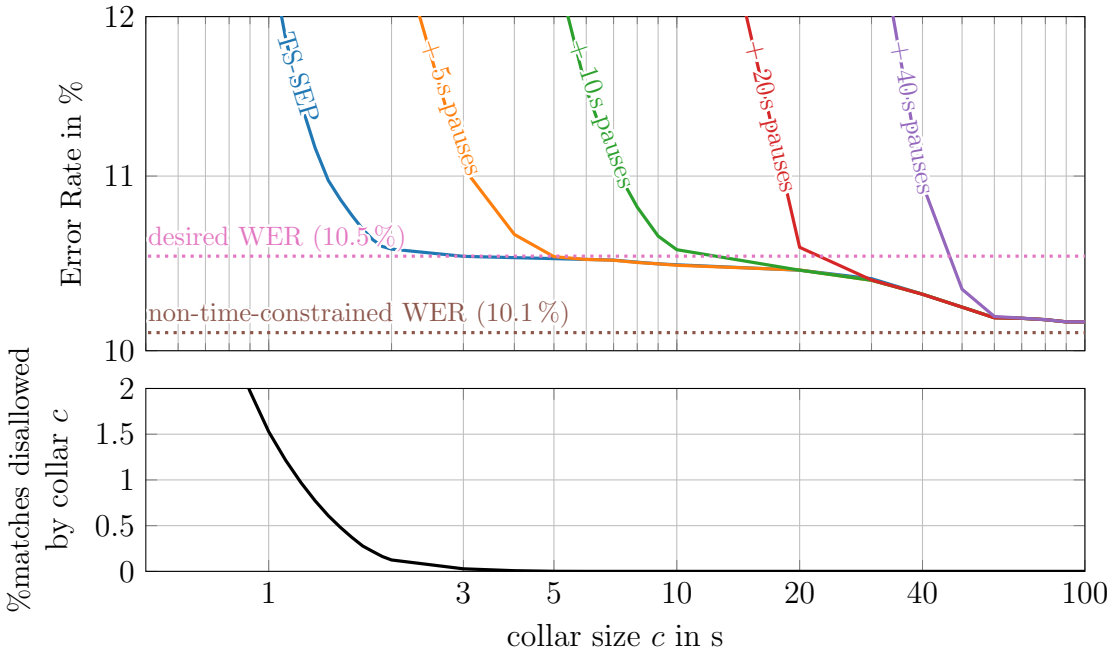


Figure 3.5: Behavior of the time-constrained WER for varying values of the collar c. *Top:* The time-constrained WER over collar values on the LibriCSS dataset. The "desired WER" is determined with forced alignments. "+ n s pauses" means that segments were artificially merged to include pauses up to n seconds to simulate segment-length over-estimation. *Bottom:* Number of matches that the collar in the time-constrained WER disallows but that are matched in the desired WER. *(Taken from [OC16])*

while the desired WER uses ground-truth timestamps). This value should be 0, which leads to a new lower bound of about 4 s.

Segmentation errors To investigate the impact of wrongly estimated segment boundaries, we artificially modify the hypothesis to over-estimate the segment lengths by merging multiple adjacent segments when the gap between them is shorter than n seconds. This is denoted by "+ n s pauses" in Fig. 3.5. While merging segments that are close together is no severe error and likely does not hurt downstream systems, we argue that the WER should penalize this merging of segments when the gap is too large. A boundary of 5 s seems reasonable here since it seems unlikely that a pause longer than this remains undetected by a decent VAD and the errors in the reference annotations are typically smaller than this. A collar of 5 s penalizes merges of segments with gaps larger than 5 s but allows merges with gaps smaller than 5 s, so we argue that 5 s is a reasonable choice for the collar.

3.1.3.3 Comparison to the DER collar

The proposed WER collar is inspired by the collar used in the DER (see Section 2.8.5), but its effect is different. The collar in the DER defines regions around the borders of segments that are excluded from the scoring, such that a collar of ∞ leads to a DER of 0. The collar in the time-constrained WER, however, excludes possible matches, and a collar of ∞ results in a time-constrained WER equal to the non-time-constrained WER.

3.1.3.4 A General Recommendation for the Collar

The case studies in this section provide a lower bound at 3.6 s but show that, when the collar is larger than this lower bound, it impacts the WER only marginally if not chosen too large. We thus recommend a value of 5 s as a starting point for the collar, which should be checked and adjusted when evaluating a new dataset. We validated that $c = 5\,\text{s}$ yields plausible WERs for LibriCSS and all datasets used in the CHiME-8 challenge, namely NOTSOFAR [1] (where the time-constrained ORC-WER, described later in Section 3.2.2, and the time-constrained cpWER, described later in Section 3.2.1, were chosen as the official metrics), the CHiME-6 dataset [20] and the Mixer6 dataset [80].

3.1.4 Handling Temporally Overlapping Words

Until now it was assumed that the order of words corresponds to the timestamps associated with the words, i.e., words do not overlap and are sorted by begin time. But in Eq. (3.1) this is not strictly required; the order of words can be arbitrary and unrelated to the timestamps. While large discrepancies in the order are physically impossible and should thus be avoided, small discrepancies where the beginning of a segment overlaps slightly with the previous segment may appear in both the reference and the hypothesis transcript. We here recommend to keep the order imposed by the segments, i.e., sort the segments and not the words by begin time. The word-level timestamps used for the constraint may then not be increasing.

This approach is compatible with the non-time-constrained WER in that a collar of $c \to \infty$ leads to the same value for the time-constrained and non-time-constrained WERs. If words would be sorted by their begin times, their order would change between time-constrained and non-time-constrained WERs, which is not desired.

3.1.5 Relation to Other Metrics

The idea of a time-constraint for computing a WER is not entirely new. It has been used in other tools before but was either based on heuristics or was disregarded for not providing enough benefit for evaluation of simpler scenarios.

3.1.5.1 NIST Tools

The `sclite` and `asclite` tools have been recommended for the evaluation of speech recognition systems in NIST evaluations [63, 64]. `sclite` is designed for short utterance-level evaluation while `asclite` uses a multi-dimensional Levenshtein algorithm for meeting-style data.

Single-utterance evaluation `sclite` incorporates the temporal distance between words into the substitution in the Wagner-Fisher algorithm. This encourages the algorithm to choose an insertion and a deletion over a substitution when the distance between words is large. Correct matches are also penalized over long distances, but not forbidden. The constraint is thus less strict than the constraint in the proposed time-constrained Levenshtein distance.

Figure 3.6: An example input and output transcript of a meeting transcription system. Each box represents an utterance and each letter a word. Timestamps are indicated with horizontal position, where time flows from left to right. This example will be used to show how the different WERs solve the assignment problem. *(Taken from [OC8])*

The `sclite` tool additionally does not use a collar and does not estimate word-level timestamps from segment-level timestamps, which limits its applicability for realistic systems that do not always provide word-level timestamps.

Meeting-level evaluation with asclite `asclite` does not provide the same time alignment as `sclite`, but only a rough time-constraint based on a heuristics for reducing computational complexity. It was not designed to improve the quality of the matching. Hard boundaries across which any matches are disallowed are inserted where no segments overlap, but the matching between these boundaries is not constrained by time. As discussed earlier, this can lead to implausible matchings, although the extent is limited by the hard boundaries.

3.1.5.2 Time-constrained WER for single-utterance evaluation

In [76], a time-constraint was considered for the evaluation of the switchboard dataset in 1996. No collar was used, but instead of a hard constraint, the alignment was penalized to minimize the accumulated word distances, likely using a similar approach to `sclite` described above. It was found that when using speaker turns as the unit of alignment instead of word-level timestamps, the score is typically two percentage points better but the additional complexity of obtaining the word-level timestamps is introduced. It was thus disregarded.

The findings from [76] are in-line with our findings that the time-constraint only has a meaningful impact when the evaluated transcripts are long enough. We also suspect that the modified algorithm that minimizes the aggregated distances introduces a larger computational overhead than our collar-based approach since no clear boundaries can be computed where words are disallowed to match.

3.2 Meeting-level Word Error Rates

In addition to the temporal alignment, the assignment problem (see Section 2.4) also needs to be solved when computing the WER for meeting scenarios. A mapping must be found between reference utterances and system output streams before the (time-constrained or non-time-constrained) Levenshtein distance can be computed.

For the classical utterance-level WER, the assignment is trivial: since each utterance is processed individually, the corresponding reference is directly given. For meeting-level WER, however, such a direct mapping of reference to hypothesis usually does not exist. For most systems (compare Chapter 2), there is an ambiguity in the speaker labels, i.e., it is either unknown which speaker in the hypothesis corresponds to which speaker in the reference or the system does not predict speaker labels at all. Such an example is depicted in Fig. 3.6,

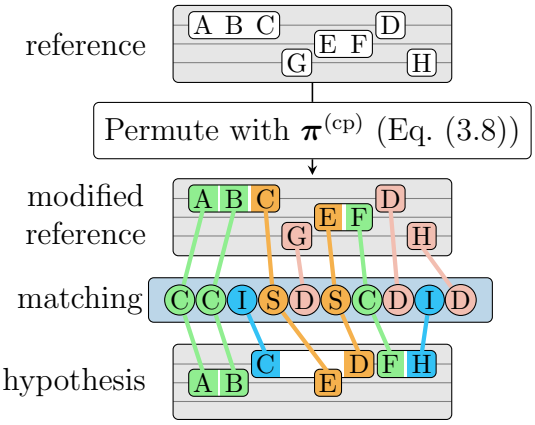


Figure 3.7: cpWER = $\frac{2+3+2}{8}$ = 87.5%. The hypothesis streams are reordered with Eq. (3.8) and empty streams are inserted such that the error rate is minimized. *(Taken from [OC8])*

where a meeting ASR system outputs two streams, but the reference transcript contains three speakers. The ASR system additionally made segmentation and recognition errors: It split the reference segments "A B C" and "D" into "A B" and "C D", missed the segment "G" and swapped the temporal positions of "D" and "F". It is not immediately obvious how the reference and hypothesis transcript can be brought together for computing a WER. This example will be used below throughout the discussions to visualize how the different meeting-level WER definitions react to these errors.

There are many approaches for finding this assignment, targeting different systems and penalizing different errors. When speaker identities are ignored while solving the permutation problem, the resulting WER is called *speaker-agnostic*. When speaker identities are taken into account, and speaker attribution errors are penalized, the resulting WER is called *speaker-attributed*.

This section uses the term Levenshtein distance, but it is always implied that the time-constrained Levenshtein distance can also be used instead. All WERs are defined by Eq. (2.16) by replacing the Levenshtein distance with the corresponding definition from the remainder of this section.

3.2.1 Speaker-attributed Evaluation With (t)cpWER

One commonly used metric in meeting scenarios is the Concatenated minimum-Permutation WER (cpWER) [20]. It is a speaker-attributed WER which finds the assignment between reference speakers and hypothesis speakers such that the error rate is minimized.

It is computed by, for each speaker, concatenating the transcriptions of all utterances of that speaker, and then computing the Levenshtein distance between the concatenated transcripts, subject to a permutation across the speakers. Let us first define as

$$\mathbf{t}_{\ell,k}^{(\mathrm{cat})} = [..., \mathbf{t}_u, ... : u \in \mathcal{U} \wedge \ell_u = k] \tag{3.3}$$

and

$$\hat{\mathbf{t}}_{\hat{\ell},k}^{(\mathrm{cat})} = [..., \hat{\mathbf{t}}_u, ... : u \in \hat{\mathcal{U}} \wedge \hat{\ell}_u = k] \tag{3.4}$$

the word sequence obtained by concatenating all reference utterance or hypothesis segment transcripts that get assigned label k by the ground-truth labels ℓ or estimated labels $\hat{\ell}$, respectively. Setting k to a label not present in ℓ or $\hat{\ell}$ leads to an empty transcript. The notation $[\mathbf{t}_1, ..., \mathbf{t}_u] = [t_{1,1}, ..., t_{1,\dim(\mathbf{t}_1)}, ..., t_{u,1}, ..., t_{u,\dim(\mathbf{t}_1)}]$ means the concatenation of the transcripts (as row vectors) into a single transcript, and $[..., \mathbf{t}_u, ... : \text{condition}(u)]$ is the concatenation of all transcripts that fulfill the condition condition(u). For a more concise notation, we denote the concatenated reference and hypothesis transcripts for each reference speaker k as

$$\mathbf{t}_k^{(\text{spk})} = \mathbf{t}_{\ell,k}^{(\text{cat})} \tag{3.5}$$

$$\hat{\mathbf{t}}_k^{(\text{spk})} = \hat{\mathbf{t}}_{\hat{\ell},k}^{(\text{cat})} \tag{3.6}$$

to make clear that these concatenated transcripts contain all words from speaker k. The labels ℓ are the labels that correspond to the reference utterances and $\hat{\ell}$ the labels at the system output. We here assume that the utterances are temporally ordered by the global utterance order relation $<_\text{u}$. The concatenated minimum-permutation Levenshtein distance is computed similarly to uPIT (compare Eq. (2.9)), but as the minimum over Levenshtein distances:

$$\text{lev}^{(\text{cp})}((\mathbf{t}_1^{(\text{spk})}, ..., \mathbf{t}_K^{(\text{spk})}), (\hat{\mathbf{t}}_1^{(\text{spk})}, ..., \hat{\mathbf{t}}_K^{(\text{spk})})) = \sum_{k=1}^{K} \text{lev}\left(\mathbf{t}_{\pi_k^{(\text{cp})}}^{(\text{spk})}, \hat{\mathbf{t}}_k^{(\text{spk})}\right), \tag{3.7}$$

with the permutation

$$\boldsymbol{\pi}^{(\text{cp})} = \underset{\boldsymbol{\pi} \in \mathcal{P}(K)}{\arg\min} \sum_{k=1}^{K} \text{lev}\left(\mathbf{t}_{\pi_k}^{(\text{spk})}, \hat{\mathbf{t}}_k^{(\text{spk})}\right), \tag{3.8}$$

where $\boldsymbol{\pi} = [\pi_1, ..., \pi_K]$, with $\forall i, j : \pi_i \neq \pi_j$, is a permutation vector and $\mathcal{P}(K)$ is the set of all permutation vectors of length K.

In Fig. 3.7, the cpWER matching is depicted for one example. The reference speakers are permuted and an empty dummy stream is inserted so that a matching can be computed between the reference and permuted hypothesis. The alignment is performed for each stream individually, i.e., segment "H" on the third reference stream cannot be matched with segment "FH" on the second permuted hypothesis stream.

The time-constrained variant of the cpWER, the Time-Constrained minimum-Permutation WER (tcpWER), is computed similarly, but with the time-constrained Levenshtein distance tclev.

Impact of speaker confusions Since the (t)cpWER groups transcripts by speakers before computing the WER, it is speaker-attributed and any speaker-attribution errors up to a permutation of the speaker labels are reflected in the WER. Specifically, a wrongly assigned utterance ideally creates deletion errors for the true speaker in the reference and insertion errors for the (wrongly) estimated speaker in the hypothesis. In practice, however, word-level errors are mixed with these speaker-attribution-induced errors, potentially creating substitution errors and correct matches in the wrongly assigned regions, such that speaker confusions usually cannot be detected by the cpWER alone.

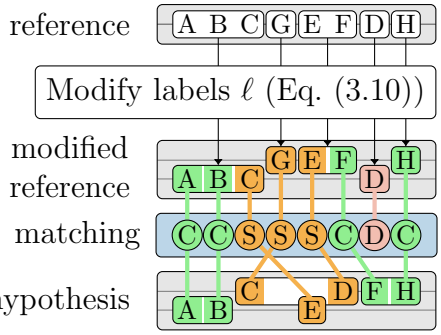


Figure 3.8: ORC-WER = $\frac{0+1+3}{8}$ = 50.0%. The reference labels ℓ are modified with Eq. (3.10) to minimize the error rate and preserve the global order of segments. *(Taken from [OC8])*

Impact of utterance splits and merges The (t)cpWER is unaffected by utterance splits and merges, i.e., the splitting of an utterance into multiple parts or the merging of multiple utterances into a single part, as long as the speaker labels are not changed by merging or splitting. This is because all transcripts with the same speaker label are concatenated into one long transcript before the assignment problem is solved.

3.2.2 Speaker-agnostic Evaluation With ORC-WER

When speaker labels are not available or performance should be evaluated excluding the estimation of speaker labels, a speaker-agnostic metric is required. The Optimal Reference Combination WER (ORC-WER)[7] [38, OC7, 77] is designed for CSS-style systems, which outputs a low number of streams S, usually fewer than there are speakers ($S \leqslant K$), and the mapping of utterances to output streams can be arbitrary. Since no speaker labels are estimated, the assignment is computed on an utterance level, where reference utterances are assigned to the estimated streams.

The ORC-WER makes assumptions on the *utterance-consistency* and *utterance-order* when computing the assignment. The *utterance-consistency* constraint requires that an utterance is placed consistently on a single output stream, i.e., it cannot be split into multiple parts such that utterance-split errors are disallowed. The *utterance-order* constraint requires that the (global) order of utterances is kept, i.e., two utterances must always appear in the same (temporal) order in the hypothesis and the reference, defined by $<_\mathrm{u}$.[8]

The assignment is solved on the reference side, modifying ℓ, such that it minimizes the error rate and adheres to the utterance-order and utterance-consistency constraints. Since the estimated transcript is not modified, we group and concatenate it into one concatenated transcript per estimated stream label:

$$\hat{\mathbf{t}}_s^{(\mathrm{str})} = \hat{\mathbf{t}}_{\hat{\ell},s}^{(\mathrm{cat})} \tag{3.9}$$

It is again assumed that the utterances are ordered by the global utterance order relation $<_\mathrm{u}$. This concatenation is equal to $\hat{\mathbf{t}}_s^{(\mathrm{spk})}$ from the cpWER, but named differently to highlight that

[7] First proposed by Sklyar in [38] and in parallel, without giving it a name, by Raj in [77].

[8] The utterance order relation here represents the order of the utterances in the input. When computing the ORC-WER, this is usually the temporal order as defined by utterance begin times, but the order may be different from the temporal order, e.g., the ORC-WER is generalized to the MIMO-WER below.

the ORC-WER does not treat the stream indices as speaker labels. The ORC Levenshtein distance is then computed by enumerating all assignments $\{\tilde{\ell}_u\} \in \{1, ..., S\}^{|\mathcal{U}|}$ of ground-truth utterances to streams, here represented as a set of assigned stream labels, and choosing the one that minimizes the overall Levenshtein distance across all streams:

$$\mathrm{lev}^{(\mathrm{ORC})}\left(\mathbf{t}_{1,...,|\mathcal{U}|}, \hat{\mathbf{t}}^{(\mathrm{str})}_{1,...,S}\right) = \min_{1 \leqslant \tilde{\ell}_1, ..., \tilde{\ell}_{|\mathcal{U}|} \leqslant S} \sum_{s=1}^{S} \mathrm{lev}\left(\mathbf{t}^{(\mathrm{cat})}_{\tilde{\ell},s}, \hat{\mathbf{t}}^{(\mathrm{str})}_{s}\right). \tag{3.10}$$

The notation $\mathbf{t}_{1,...,|\mathcal{U}|} = (\mathbf{t}_1, ..., \mathbf{t}_{|\mathcal{U}|})$ means that all transcripts from $\mathbf{t}_1$ to $\mathbf{t}_{|\mathcal{U}|}$ are passed to $\mathrm{lev}^{(\mathrm{ORC})}$. The assigned reference stream transcriptions are constructed similarly to Eq. (3.9) by assuming that they are sorted according to the global utterance order $<_{\mathrm{u}}$ and then concatenating. The utterance-order constraint is ensured by sorting the segment transcripts by $<_{\mathrm{u}}$ before concatenation and the utterance-consistency constraint is ensured by treating every utterance transcript as a single unit such that each utterance is assigned completely to a stream. Given the two constraints above, the only degree of freedom left is the stream labels for each reference utterance. The minimum is computed over all $S^{|\mathcal{U}|}$ possible assignments.

An example matching is depicted in Fig. 3.8. The reference labels are modified to minimize the WER. The segment order cannot be modified: the reference segment "D" cannot be moved before "EF", so it cannot be matched with the "D" in the hypothesis segment "CD".

Impact of speaker confusions The reference speaker labels are modified such that the overall WER is minimized. So, the ORC-WER is to some extent invariant to speaker confusions, as long as each utterance is assigned only a single label, i.e., there are no utterance splits.

Impact of utterance splits and merges The ORC-WER groups all hypothesis segments with the same label into one stream and assigns the reference utterances to these streams. Because of this, the ORC-WER is unaffected by utterance merge errors, even if two utterances stemming from different speakers are merged. It is, however, affected by utterance split errors when a single reference utterance is split across multiple output streams. The part of the utterance that causes the smaller error is counted as an error, or mixed with other errors.

Impact of utterance order changes The ORC-WER penalizes changes in the order of utterances between the reference and the hypothesis, but only for utterances that are mapped to the same stream. Without a time constraint, it is impossible to detect order changes across streams. This sometimes has an unwanted negative effect on the WER, e.g., when two utterances appear with almost identical but inaccurate starting times in the reference or when a system is analyzed that orders utterances by speaker rather than by their original order, such as certain SOT architectures [18, 37].

3.2.3 Speaker-agnostic Evaluation With MIMO-WER

The MIMO-WER [OC7] is a speaker-agnostic metric that allows for more flexibility in the assignment than the ORC-WER. Specifically, it keeps the utterance-consistency constraint from the ORC-WER but makes a less strict assumption on the utterance order: Two utterances must appear in the same order in reference and hypothesis only if they are from

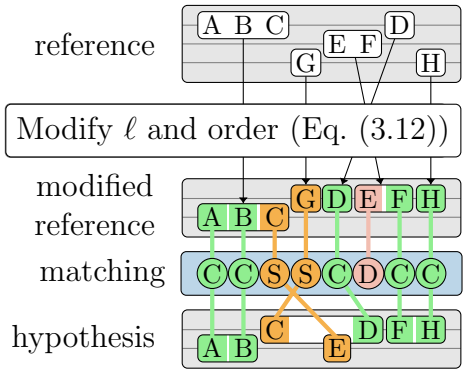


Figure 3.9: MIMO-WER = $\frac{0+1+2}{8}$ = 37.5%. The reference labels ℓ and order are modified to minimize the error rate, but the order within a speaker is preserved. *(Taken from [OC8])*

the same speaker. The order of utterances between speakers is free as long as it is consistent with the temporal order of utterances within a speaker. This allows for more flexibility in the assignment and can lead to a lower error rate than the ORC-WER. It is especially important for systems that do not necessarily follow a temporal order for all utterances, e.g., when a SOT-based system predicts the speakers sequentially.

This means that instead of just the speaker labels, also the order of the utterances may change. This can be represented by a *sequence MERGE* operation.

Sequence merge A sequence (or string) merge [81], also sometimes referred to as a shuffle [82] or interleaving, combines multiple sequences into one sequence while keeping the order of the elements within each sequence. Given two sequences $\mathbf{t}_1 = [t_{1,1}, t_{1,2}, ...]$ and $\mathbf{t}_2 = [t_{2,1}, t_{2,2}, ...]$, a sequence $\mathbf{t}^{(\text{MERGE})}$ is called a merge of $\mathbf{t}_1$ and $\mathbf{t}_2$ if $\mathbf{t}_1$ and $\mathbf{t}_2$ can be obtained by deleting elements from $\mathbf{t}^{(\text{MERGE})}$, the sets of deleted elements removed for obtaining $\mathbf{t}_1$ and $\mathbf{t}_2$ are disjoint and their union removes all elements from $\mathbf{t}^{(\text{MERGE})}$. We denote with $\text{MERGE}(\mathbf{t}_1, ..., \mathbf{t}_N)$ the set of all valid string merges of $\mathbf{t}_1$ to $\mathbf{t}_N$. The problem of determining if $\mathbf{t}^{(\text{MERGE})} \in \text{MERGE}(\mathbf{t}_1, ..., \mathbf{t}_N)$ is the merge recognition problem and is known to be NP-complete [81].

Formulating the MIMO Levenshtein distance with sequence merges The MIMO Levenshtein distance can be expressed as a minimum of the ORC Levenshtein distance across all sequence merges of the references utterances grouped by speaker compared to the estimated stream transcripts $\hat{\mathbf{t}}^{(\text{str})}_{1,...,S}$. The sequence merge is performed on an utterance-level (instead of the word-level for which it was introduced above) to ensure the utterance-consistency constraint, so:

$$\mathcal{M}^{(\text{MIMO})}_{\mathbf{t}_{1,...,|\mathcal{U}|},\ell_{1,...,|\mathcal{U}|}} = \text{MERGE}\left((\mathbf{t}_u : \ell_u = 1), ..., (\mathbf{t}_u : \ell_u = S)\right) \tag{3.11}$$

is the set of all merges of utterance sequences. The round parenthesis $(\mathbf{t}_{1,...,|\mathcal{U}|})$ denote a sequence of utterances and are chosen over the square brackets $[\mathbf{t}_{1,...,|\mathcal{U}|}]$, which denoted concatenation in cpWER and ORC-WER, to highlight that the utterances are *not* concatenated on a word-level, but each utterance's transcript is treated atomically. Each element in $\mathcal{M}^{(\text{MIMO})}$ represents a re-ordering of the utterances such that all utterances from the same speaker keep their temporal order. The MIMO-WER can now be computed as the

minimum over the ORC-WERs over all sequence merges in $\mathcal{M}^{(\text{MIMO})}$ compared to the stream transcripts $\hat{\mathbf{t}}^{(\text{str})}_{1,\ldots,S}$ as

$$\text{lev}^{(\text{MIMO})}\left(\mathbf{t}_{1,\ldots,|\mathcal{U}|}, \hat{\mathbf{t}}^{(\text{str})}_{1,\ldots,S}\right) = \min_{\tilde{\mathbf{t}}_{1,\ldots,|\mathcal{U}|} \in \mathcal{M}^{(\text{MIMO})}_{\mathbf{t}_{1,\ldots,|\mathcal{U}|}, \ell_{1,\ldots,|\mathcal{U}|}}} \text{lev}^{(\text{ORC})}\left(\tilde{\mathbf{t}}_{1,\ldots,|\mathcal{U}|}, \hat{\mathbf{t}}^{(\text{str})}_{1,\ldots,S}\right). \tag{3.12}$$

The reference speaker transcripts $\mathbf{t}^{(\text{spk})}_k$ and hypothesis stream transcripts $\hat{\mathbf{t}}^{(\text{spk})}_k$ are defined in the same way as for the cpWER and ORC-WER, respectively. The example in Fig. 3.9 shows how the MIMO-WER can reorder the reference segments. It moves the segment "D" before "EF" in the reference so that it is matched with the D in the hypothesis segment "CD".

This naive formulation of the MIMO Levenshtein distance computes too many assignments. While every element in MERGE($\mathbf{t}^{(\text{spk})}_1, ..., \mathbf{t}^{(\text{spk})}_K$) is unique and every assignment checked in the ORC Levenshtein distance is unique given one reference sequence, the combination of both (shuffle and stream assignment) creates duplicates. A more efficient algorithm is described in Section 3.3.4.

Impact of utterance order changes In contrast to the ORC-WER, the MIMO-WER allows for some changes in the order of utterances between speakers. Specifically, the order of utterances must be consistent with the temporal order of the utterances within each speaker. Let us define as the partial relation $<_{\text{stream}}$ the utterance order found on the system output streams, i.e., $u <_{\text{stream}} u'$ if both u and u' are on the same stream ($\hat{\ell}_u = \hat{\ell}_{u'}$) and u comes before u' on that stream. Let us additionally define as $<_{\text{spk}}$ the partial relation on the utterance order within each speaker, i.e., $u <_{\text{spk}} u'$ if both utterances come from the same speaker ($\ell_u = \ell_{u'}$) and u comes before u' in the temporal utterance ordering ($u <_{\text{u}} u'$). Given $<_{\text{stream}}$ and $<_{\text{spk}}$, there must exist a global ordering $<_{\text{total}}$ that orders all utterances and is consistent with both partial orderings $<_{\text{spk}}$ and $<_{\text{stream}}$. Note that $<_{\text{total}}$ does not have to resemble the temporal utterance order $<_{\text{u}}$ on a global level, but only within each speaker. The set MERGE contains exactly all sequences for which such a total order exists.

It is not always obvious when a particular ordering is implausible, i.e., when such a global ordering relation $<_{\text{total}}$ does not exist. For example, given $\mathbf{t} = [\mathtt{a}, \mathtt{b}, \mathtt{x}, \mathtt{y}]$, $\ell = [1, 1, 2, 2]$, speaker 1 has utterances `a` and `b` while speaker 2 has utterances `x` and `y`. If the assignment would result in the sequences $[\mathtt{b}, \mathtt{x}]$ and $[\mathtt{y}, \mathtt{a}]$, this would be implausible. While both sequences themselves are compatible with both the speaker order and the stream order, there exists no global order that is compatible with all of them. This example contains a contradiction: `a` must come before `b` (in $<_{\text{spk}}$), `b` must come before `x` (in $<_{\text{stream}}$), `x` before `y` (in $<_{\text{spk}}$). Transitively, `y` must come before `a`, which contradicts $<_{\text{stream}}$, where `a` comes before `y`.

ORC-WER as a special case of MIMO-WER The MIMO-WER was defined as a generalization of the ORC-WER in Eq. (3.12). The ORC-WER is as such a special case of the MIMO-WER when only a single reference speaker is present. The ORC-WER can thus be computed by replacing all reference labels with a dummy label $\ell_u = 0 \ \forall u \in \mathcal{U}$, in which case only a single string merge exists, and executing the MIMO-WER algorithm.

3.2.4 Speaker-agnostic System Analysis With DI-cpWER

When developing a meeting transcription pipeline that contains a dedicated speaker diarization block, it is often desired to evaluate the performance without the diarization block in order

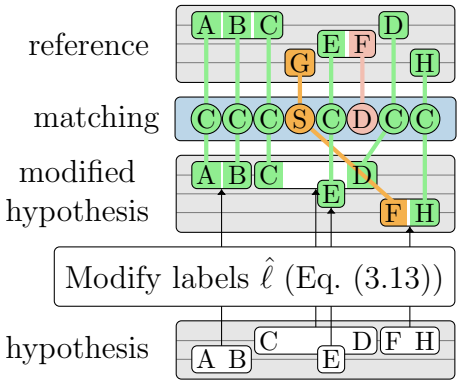


Figure 3.10: DI-cpWER = $\frac{0+1+1}{8}$ = 25.0%. The hypothesis stream labels are modified with Eq. (3.13) to minimize the error rate. The global order of segments is preserved. *(Taken from [OC8])*

to obtain a potential upper bound, i.e., the performance that could be achieved with perfect speaker attributions. The DI-cpWER [36, OC8] was specifically designed for this purpose. It is equivalent to the ORC-WER with reference and hypothesis swapped:

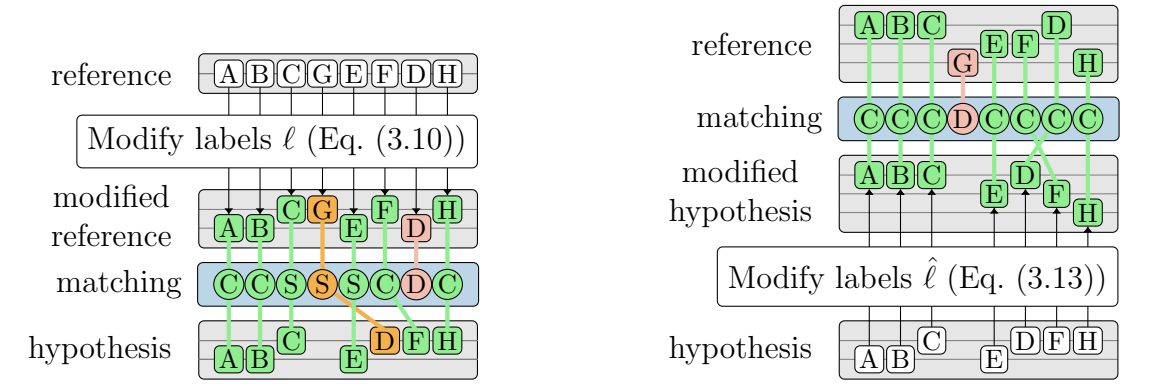
$$\text{lev}^{(\text{DI-cp})}(\mathbf{t}^{(\text{spk})}_{1,\dots,K}, \hat{\mathbf{t}}_{1,\dots,S}) = \text{lev}^{(\text{ORC})}(\hat{\mathbf{t}}_{1,\dots,S}, \mathbf{t}^{(\text{spk})}_{1,\dots,K}). \tag{3.13}$$

Instead of modifying the reference labels, it modifies the hypothesis labels such that the WER is minimized, effectively performing an oracle speaker attribution w.r.t. the WER. Fig. 3.10 shows how the DI-cpWER modifies the hypothesis labels instead of the reference labels, compared to the ORC-WER. The DI-cpWER is here smaller than the ORC-WER because the example contains utterance split errors, but no combined utterance merge and speaker attribution error which would be penalized by the DI-cpWER. The idea for the DI-cpWER was proposed in [36] and the analysis and efficient algorithms for its computations were developed in [OC8].

Relation to ORC-WER The algorithmic considerations are the same as for the ORC-WER with swapped reference and hypothesis. Instead of utterance splits, the DI-cpWER penalizes utterance merges and is unaffected by utterance splits. ORC-WER is close, and often equal, to DI-cpWER when no utterance-merge and no utterance-split errors are present. Since the DI-cpWER increases with utterance-merge errors and the ORC-WER increases with utterance-split errors, their difference is a rough indicator for which kind of segmentation error is more prominent. But, no definitive conclusions about the actual frequency of these errors can be drawn from it.

Fooling the DI-cpWER The DI-cpWER prefers smaller segments, so can be improved by only modifying the segmentation and splitting all segments down to a word level. This effectively removes the penalty for utterance merges and thus improves the score. Due to this issue, the metric should never be used for system ranking. It should only by applied for system analysis where the designer of the system is aware of this issue and not interested in fooling the metric, but in actually improving the system.

System analysis The DI-cpWER is directly compatible with the cpWER, i.e., if no speaker attribution errors are present, then DI-cpWER equals cpWER. This is validated



(a) wl-ORC-WER = $\frac{0+1+1}{8}$ = 25.0%. The alignment is the same as ORC-WER (Fig. 3.8), but all segments are broken down into words before matching.

(b) wl-DI-cpWER = $\frac{0+1+0}{8}$ = 12.5%. The alignment is the same as DI-cpWER (Fig. 3.10), but all segments are broken down into words before matching.

Figure 3.11: Visualization of word-level matchings. Every word is considered its own segment. Segments present in the reference or hypothesis are ignored. *(Taken from [OC8])*

experimentally later in Section 3.5. Because of this property, the difference between cpWER and DI-cpWER measures the impact of speaker attribution errors on the system performance. The DI-cpWER alone shows the potential performance of the system if the speaker attribution was perfect. This analysis helps to find the most promising directions for system improvement.

Estimating speaker confusions from speaker-agnostic alignments Instead of just looking at the difference between DI-cpWER and cpWER, the number of wrongly recognized speaker labels can be predicted by counting how many times the speaker label changes in the alignment of a speaker-agnostic WER (such as DI-cpWER or ORC-WER). This approach was chosen in the SA-WER [64], but it can be applied to any speaker-agnostic alignment algorithm. However, it is not exact and not compatible to the cpWER, or any other speaker-attributed WER, i.e., the cpWER cannot be computed from the speaker-agnostic WER and the number of confusions. Because of this, we here refrain from this approach. When these statistics are computed on a word level, as in the SA-WER, the number of speaker confusions is typically over-estimated.

3.2.5 Other WER Definitions for Meeting Scenarios

Other approaches exist in the literature for computing a WER in the meeting scenario. These approaches, however, come with significant drawbacks, as detailed in the following.

3.2.5.1 Speaker-agnostic Analysis With Word-Level Assignments

The ORC-WER, MIMO-WER and DI-cpWER solve the assignment problem on a segment level, so that they are only insensitive to speaker attribution errors and certain types of segmentation errors (ORC-WER and MIMO-WER are is insensitive to utterance-merge errors, DI-cpWER is insensitive to utterance-split errors), but other kinds of segmentation errors are penalized. To make the metrics invariant to all segmentation errors, one has to drop the

utterance-consistency constraint and solve the assignment problem on a word level. Applied to the WERs defined above, this leads to the word-level ORC-WER (wl-ORC-WER), word-level MIMO-WER (wl-MIMO-WER) and word-level DI-cpWER (wl-DI-cpWER). Fig. 3.11 shows the matching for the word-level ORC-WER (Fig. 3.11a) and word-level DI-cpWER (Fig. 3.11b). The number of errors is generally smaller, but the count is potentially less informative than the segment-level matchings.

Computing the matching on a word level is computationally more challenging than the segment-level matching due to the increased number of possible assignments. It is thus often not feasible to compute.

Word-level WERs are over-optimistic Since there is no constraint on the utterance consistency, words can be freely moved across by the assignment solver to minimize the WER. This can again lead to implausible constellations where words are distributed across streams to trade substitutions for insertions and deletions, even if the words may not be uttered by the same speaker.

3.2.5.2 asclite

The asclite tool, already mentioned above for its time-constraint, solves the assignment problem on a word level. Its core algorithm is similar to the word-level MIMO-WER, although asclite can align graphs and alternative transcripts instead of just word sequences. As discussed above, this word-level alignment is over-optimistic.

3.2.5.3 Diarization-Aware WER (DA-WER)

The Diarization-Aware WER (DA-WER) was proposed for the 7th CHiME challenge to measure both diarization and speech recognition performance. It is similar to the cpWER, but the permutation is chosen to minimize the DER instead of the WER:

$$\boldsymbol{\pi}^{(\mathrm{DER})} = \underset{\boldsymbol{\pi} \in \mathcal{P}(K)}{\arg\min} \sum_{k=1}^{K} \mathrm{DER}\left(\mathbf{t}_{\pi_k}^{(\mathrm{spk})}, \hat{\mathbf{t}}_{k}^{(\mathrm{spk})}\right). \tag{3.14}$$

By this, the challenge participants were forced to provide both transcription and diarization estimates.

The permutation $\boldsymbol{\pi}^{(\mathrm{DER})}$ is often equal to the permutation $\boldsymbol{\pi}^{(\mathrm{cp})}$ found by the cpWER, but it is not guaranteed so, even if the diarization estimate is good. Especially when speech activity is overall high with many overlaps, the permutation $\boldsymbol{\pi}^{(\mathrm{DER})}$ can be sub-optimal for WER and lead to a low score.

The DA-WER may also be over-optimistic for the joint evaluation of diarization and speech recognition. When both tasks are solved independently and are wrongly combined, i.e., the speech content is correct, but it is attributed to the wrong segment, the DA-WER can be low although the alignment is implausible[9]. Such an example is discussed in Section 3.5.8 and displayed in Fig. 3.20c. Instead, the cpWER or tcpWER should be used.

[9] This was observed in one submission to the challenge.

3.3 Efficient Computation of Meeting-level WERs

The last section presented the definitions of the WERs. The formulations were chosen to highlight intuitively what is computed, but they do not describe an efficient way for obtaining the result. In this section, we discuss efficient algorithms for computing the meeting-level WERs.

3.3.1 cpWER

The assignment problem of the cpWER and tcpWER is conceptually equivalent to the permutation problem in the uPIT approach for training neural networks for speech separation described in Section 2.6.1. The naive approach for computing $\mathrm{lev}^{(\mathrm{cp})}$ computes the Levenshtein distances for $K!$ many permutations, in total $K! \cdot K$ distances have to be computed. This is feasible for low numbers of speakers (e.g., $K \leqslant 3$), but quickly becomes infeasible for larger numbers of speakers. The permutation $\boldsymbol{\pi}^{(\mathrm{cp})}$ can be found by applying the Hungarian algorithm [83, 84] to the score matrix $\mathbf{M} = [\mathrm{lev}(\mathbf{t}_i^{(\mathrm{spk})}, \hat{\mathbf{t}}_j^{(\mathrm{spk})})]_{i,j}$.[10]

3.3.2 Exact Algorithm for ORC-WER

The ORC-WER can be reformulated from its brute-force approach in Eq. (3.10) to a recursive approach leading to a dynamic programming realization, which significantly reduces the computational effort for long inputs.

3.3.2.1 Word-level Algorithm

First, the simpler word-level case is introduced, where every reference segment's transcript $\mathbf{t}$ consists of a single word $\mathbf{t}_u = t_u$, the number of words in the reference is equal to $|\mathcal{U}|$ and the word-level ORC Levenshtein distance can be written as

$$\mathrm{lev}^{(\text{wl-ORC})}\left(t_{1,\ldots,|\mathcal{U}|}, \hat{\mathbf{t}}^{(\mathrm{str})}_{1,\ldots,S}\right) = \min_{1 \leqslant \tilde{\ell}_1,\ldots,\tilde{\ell}_{|\mathcal{U}|} \leqslant S} \sum_{s=1}^{S} \mathrm{lev}\left(\mathbf{t}^{(\mathrm{cat})}_{\ell,s}, \hat{\mathbf{t}}^{(\mathrm{str})}_{s}\right). \tag{3.15}$$

The hypothesis transcripts are still considered as word sequences. This corresponds to dropping the utterance-consistency constraint from the ORC-WER. Starting with Eq. (3.15), the Levenshtein distance corresponding to the stream s' that the last word ($u = |\mathcal{U}|$) is assigned to is split off the sum:

$$\mathrm{lev}^{(\text{wl-ORC})}\left(t_{1,\ldots,|\mathcal{U}|}, \hat{\mathbf{t}}^{(\mathrm{str})}_{1,\ldots,S}\right) = \min_{\substack{1 \leqslant \tilde{\ell}_1,\ldots,\tilde{\ell}_{|\mathcal{U}|-1} \leqslant S \\ 1 \leqslant s' \leqslant S}} \left(\mathrm{lev}\left(\mathbf{t}^{(\mathrm{cat})}_{\ell,s'}, \hat{\mathbf{t}}^{(\mathrm{str})}_{s'}\right) + \sum_{\substack{s=1 \\ s \neq s'}}^{S} \mathrm{lev}\left(\mathbf{t}^{(\mathrm{cat})}_{\ell,s}, \hat{\mathbf{t}}^{(\mathrm{str})}_{s}\right) \right). \tag{3.16}$$

[10] This notation will be introduced for the training of a speech separation system in more detail later in Chapter 5.

The former summand can be expanded with one update of the Wagner-Fischer algorithm from Eq. (2.17):

$$\operatorname{lev}\left(\mathbf{t}_{\ell,s'}^{(\text{cat})}, \hat{\mathbf{t}}_{s'}^{(\text{str})}\right) = \min \begin{cases} \operatorname{lev}\left(\operatorname{head}(\mathbf{t}_{\ell,s'}^{(\text{cat})}), \hat{\mathbf{t}}_{s'}^{(\text{str})}\right) + C_{\text{D}}, \\ \operatorname{lev}\left(\mathbf{t}_{\ell,s'}^{(\text{cat})}, \operatorname{head}(\hat{\mathbf{t}}_{s'}^{(\text{str})})\right) + C_{\text{I}}, \\ \operatorname{lev}\left(\operatorname{head}(\mathbf{t}_{\ell,s'}^{(\text{cat})}), \operatorname{head}(\hat{\mathbf{t}}_{s'}^{(\text{str})})\right) + C_{\text{C/S}}(t_{|\mathcal{U}|}, \operatorname{tail}(\hat{\mathbf{t}}_{s'}^{(\text{str})})). \end{cases} \tag{3.17}$$

We can now plug in Eq. (3.17) into Eq. (3.16) and change the order of the min operations to obtain

$$\operatorname{lev}^{(\text{wl-ORC})} = \min \begin{cases} \min\limits_{\substack{1 \leqslant \tilde{\ell}_1, \ldots, \tilde{\ell}_{|\mathcal{U}|-1} \leqslant S \\ 1 \leqslant s' \leqslant S}} \left(\operatorname{lev}\left(\operatorname{head}(\mathbf{t}_{\ell,s'}^{(\text{cat})}), \hat{\mathbf{t}}_{s'}^{(\text{str})}\right) + \sum\limits_{\substack{s=1 \\ s \neq s'}}^{S} \operatorname{lev}\left(\mathbf{t}_{\ell,s}^{(\text{cat})}, \hat{\mathbf{t}}_{s}^{(\text{str})}\right) \right) + C_{\text{D}}, \\ \min\limits_{\substack{1 \leqslant \tilde{\ell}_1, \ldots, \tilde{\ell}_{|\mathcal{U}|-1} \leqslant S \\ 1 \leqslant s' \leqslant S}} \left(\operatorname{lev}\left(\mathbf{t}_{\ell,s'}^{(\text{cat})}, \operatorname{head}(\hat{\mathbf{t}}_{s'}^{(\text{str})})\right) + \sum\limits_{\substack{s=1 \\ s \neq s'}}^{S} \operatorname{lev}\left(\mathbf{t}_{\ell,s}^{(\text{cat})}, \hat{\mathbf{t}}_{s}^{(\text{str})}\right) \right) + C_{\text{I}}, \\ \min\limits_{\substack{1 \leqslant \tilde{\ell}_1, \ldots, \tilde{\ell}_{|\mathcal{U}|-1} \leqslant S \\ 1 \leqslant s' \leqslant S}} \left(\operatorname{lev}\left(\operatorname{head}(\mathbf{t}_{\ell,s'}^{(\text{cat})}), \operatorname{head}(\hat{\mathbf{t}}_{s'}^{(\text{str})})\right) + \sum\limits_{\substack{s=1 \\ s \neq s'}}^{S} \operatorname{lev}\left(\mathbf{t}_{\ell,s}^{(\text{cat})}, \hat{\mathbf{t}}_{s}^{(\text{str})}\right) \right) \\ \qquad + C_{\text{C/S}}(t_{|\mathcal{U}|}, \operatorname{tail}(\hat{\mathbf{t}}_{s'}^{(\text{str})})). \end{cases} \tag{3.18}$$

The first term in the outer minimum can be simplified to

$$\min_{1 \leqslant \tilde{\ell}_1, \ldots, \tilde{\ell}_{|\mathcal{U}|-1} \leqslant S} \sum_{s=1}^{S} \operatorname{lev}\left(\mathbf{t}_{\ell,s}^{(\text{cat})}, \hat{\mathbf{t}}_{s}^{(\text{str})}\right) + C_{\text{D}} = \operatorname{lev}^{(\text{wl-ORC})}(t_{1,\ldots,|\mathcal{U}|-1}, \hat{\mathbf{t}}_{1,\ldots,S}^{(\text{str})}) + C_{\text{D}}. \tag{3.19}$$

Both summands are independent of s' because the $|\mathcal{U}|$-th word is removed by the head in the first summand and $\mathbf{t}_{\ell,s}^{(\text{cat})}$ in the sum is independent of $\ell_{|\mathcal{U}|}$ if $s \neq s'$, which is given by the sum bounds. The minimum over s' can thus be dropped and the terms can be combined into a single sum. The other two cases can be simplified in a similar manner. We finally obtain:

$$\operatorname{lev}^{(\text{wl-ORC})}(t_{1,\ldots,|\mathcal{U}|}, \hat{\mathbf{t}}_{1,\ldots,S}^{(\text{str})}) = \min \begin{cases} \operatorname{lev}^{(\text{wl-ORC})}(t_{1,\ldots,|\mathcal{U}|-1}, \hat{\mathbf{t}}_{1,\ldots,S}^{(\text{str})}) + C_{\text{D}}, \\ \min\limits_{1 \leqslant s \leqslant S} \operatorname{lev}^{(\text{wl-ORC})}(t_{1,\ldots,|\mathcal{U}|}, \hat{\mathbf{t}}_{1}^{(\text{str})}, \ldots, \operatorname{head}(\hat{\mathbf{t}}_{s}^{(\text{str})}), \ldots, \hat{\mathbf{t}}_{S}^{(\text{str})}) \\ \qquad + C_{\text{I}}, \\ \min\limits_{1 \leqslant s \leqslant S} \operatorname{lev}^{(\text{wl-ORC})}(t_{1,\ldots,|\mathcal{U}|-1}, \hat{\mathbf{t}}_{1}^{(\text{str})}, \ldots, \operatorname{head}(\hat{\mathbf{t}}_{s}^{(\text{str})}), \ldots, \hat{\mathbf{t}}_{S}^{(\text{str})}) \\ \qquad + C_{\text{C/S}}(t_{|\mathcal{U}|}, \operatorname{tail}(\hat{\mathbf{t}}_{s}^{(\text{str})})). \end{cases} \tag{3.20}$$

This constitutes a recursive formulation of the word-level ORC Levenshtein distance that is very similar to the standard Levenshtein distance defined in Eq. (2.17), but it operates on $S + 1$ word sequences instead of two. Each application of Eq. (3.20) removes words from the reference and hypothesis sequences until empty sequences are left. The recursion terminates with $\text{lev}^{(\text{wl-ORC})}(\varnothing, \varnothing, ...) = 0$ when all sequences are empty, and recursions where the head of an empty sequence $\text{head}(\varnothing)$ would be required are skipped.

Similar to the standard Levenshtein distance update in Eq. (2.17), we also have a minimum operation over the three edit operations deletion, insertion and substitution or correct match. The first case, the deletion, is computed from the distance with the last reference word removed. The second case, the insertion, removes one word from a hypothesis word sequence. Since we have multiple hypothesis sequences, and an insertion operation can only ever insert one word at a time, we take the minimum over all of streams where a word could be inserted. The third case, the substitution or correct match, removes one word from the reference and one word from a hypothesis, and we again have to take the minimum over all streams from which a word is removed.

A shorter notation similar to the Levenshtein matrix in Eq. (2.19) is achieved by collecting the intermediate ORC Levenshtein distance values in a tensor $\mathbf{L}^{(\text{wl-ORC})}$ of $S + 1$ dimensions, where $\mathbf{L}^{(\text{wl-ORC})} \in \mathbb{N}^{(|\mathcal{U}|+1)\times(\dim(\hat{\mathbf{t}}_1^{(\text{str})})+1)\times ...\times(\dim(\hat{\mathbf{t}}_S^{(\text{str})})+1)}$. Each dimension represents one word sequence, where the first dimension represents the reference words $t_1, ..., t_{|\mathcal{U}|}$ and the latter S dimensions each represent one of the S grouped hypothesis transcripts $\hat{\mathbf{t}}_s^{(\text{str})}$. The position along each dimension, $0 \leqslant i \leqslant |\mathcal{U}|$ for the reference and $0 \leqslant h_s \leqslant \dim(\hat{\mathbf{t}}_s^{(\text{str})})$ for the hypothesis transcripts, determines the length of the (prefix) sub-sequence for which the wl-ORC Levenshtein distance is computed, so that the elements are defined as $\mathbf{L}^{(\text{wl-ORC})} = [l_{i,\mathbf{h}}^{(\text{wl-ORC})}]_{i,\mathbf{h}} = [\text{lev}^{(\text{wl-ORC})}(t_1, ..., t_i, \hat{\mathbf{t}}_{\leqslant h_1}, ..., \hat{\mathbf{t}}_{\leqslant h_S})]_{i,\mathbf{h}}$. The indices that correspond to the hypothesis transcripts are collected in a multi-index $\mathbf{h} = (h_1, ..., h_S)$, so that they can be treated as a single object. Using this matrix notation, Eq. (3.20) can be written as

$$l_{i,\mathbf{h}}^{(\text{wl-ORC})} = \min \begin{cases} l_{i-1,\mathbf{h}}^{(\text{wl-ORC})} + C_{\text{D}} \\ \min_{1\leqslant s\leqslant S} l_{i,\mathbf{h}-\mathbf{e}_s^{(S)}}^{(\text{wl-ORC})} + C_{\text{I}} \\ \min_{1\leqslant s\leqslant S} l_{i-1,\mathbf{h}-\mathbf{e}_s^{(S)}}^{(\text{wl-ORC})} + C_{\text{C/S}}. \end{cases} \tag{3.21}$$

The multi-index $\mathbf{e}_i^{(S)}$ is a one-hot index of length S with a 1 at index i and 0 elsewhere. The subtraction of $\mathbf{e}_i^{(S)}$ from a multi-index $\mathbf{h}$ is equal to reducing the i-th element in $\mathbf{h}$ by one: $\mathbf{h} - \mathbf{e}_i^{(S)} = (h_1, ..., h_i - 1, ..., h_S)$. The tensor is initialized with $l_{0,\mathbf{h}}^{(\text{wl-ORC})} = \sum_{s=1}^{S} h_s$ and $l_{i,(0,...,0)}^{(\text{wl-ORC})} = i$, and we set $l_{i,(0,...-1,...,0)}^{(\text{wl-ORC})} = \infty$ for invalid negative hypothesis indices, so that they are ignored in the minimum operation. This equation can directly be translated into an efficient implementation.

3.3.2.2 Utterance-level Algorithm

The word-level algorithm is modified to the full ORC Levenshtein distance algorithm by inserting utterance change tokens [ct] between utterances into the reference word sequence and ensuring that words are matched consistently to a single stream between two [ct] tokens. Since each reference utterance must be matched to a single hypothesis transcript completely, the minimum operations in the lower two cases of Eq. (3.21), which allow subsequent insertions,

substitutions and correct matches across multiple hypothesis transcripts, are removed. The hypothesis index s is instead re-introduced as a tensor dimension to denote the stream index that the current utterance is exclusively assigned to. The minimum is then computed only after matching full utterances, which appears at the change tokens. The update equation for the utterance-level ORC Levenshtein distance becomes

$$l^{(\mathrm{ORC})}_{i,s,\mathbf{h}} = \begin{cases} \min\limits_{1 \leqslant \tilde{s} \leqslant S} l^{(\mathrm{ORC})}_{i-1,\tilde{s},\mathbf{h}}, & \text{if } t_i = [\mathrm{ct}], \\ \min \begin{cases} \mathrm{lev}^{(\mathrm{ORC})}_{i-1,s,\mathbf{h}} + C_{\mathrm{D}}, \\ \mathrm{lev}^{(\mathrm{ORC})}_{i,s,\mathbf{h}-\mathbf{e}^{(S)}_s} + C_{\mathrm{I}}, \\ \mathrm{lev}^{(\mathrm{ORC})}_{i-1,s,\mathbf{h}-\mathbf{e}^{(S)}_s} + C_{\mathrm{C/S}}, \end{cases} & \text{otherwise.} \end{cases} \tag{3.22}$$

The total Levenshtein distance is then

$$\mathrm{lev}^{(\mathrm{ORC})} = \min_{1 \leqslant s \leqslant S} l^{(\mathrm{ORC})}_{|\mathcal{U}|,s,(\dim(\hat{\mathbf{t}}^{(\mathrm{str})}_1),\ldots,\dim(\hat{\mathbf{t}}^{(\mathrm{str})}_S))}. \tag{3.23}$$

The minimum in the lower case where $t_i \neq [\mathrm{ct}]$ resembles the standard Levenshtein distance Eq. (2.19) across a slice of $\mathbf{L}^{(\mathrm{ORC})}$, for a reference utterance compared to the full stream s. This allows for using sophisticated Levenshtein distance algorithms, such as the one from [71], also for the multi-dimensional Levenshtein distance. Eq. (3.22) contains fewer transitions between streams as Eq. (3.21), but the additional index s and thus one more dimension in the tensor $\mathbf{L}^{(\mathrm{ORC})}$.

Eq. (3.22) falls back to Eq. (3.21) when a change token is inserted after every reference word. In that case, the upper and lower minimum operations are executed alternately, and the deletion case is redundant across streams. Combining both minimum operations into a single minimum operation and removing the redundant deletion cases leads again to Eq. (3.21).

3.3.2.3 Pruning for the Time-constrained ORC-WER

The pruning that was applied to the time-constrained (standard) Levenshtein distance in Section 3.1.1.1 can in a similar way be applied to the time-constrained ORC-WER. For the lower part of Eq. (3.22) that resembles the standard Levenshtein distance, the pruning is identical to the standard time-constrained Levenshtein distance. The minimum in the first case only has to be computed for indices s where the corresponding hypothesis transcript $\hat{\mathbf{t}}^{(\mathrm{str})}_s$ contains words that overlap with the current reference utterance. For regions where no words in the hypothesis overlap with the current reference utterance, the update and recursion can be skipped entirely and be replaced by a simple addition of the number of non-overlapping words. For many realistic scenarios, this pruning can significantly speed up the computation and enable the use of the Time-Constrained ORC-WER (tcORC-WER) where the ORC-WER is not feasible. This is shown empirically later in the experiments in Section 3.5.7.

3.3.2.4 Computational Complexity

Brute-force algorithm The brute-force algorithm from Eq. (3.10) computes the Levenshtein distance for all streams for in total $S^{|\mathcal{U}|}$ assignments. Given the assignment labels

$\tilde{\ell}$, each assignment requires $\sum_{s=1}^{S} \dim(\hat{\mathbf{t}}_s) \cdot \sum_{i=\tilde{\ell}_i} \dim(\mathbf{t}_i)$ computations of the Levenshtein distance, a rough upper bound of which is given by $\mathcal{O}(S \cdot |\mathcal{U}| \max_s\{\dim(\hat{\mathbf{t}}_s)\} \cdot \max_u \dim(\mathbf{t}_u))$. This results in an overall complexity of $\mathcal{O}(S^{(|\mathcal{U}|+1)} \cdot |\mathcal{U}| \cdot \max_s\{\dim(\hat{\mathbf{t}}_s)\} \cdot \max_u \dim(\mathbf{t}_u))$, which is exponential in the number of reference utterances. It is only feasible to compute for short recordings, which caused the authors of [38, 77] to ignore recordings with many utterances.[11]

A trivial optimization may seem to be to cut the transcriptions at points where no speaker is active and solve the assignment problem for each of the obtained regions individually. This approach works for the time-constrained ORC-WER (tcORC-WER) when the collar regions are considered also as speech activity. The pruning described for the time-constrained ORC-WER in Section 3.3.2.3 for the dynamic programming algorithm does this optimization implicitly and on a much finer scale, i.e., for each utterance instead of a connected set of utterances. For the non-time-constrained ORC-WER, it does not yield the optimal solution because words can match across arbitrary distances, i.e., across the boundaries of these speech activity regions, without the temporal constraint. This non-optimal optimization heuristic was used in the `SCTK` toolkit.

Dynamic programming algorithm The tensor $\mathbf{L}^{(\text{ORC})}$ in Eq. (3.22) is of size $(|\mathcal{U}| + \sum_{u\in\mathcal{U}} \dim(\mathbf{t}_u)) \cdot S \cdot (\dim(\hat{\mathbf{t}}_1^{(\text{str})}) + 1) \cdot ... \cdot (\dim(\hat{\mathbf{t}}_S^{(\text{str})}) + 1)$. The first dimension contains all words in the reference and the change tokens [ct]. The second dimension represents the hypothesis stream transcript to which an utterance is assigned. The remaining dimensions represent the sub-sequences of the hypothesis transcripts for which partial ORC Levenshtein distances are computed. Each cell update requires a minimum operation over S other cells (the upper minimum in Eq. (3.22) which happens $\sum_{u\in\mathcal{U}} \dim(\mathbf{t}_u)$ times) or three other cells (the lower minimum in Eq. (3.22) which happens $\dim(\mathbf{t})$ times). The overall complexity is thus $\mathcal{O}((S|\mathcal{U}| + 3 \cdot \sum_{u\in\mathcal{U}} \dim(\mathbf{t}_u)) S \prod_{s=1}^{S} \dim(\hat{\mathbf{t}}_s^{(\text{spk})}))$.

For the word-level algorithm, where $|\mathcal{U}| = \sum_{u\in\mathcal{U}} \dim(\mathbf{t}_u)$, the deletion case can be pulled out of the lower minimum, and the second dimension of $\mathbf{L}^{(\text{ORC})}$ collapses from S to 1 after eliminating redundant cases. This leads to a complexity of $\mathcal{O}((2S+1) \sum_{u\in\mathcal{U}} \dim(\mathbf{t}_u) \prod_{s=1}^{S} \dim(\hat{\mathbf{t}}_s^{(\text{spk})}))$, which is equivalent to the bound derived in [OC7].

The computations can be pruned for the tcORC-WER, as roughly outlined in Section 3.3.2.3. The effect of the pruning depends highly on the number of hypothesis words that each reference utterance overlaps with. For cases where only words on a single system output transcription overlap with a reference utterance, all other system outputs can be ignored, which causes a significant speedup. If the collar is chosen large enough so that almost all words overlap with all other words, the pruning logic adds additional complexity and slows down the computation. In practical examples, the pruning leads to a significant speedup, as shown in Section 3.5.7.

Theoretical bound The ORC assignment problem is NP-hard. To show this, we show that the problem is harder than the sequence or string MERGE problem, which is proven to be NP-complete for arbitrary numbers of sequences [81]. The sequence MERGE problem was already briefly introduced for the MIMO-WER in Section 3.2.3 as determining whether a sequence $\mathbf{t}^{(\text{MERGE})}$ is a merge of multiple sequences $\mathbf{t}_1$ to $\mathbf{t}_N$, notated as $\mathbf{t}^{(\text{MERGE})} \in \text{MERGE}(\mathbf{t}_1, ..., \mathbf{t}_N)$. This is similar to the ORC problem, and the MERGE problem can in fact be answered by

[11] In [77] the number of utterances was limited to 12 using the LibriCSS-segments dataset, and in [38] it was limited to 23 for LibriCSS-raw.

checking $\text{lev}^{(\text{ORC})}(\mathbf{t}^{(\text{MERGE})}, (\mathbf{t}_1, ..., \mathbf{t}_N)) = 0$. The MERGE recognition problem can be solved in $\mathcal{O}\left(\dim(\mathbf{t}^{(\text{MERGE})}) \prod_i (\dim(\mathbf{t}_i) + 1)\right)$ [81], which belongs to the same complexity class as the ORC algorithm. From this follows that ORC is at least as hard as MERGE, and since ORC is not a decision problem, it must be NP-hard with a complexity exponential in the number of streams S. It can additionally be seen that the (word-level) ORC alignment finds the string merge (or one of them, if there are multiple ambiguous merges) that is closest to the reference in terms of the Levenshtein distance.

3.3.3 Greedy Algorithm for Approximating ORC-WER

For large numbers of speakers, the exact algorithm for computing the ORC-WER can become infeasible. A simple greedy approach can approximate the ORC-WER with high precision and low complexity [36, OC8][12]. The naive algorithm is outlined in Algorithm 1. It iteratively modifies the reference speaker labels ℓ if the modification reduces the error rate. In temporal order, it checks for every reference utterance if assigning it to a different stream would improve the Levenshtein distance (and with it the error rate). If it does, the new label is assigned. This process is repeated until no further improvement can be made by modifying the label of a single utterance at once. It is guaranteed to terminate because the update of a single utterance can only ever improve the error rate. But note that this algorithm is not guaranteed to find the optimal assignment. It, nevertheless, often finds a good approximation of the optimal assignment, as can be seen in Section 3.5.6.

Initialization The outlined algorithm only modifies the labels ℓ until it finds a good solution, but it has not yet been stated how ℓ is initialized. If speaker labels are available, they can be used to initialize ℓ, e.g., with the assginment $\ell = \ell^{(\text{cp})}$ obtained from the cpWER algorithm. If speaker labels are not available, the labels can be initialized randomly or in a round-robin manner. Assigning all utterances to a single stream is not recommended because the initial iterations will not perform plausible assignments, but merely balance the number of words across streams.

Handling edge cases Using just this simple approach, i.e., only a single pass of GreedyUpdateLabels in Algorithm 1, can fail to solve cases where two utterances would have to be swapped to improve the error rate while changing only one of them increases the error rate. In such cases, the solution can be improved by first running the algorithm with a substitution cost of $C_\text{S} = C_\text{I} + C_\text{D} = 2$, which allows for swapping two utterances, and then optimizing with a substitution cost of $C_\text{S} = 1$. Using $C_\text{S} = 2$ enables the algorithm to trade a substitution for a deletion and an insertion, which reduces the costs for changing only one label in these cases.

The complexity of the greedy algorithm outlined in Algorithm 1 can be improved significantly by re-using intermediate computations [OC8]. The optimized algorithm is outlined in Algorithm 2 and described in the following.

[12] The naive greedy algorithm was initially introduced for the DI-cpWER in [36], but is here equivalently described for the ORC-WER.

Algorithm 1 Naive greedy approximation to the ORC Levenshtein distance $\text{lev}^{(\text{ORC})}$

Input: Reference and hypothesis transcripts $\mathbf{t}_{1,...,|\mathcal{U}|}$, $\hat{\mathbf{t}}^{(\text{str})}_{1,...,S}$ and initial assigned labels ℓ
Output: Modified reference speaker labels ℓ and the corresponding Levenshtein distance

function LevDist($\mathbf{t}$, $\hat{\mathbf{t}}$, ℓ, C_S)
  ▷ *Compute the sum of the Levenshtein distances across all streams. Similar to Eq. (3.8) but without the minimum operation.* ◁
  return $\sum_{s=1}^{S} \text{lev}\left(\mathbf{t}^{(\text{cat})}_{\ell,s}, \hat{\mathbf{t}}^{(\text{str})}_{s}, C_\text{S}\right)$,
function greedyUpdateLabels($\mathbf{t}$, $\hat{\mathbf{t}}$, ℓ, C_S)
  while not converged **do**
    for $u \in \mathcal{U}$ **do**
      for $s \in \{1, ..., S\}$ **do**
        ▷ *Compute Levenshtein distance with current reference labels* ◁
        $d \leftarrow$ LevDist($\mathbf{t}$, $\hat{\mathbf{t}}$, ℓ,C_S)
        ▷ *Compute Levenshtein distance with updated reference labels (put utterance u on stream s)* ◁
        $\ell' \leftarrow [\ell, ..., \ell_{u-1}, s, \ell_{u+1}, ..., \ell_{|\mathcal{U}|}]$
        $d' \leftarrow$ LevDist($\mathbf{t}, \hat{\mathbf{t}}, \ell', C_\text{S}$)
        ▷ *Update the labels if the new distance is smaller (better)* ◁
        if $d' < d$ **then**
          $\ell \leftarrow \ell'$
  return ℓ
▷ *Solve "swapping" problem by running one update with a substitution cost of 2* ◁
$\ell \leftarrow$ greedyUpdateLabels($\mathbf{t}, \hat{\mathbf{t}}, \ell, 2$)
$\ell \leftarrow$ greedyUpdateLabels($\mathbf{t}, \hat{\mathbf{t}}, \ell, 1$)
▷ *Return the corrected labels and the Levenshtein distance* ◁
return ℓ, LevDist($\mathbf{t}$, $\hat{\mathbf{t}}$, ℓ, 1)

Reusing computations across iterations It can first be realized that computations of d and d' in Algorithm 1 can be re-used across iterations and that by moving an utterance from one stream to another stream, only the distances for these streams change while the distances for all other streams are not modified. Let us split the computation of d into the Levenshtein distances of the individual streams. Let $l^+_{su} = \text{lev}([..., \mathbf{t}_{u'}, ... : \ell_{u'} = s \vee u' = u], \hat{\mathbf{t}}^{(\text{spk})}_s)$ be the Levenshtein distance for stream s given the current labels ℓ and additionally with utterance u placed on the stream (if it is not already placed there by ℓ). Let accordingly $l^-_{su} = \text{lev}([\mathbf{t}_{u'}, ... : \ell_{u'} = s \wedge u' \neq u], \hat{\mathbf{t}}^{(\text{spk})}_s)$ be the Levenshtein distance for stream s given the current labels ℓ but without utterance u on the stream.

The inner for loop, which updates ℓ_u, can be written as an argmin operation

$$\ell_u = \arg\min_{1 \leqslant s \leqslant S} (l^+_{su} + \sum_{\tilde{s} \neq s} l^-_{\tilde{s}u}) \tag{3.24}$$

$$= \arg\min_{1 \leqslant s \leqslant S} (l^+_{su} - l^-_{su}). \tag{3.25}$$

So, to determine the best stream for the utterance u, we only need the distances of two streams for every candidate s instead of the full distance d. Additionally, one of the two distances l^+_{su} or l^-_{su} is already known from an earlier iteration. This reduces the number of

times that the Levenshtein distance for a complete stream has to be computed for every utterance and iteration from $2S^2$ to S.

Reusing partial Levenshtein matrices The second optimization is based on the observation that in the Levenshtein matrix $\mathbf{L}$ for every j, there exists (at least) one i such that $l_{i,j}$ lies on an optimal path (see Section 2.8.6.1 for an explanation of optimal paths). For these positions in the matrix, it holds:

$$\text{lev}(\mathbf{t}, \hat{\mathbf{t}}) = \text{lev}(\mathbf{t}_{\leqslant i}, \hat{\mathbf{t}}_{\leqslant j}) + \text{lev}(\mathbf{t}_{>i}, \hat{\mathbf{t}}_{>j}), \forall (i,j) \text{ on an optimal path.} \tag{3.26}$$

The notation $(\cdot)_{>i}$ here denotes the sub-sequence starting from the i-th element until the end, similar to how $(\cdot)_{\leqslant i}$ is the sub-sequence up to the i-th element. The values $\text{lev}(\mathbf{t}_{>i}, \hat{\mathbf{t}}_{>j})$ are computed and stored in a matrix $\mathbf{L}^{(\text{bw})} = [\text{lev}(\mathbf{t}_{>i}, \hat{\mathbf{t}}_{>j})]_{i,j}$ in the same way that $\text{lev}(\mathbf{t}_{\leqslant i}, \hat{\mathbf{t}}_{\leqslant j}) = l_{i,j}$ are stored in $\mathbf{L}$. Because the Levenshtein distance is symmetric, i.e., $\text{lev}(\mathbf{t}, \hat{\mathbf{t}}) = \text{lev}(\hat{\mathbf{t}}, \mathbf{t})$, the matrix $\mathbf{L}^{(\text{bw})}$ is obtained by reversing both sequences, computing the Levenshtein matrix according to Eq. (2.19), and then reversing the order of the rows and columns:

$$\mathbf{L}^{(\text{bw})} = \text{rev}(\mathbf{L}(\text{rev}(\mathbf{t}), \text{rev}(\hat{\mathbf{t}}))). \tag{3.27}$$

The function rev reverses the order of a sequence ($\text{rev}([t_1, ..., t_N]) = [t_N, ..., t_1]$) and reverses the order of the rows and columns of a matrix (For the matrix $\mathbf{L} = [l_{i,j}]_{i,j} \in \mathbb{N}^{(\dim(\mathbf{t})+1)\times(\dim(\hat{\mathbf{t}})+1)}$ it gives $\text{rev}(\mathbf{L}) = [l_{(\dim(\mathbf{t})+1)-i,(\dim(\hat{\mathbf{t}})+1)-j}]_{i,j}$). The sum of the two matrices $\mathbf{L} + \mathbf{L}^{(\text{bw})} = [\text{lev}(\mathbf{t}_{\leqslant i}, \hat{\mathbf{t}}_{\leqslant j}) + \text{lev}(\mathbf{t}_{>i}, \hat{\mathbf{t}}_{>j})]_{i,j}$ then, according to Eq. (3.26), contains the value of the Levenshtein distance $\text{lev}(\mathbf{t}, \hat{\mathbf{t}})$ in every cell on an optimal path. Any other entry that is not on an optimal path must be larger than $\text{lev}(\mathbf{t}, \hat{\mathbf{t}})$ because otherwise an optimal path would go through this entry. Thus, the total Levenshtein distance can be computed from the forward and backward Levenshtein matrices for every column index i as

$$\forall i: \quad \text{lev}(\mathbf{t}, \hat{\mathbf{t}}) = \min_{0 \leqslant j \leqslant \dim(\hat{\mathbf{t}})} (l_{i,j} + l^{(\text{bw})}_{i,j}). \tag{3.28}$$

With this property, l^+_{su} and l^-_{su} are found more efficiently. Given $\mathbf{L}_s$ and $\mathbf{L}^{(\text{bw})}_s$ for matching the assigned reference transcript to the hypothesis transcript on stream s, the Levenshtein distance for removing utterance u, positioned from column i_1 to column i_2 in $\mathbf{L}$ is computed with Eq. (3.28) but with the corresponding columns removed:

$$l^-_{su} = \min_{1 \leqslant j \leqslant \hat{\mathbf{t}}^{(\text{str})}_s} (l_{s,i_1,j} + l^{(\text{bw})}_{s,i_2,j}). \tag{3.29}$$

The value for l^+_{su} where utterance u is inserted at index i, can be computed similarly, but with the corresponding columns inserted into $\mathbf{L}$. These columns are computed with Eq. (2.19) based on the i-th column of $\mathbf{L}$.

Complexity This optimization significantly speeds up the computation of l^+_{su} and l^-_{su}, as the Levenshtein distance for the complete stream does not have to be computed for every utterance, stream and update. With this optimization, only one matrix, $\mathbf{L}^{(\text{bw})}_s$, has to be pre-computed on the full streams for every stream s. During the iteration, only minor updates have to be computed that in total amount to matching every utterance (instead

Algorithm 2 Optimized greedy update routine

function ForwardLevCol($\mathbf{l}$, $\mathbf{t}$, $\hat{\mathbf{t}}$, C_S)
  ▷ *Compute a column of the Levenshtein matrix from a previous column and the current utterance using the Wagner-Fischer algorithm from Eq. (2.19). Details are left out for brevity.* ◁
function greedyUpdateLabels($\mathbf{t}_{1,...,|\mathcal{U}|}$, $\hat{\mathbf{t}}^{(\mathrm{str})}_{1,...,S}$, ℓ, C_S)
  while not converged **do**
    ▷ *Initialize columns of the backward matrix* ◁
    for $s \in \{1, ..., S\}$ **do**
      $\mathbf{l}^{(\mathrm{bw})}_{s,|\mathcal{U}|+1} \leftarrow [1, ..., \dim(\mathbf{t}^{(\mathrm{spk})}_s)]$
      for $u \in (|\mathcal{U}|, |\mathcal{U}|-1, ..., 1)$ **do**
        if $\ell_u = s$ **then**
          $\mathbf{l}^{(\mathrm{bw})}_{s,u} \leftarrow \mathrm{rev}(\mathrm{ForwardLevCol}(\mathrm{rev}(\mathbf{l}^{(\mathrm{bw})}_{s,u+1}), \mathrm{rev}(\hat{\mathbf{t}}^{(\mathrm{str})}_s), \mathrm{rev}(\mathbf{t}_u)), C_\mathrm{S})$
        else
          $\mathbf{l}^{(\mathrm{bw})}_{s,u} \leftarrow \mathbf{l}^{(\mathrm{bw})}_{s,u+1}$
      ▷ *Initialize the first column of each (forward) Levenshtein matrix.* ◁
      $\mathbf{l}_s \leftarrow [1, ..., \dim(\hat{\mathbf{t}}^{(\mathrm{spk})}_s)]$
      ▷ *Initialize storage for the current Levenshtein distance on each stream* ◁
      $l_s \leftarrow l^{(\mathrm{bw})}_{s,0,0}$, for all $1s \leqslant S$
    for $u \in \mathcal{U}$ **do**
      for $s \in \{1, ..., S\}$ **do**
        ▷ *Compute the updated column of the forward matrix when placing utterance u at the current position.* ◁
        $\mathbf{l}_s \leftarrow \mathrm{ForwardLevCol}(\mathbf{l}_s, \hat{\mathbf{t}}^{(\mathrm{str})}_s, \mathbf{t}_u, C_\mathrm{S})$
        ▷ *Compute the cost differences to the current assignment. Re-use values stored in l_s from the previous iteration where possible.* ◁
        if $\ell_u = s$ **then**
          $l^+_{su} \leftarrow l_s$
          $l^-_{su} \leftarrow \min_i(l_{s,i} + l^{(\mathrm{bw})}_{s,u-1,i})$
        else
          $l^+_{su} \leftarrow \min_i(l'_{s,i} + l^{(\mathrm{bw})}_{s,u,i})$
          $l^-_{su} \leftarrow l_s$
      ▷ *Find the best stream for utterance u as the minimum over the differences* ◁
      $\ell_u \leftarrow \arg\min_{1 \leqslant s \leqslant S}(l^+_{su} - l^-_{su})$
      ▷ *Update current stream costs* ◁
      $l_s \leftarrow l^-_{s,u}$, for all $1 \leqslant s \leqslant S$
      $l_{\ell_u} \leftarrow l^+_{\ell_u,u}$
  return ℓ

of the complete assigned reference stream) to every hypothesis stream. Additionally, the matrices $\mathbf{L}_s$ and $\mathbf{L}_s^{(\mathrm{bw})}$ do not have to be stored completely in memory, but $\mathbf{L}_s$ can be computed while progressing through the utterances, and columns from $\mathbf{L}^{(\mathrm{bw})}$ only have to be stored at utterance boundaries and can be discarded after they are used.

The overall complexity of a single iteration, assuming initial labels ℓ' for the reference utterance, is then $\mathcal{O}(\sum_{u\in\mathcal{U}}\sum_{s=1}^{S}\dim(\mathbf{t}_u)\cdot\dim(\hat{\mathbf{t}}_s^{(\mathrm{stream})}))$, where the complexity for pre-computing the backward matrices disappears because the computation of the updates for the forward matrices is an upper bound on the complexity for the backward matrices. The algorithm often converges after only a few iterations. The number of iterations required for convergence is assumed as a constant, so that it disappears in the complexity analysis.

3.3.4 MIMO-WER

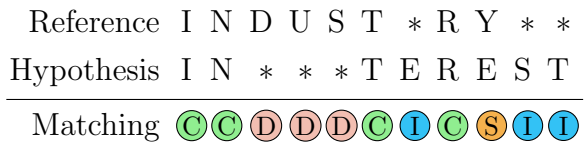
The MIMO-WER can be computed efficiently, similarly to the ORC-WER, using a dynamic programming algorithm. The derivations are similar to the ones for the ORC algorithm and left out here for brevity. Compared to ORC, the MIMO algorithm treats reference speakers independently, so that the utterances of each speaker can be matched independently of the utterances of other speakers. In comparison to Eq. (3.22), an additional index k is inserted that denotes to which reference speaker the utterance belongs that gets assigned to stream s. Different reference words are identified by an additional multi-index $\mathbf{r} = (r_1, ..., r_k)$:

$$l_{k,s,\mathbf{r},\mathbf{h}}^{(\mathrm{MIMO})} = \begin{cases} \min\limits_{\substack{1\leqslant\tilde{k}\leqslant K\\ 1\leqslant\tilde{s}\leqslant S}} l_{\tilde{k},\tilde{s},\mathbf{r}-\mathbf{e}_{\tilde{k}}^{(K)},\mathbf{h}}^{(\mathrm{MIMO})}, & \text{if } \forall\tilde{k}: t_{\tilde{k},r_{\tilde{k}}}^{(\mathrm{spk})} = [\mathrm{ct}], \\ \min\begin{cases} l_{k,s,\mathbf{r}-\mathbf{e}_k^{(K)},\mathbf{h}}^{(\mathrm{MIMO})} + C_{\mathrm{D}}, \\ l_{k,s,\mathbf{r},\mathbf{h}-\mathbf{e}_s^{(S)}}^{(\mathrm{MIMO})} + C_{\mathrm{I}}, \\ l_{k,s,\mathbf{r}-\mathbf{e}_k^{(K)},\mathbf{h}-\mathbf{e}_s^{(S)}}^{(\mathrm{MIMO})} + C_{\mathrm{C/S}}, \end{cases} & \text{otherwise.} \end{cases} \tag{3.30}$$

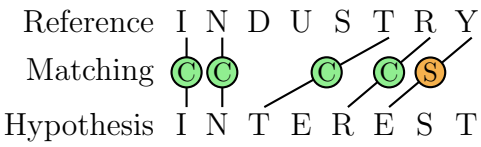
Here, the lower minimum again resembles the standard Levenshtein distance Eq. (2.19) across a slice of $\mathbf{L}^{(\mathrm{MIMO})}$, for a reference utterance from speaker k compared to the full stream s. The upper minimum only applies when all indices in the multi-index $\mathbf{r} = (r_1, ..., r_K)$ point to a change token.[13]

Computational complexity The computational complexity of the MIMO-WER is larger than that of the ORC-WER because it allows for changes in the utterance order. $\mathbf{L}^{(\mathrm{MIMO})}$ contains K additional dimensions compared to $\mathbf{L}^{(\mathrm{ORC})}$. The upper case in Eq. (3.30) is executed for every combination of change tokens across the speakers. If we denote with N_k the number of utterances of speaker k, the upper case is executed $\prod_{k=1}^{K} N_k$ times for every combination of hypothesis indices, of which there are $\prod_{s=1}^{S}\dim(\hat{\mathbf{t}}_s^{(\mathrm{str})})$ many, and accesses KS many cells in $\mathbf{L}^{(\mathrm{MIMO})}$. The lower case is executed $KS\prod_{k=1}^{K} N_k \prod_{s=1}^{S}\dim(\hat{\mathbf{t}}_s^{(\mathrm{str})})\sum_{k=1}^{K}\dim(\mathbf{t}_k^{(\mathrm{spk})})$ many times and accesses 3 cells of $\mathbf{L}^{(\mathrm{MIMO})}$. This leads to a total complexity of $\mathcal{O}((1+3\cdot\sum_{k=1}^{K}\dim(\mathbf{t}_k^{(\mathrm{spk})}))KS\prod_{k=1}^{K} N_k\prod_{s=1}^{s}\dim(\hat{\mathbf{t}}_s^{(\mathrm{spk})}))$. Note that not all possible cells in $\mathbf{L}^{(\mathrm{MIMO})}$ are reflected in this complexity. This is because the lower minimum operation is only ever

[13] The equation was modified from [OC7] to make this explicit.



(a) *Alignment: nulls* (∗) are inserted so that both sequences have the same length and matching tokens align.



(b) *Trace:* Matching tokens are connected with a line. Insertions and deletions are shown implicitly by tokens that are not connected.

Figure 3.12: Different ways for visualizing a Levenshtein matching defined in [85]. Correct or Substitution is marked with C and S and Insertion or Deletion with I and D. Colors are used to facilitate identifying error types. Multiple alignments can correspond to the same trace.
(Taken from [OC8])

executed for cells where all but one reference index point to a change token [ct]. All combinations of indices where more than one reference index point to a word other than [ct] are never visited. The overall complexity is exponential in the number of reference speakers K and the number of hypothesis streams S. A greedy approximation is not considered here.

3.3.5 DI-cpWER

The DI-cpWER, as already discussed, is equal to the ORC-WER with reference and hypothesis swapped. Thus, the same complexity analysis and algorithms apply to the DI-cpWER as to the ORC-WER, but with swapped arguments. Specifically, the naive greedy algorithm from Section 3.3.3 was initially used for the DI-cpWER in [36].

3.4 System Analysis With Error Visualization

A good visualization of the alignment between a reference and a hypothesis transcript can reveal the cause of errors of the recognizer better than what is possible from the metrics alone. While a metric merely counts the errors to get an aggregated performance measure, a visualization can provide detailed insights where errors occur. Such insights can be crucial for system development and debugging. This section describes the design rationale behind a visualization tool developed for the visualization of meeting-level alignments. The proposed visualization style is used in Section 3.5.8 for analyzing errors in three case studies.

3.4.1 Visualization Styles

The two most common visualization techniques for alignments are the *trace* and the *alignment* visualization [85]. Both are visualized for an example pair of reference and hypothesis transcript in Fig. 3.12.

In *alignment* style (Fig. 3.12a), filler words are inserted into both the reference and the hypothesis transcripts such that matching words align, i.e., have the same position in the sequence. This type of visualization is commonly used for single-speaker utterance recognition tasks [9]. It works with plain text, which makes it simple to create and view. It is clean and easy to read for short sequences, but quickly becomes cluttered for long sequences, when the

line length exceeds the page or screen width. Because of this, it is not well suited for long meeting-style data.

The *trace* visualization (Fig. 3.12b) displays words of reference and hypothesis side-by-side and connects two words with a line if they match. This style allows for keeping the temporal constellation of the words and for keeping the visualization clean for long sequences.

3.4.2 Visualizing Matchings of Long Word Sequences

As already mentioned, the alignment visualization is not well suited for long sequences, as it quickly becomes cluttered. The trace visualization, on the other hand, can be used for long sequences.

Interactive tool An interactive tool is required to view the trace alignment for long sequences, so that the user can zoom and scroll to the important areas. Such a tool was developed as part of the `MeetEval` toolkit.[14] The tool generates visualizations as interactive HTML documents that can be viewed in any modern web browser.

Vertical trace For long word sequences, it turned out that a vertical trace, where time flows from top to bottom, is easier to read than horizontal direction. One reason for this is that words have different lengths, so they would have to be rotated to fit without overlapping in the horizontal direction. This is not necessary in the vertical direction, where the words can be displayed in a single column.

Error summary In order to quickly find relevant positions, a summary of the errors is displayed at the top of the trace in the tool. It highlights the number of correct words, substitutions, insertions and deletions over time using a stacked bar chart. This allows the user to quickly identify the time intervals where errors cluster.

3.5 Analysis

This section presents an analysis of the impact of different types of errors on the WERs. Experiments are both performed with artificially modified transcripts to have precise control over the errors and on real transcripts obtained from meeting transcription systems on different datasets to get a picture of the behavior in realistic scenarios.

3.5.1 Datasets and Systems

The analysis uses the LibriCSS dataset (see Section 2.5.1.4) and the submissions to the DASR track of the 7th CHiME challenge [2]. By using the challenge submissions, it is possible to analyze the metrics across a large number of different systems and two different datasets[15]. These include a variety of approaches, all based on modular pipelines, shortly outlined below. The baseline system [2] uses a diarization module from Pyannote [12] based on EEND neural

[14] A showcase is available at https://fgnt.github.io/meeteval_viz.

[15] The DASR challenge included three datasets, CHiME-6, DiPCo and Mixer6, but Mixer6 is excluded here for licensing reasons.

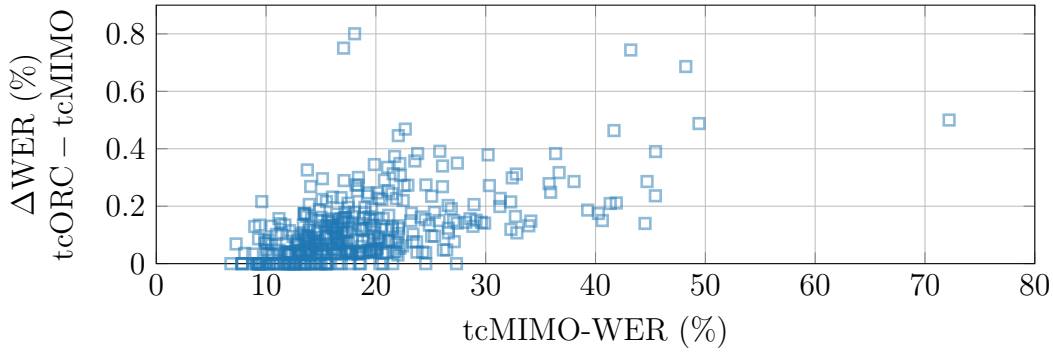


Figure 3.13: Absolute difference in percentage points of time-constrained MIMO-WER (MIMO-WER) and tcORC-WER for examples across all collected submissions from the 7th CHiME challenge. Their average difference is smaller than 0.2 percentage points on this data. The ORC-WER is thus a good replacement for the MIMO-WER for this set of data and systems. *(Taken from [OC8])*

diarization followed by GSS [23] for multi-channel source extraction and a single-channel ASR system applied to the separated speech. Both [15] and [16] keep the basic pipeline structure but replace the diarization and ASR modules with stronger sub-systems. Both [17] and [OC15] replace the diarization with a TS-VAD [35] system. The challenge winner [27] uses a sophisticated multi-stage pipeline with multiple applications of clustering-based diarization, a multi-channel cACGMM mixture model, neural speaker diarization, and GSS.

3.5.2 Comparison of ORC-WER and MIMO-WER

The time-constrained versions of ORC-WER and MIMO-WER are compared in Fig. 3.13. The plot shows the time-constrained versions due to the reduced computational complexity, but the same conclusions also hold for the non-time-constrained variants. The two performance measures are strongly correlated and differ by less than 0.2 percentage points on average. This matches the discussions in Section 3.2.3 that differences only occur when a system does not keep the physical utterance order. The submissions to the CHiME challenge contained no system that was explicitly allowed to modify the utterance order.

Minor differences can occur even if the utterance order is not changed, e.g., when the order is unclear (the begin times are close) or when artifacts or similar words cause ambiguous assignments. In most cases, however, (tc)ORC-WER can be used in place of (tc)MIMO-WER to make the computation feasible without loosing much expressiveness. In the following experiments, only the ORC-WER is analyzed due to the higher computational complexity of MIMO-WER.

3.5.3 Impact of Timestamp Errors on the WERs

Fig. 3.14 shows the behavior of WERs when the timestamps are inaccurate, simulated by a random jitter $|\Delta_t|$ of up to 15 s added to the hypothesis timestamps. The analysis is based on the LibriCSS-raw dataset and the ground-truth transcriptions are scored against the ground-truth transcriptions with modified timestamps.

The non-time-constrained cpWER is unaffected by timestamp errors and remains constant at 0 %. All time-constrained WERs increase with the amount of jitter, but only significantly

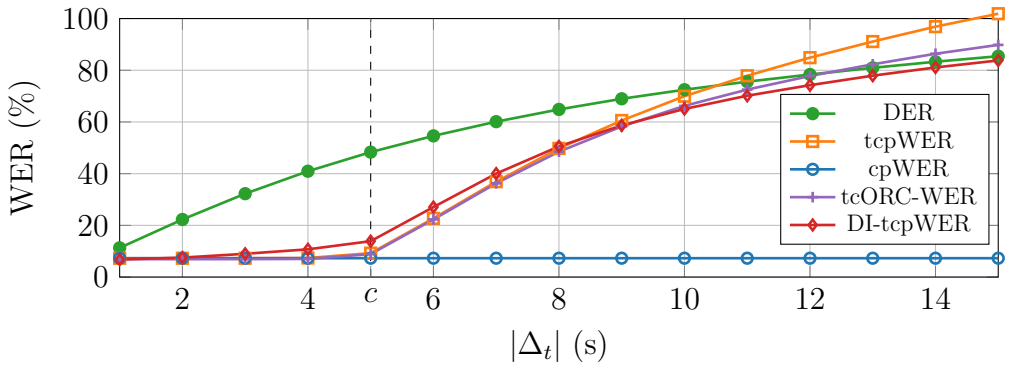


Figure 3.14: Impact of a random jitter in the timestamps on different WERs. Analyzed on the LibriCSS-raw dataset. Time-constrained WERs are computed with a collar of $c = 5\,\text{s}$.
(Taken from [OC8])

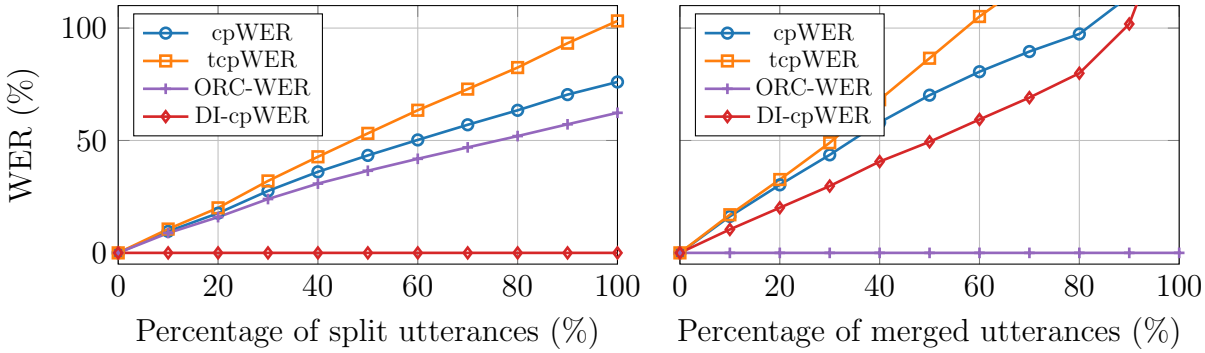


Figure 3.15: Impact of segmentation errors (utterance splits or utterance merges) in combination with speaker attribution errors on different WERs. Analyzed on the LibriCSS-raw dataset.

after the jitter surpasses the collar, marked with c.

All time-constrained error rates correlate to some degree with the DER while the non-time-constrained cpWERs does not correlate with the DER.

3.5.4 Impact of Segmentation Errors on the WERs

The impact of utterance split and utterance merge errors is analyzed in Fig. 3.15. Utterances from LibriCSS-raw are randomly split into two (left plot) or merged into one (right plot). The cpWER and tcpWER react to both of these errors, where the tcpWER is slightly more sensitive to the errors due to the imprecise estimation of the word-level timestamps. As already discussed in Section 3.2.2, the ORC-WER ignores utterance merges but penalizes utterance splits. The DI-cpWER (Section 3.2.4), on the other hand, ignores utterance splits and penalizes utterance merges.

Note that the analysis in Fig. 3.15 is performed on data with exclusively only utterance split or utterance merge errors and no other errors. In a real evaluation, typically both utterance merge and utterance split errors are present and mixed with other errors. The sign of the difference of ORC-WER and DI-cpWER still indicates which of the two, utterance merge or utterance split, causes more word-level errors.

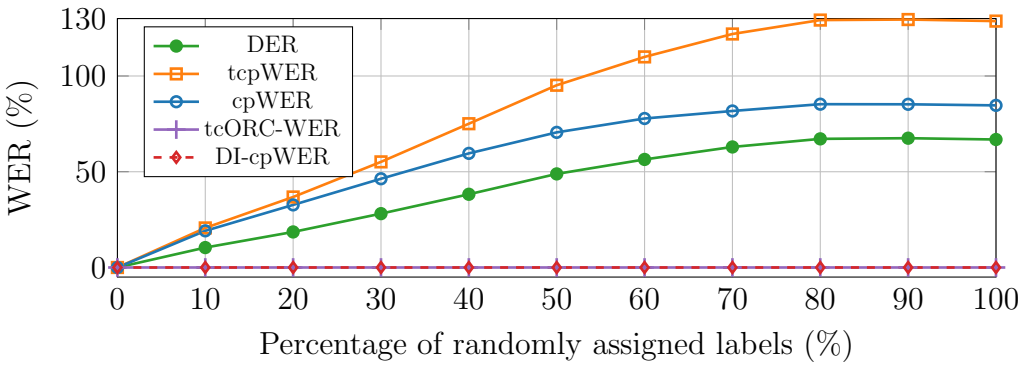


Figure 3.16: Impact of speaker attribution errors on different WERs Analyzed on the LibriCSS-raw dataset. DI-cpWER is unaffected while all others penalize speaker attribution errors to some degree. *(Taken from [OC8])*

3.5.5 Impact of Speaker Attribution Errors on the WERs

The impact of speaker attribution errors on the WERs is analyzed in detail in Fig. 3.16. For this analysis, the reference transcript of LibriCSS-raw is evaluated against the reference with randomly changed speaker labels. The amount specifies how many labels are switched to a random other label. At an amount of 0, the speaker labels are unchanged (resulting in an error rate of 0) and at an amount of 1, all labels are changed to a different label.

Since there are no segmentation errors in this scenario, i.e., utterance split or utterance merge, the DI-(t)cpWER and (tc)ORC-WER are constant over the amount of label switches. As analyzed earlier, both of these metrics react to segmentation errors only if they coincide with a speaker attribution error, in which case they increase.

The cpWER, designed to count speaker attribution errors, initially increases linearly with the amount of speaker label switches, but flattens when the amount of speaker attribution errors goes to 100 %. This is because insertion and deletion errors are merged into substitution errors if it improves the error rate, even if the matching is implausible, i.e., words are temporally far apart. At some point, almost all words are wrongly recognized and merged into substitutions, so that the error rate cannot increase further. It does not reach 100 % because common words are matched as correct even if they stem from different speakers and, due to the permutation invariance, some utterances are always correctly matched.

The tcpWER is the only WER that surpasses 100 % in the analysis in Fig. 3.16. Speaker attribution errors have a larger effect on the tcpWER compared to the cpWER because the tcpWER only allows matchings within the collar. A speaker label switch that causes a segment to land in a region where the reference speaker is silent creates insertion errors in the tcpWER, but the cpWER may hide these errors by matching them with words that are temporally far away. The tcpWER thus reflects speaker attribution errors more accurately than the cpWER. Such a situation where the cpWER creates an implausible mapping but tcpWER does not is analyzed in a case study in Section 3.5.8.

This effect is analyzed in more detail in Fig. 3.17, where the decomposition of the total amount of errors into insertions, deletions and substitutions is shown for cpWER and tcpWER across different amounts of wrongly assigned speaker labels. The cpWER maintains an almost constant amount of insertions and deletions when sufficiently many speaker attribution errors are present. The remaining errors are merged into substitutions. The tcpWER maintains a

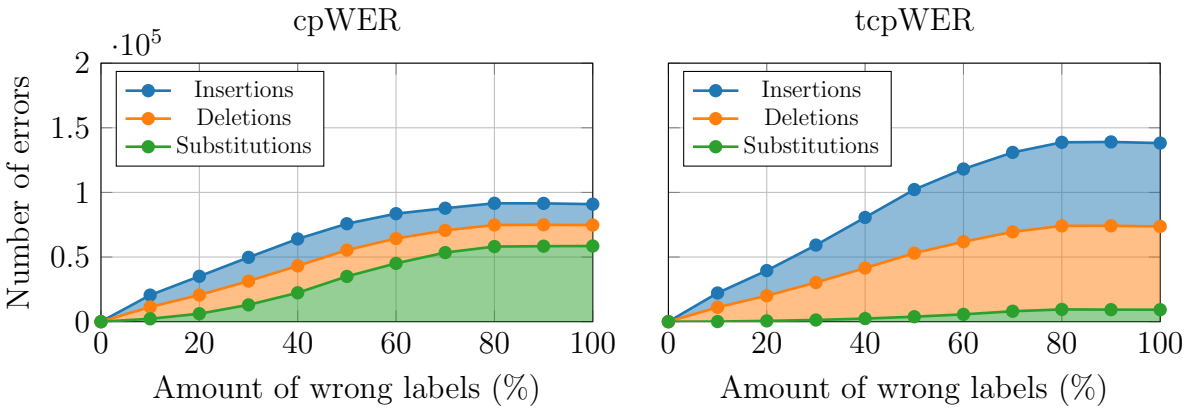


Figure 3.17: Breakdown as stacked area plots of the total number of errors into insertions, deletions and substitutions for cpWER and tcpWER for different amounts of wrongly assigned speaker labels on the LibriCSS-raw dataset. The cpWER merges insertions and deletions into substitutions across implausible distances, while the tcpWER does so only to a very limited extend.

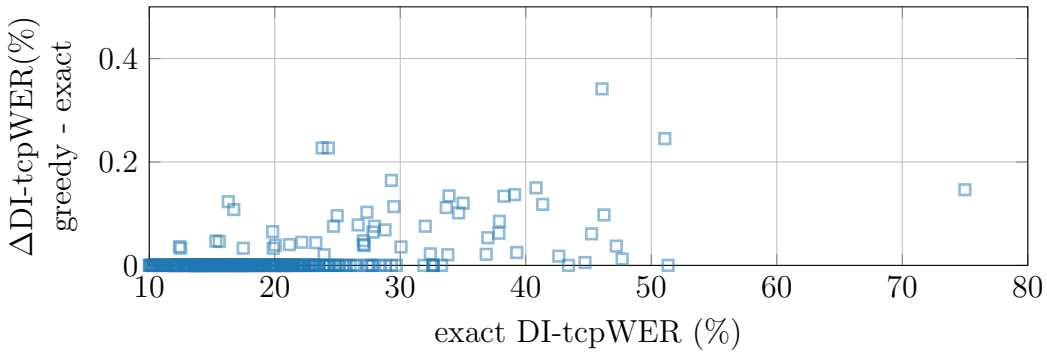


Figure 3.18: Absolute difference between the results of the the greedy algorithm and the exact algorithm for the DI-tcpWER. These results also hold for the (tc)ORC-WER. *(Taken from [OC8])*

low amount of substitution errors, caused by overlapping utterances being assigned to the wrong speaker. All remaining speaker attribution errors lead to a roughly equal amount of insertions and deletions. The tcpWER is overall higher but more plausible than the cpWER.

Note that all analyzed WERs ignore speaker labels to some degree, e.g., up to a permutation for the cpWER. Because of this and the fact that common words are matched as correct even if the speaker label is wrong and even across long temporal distances, no non-time-constrained WER reaches 100 % error rate, even if all speaker labels are switched. All speaker-attributed error rates (cpWER and tcpWER) correlate to some degree with the DER.

3.5.6 Accuracy of the Greedy Algorithm for ORC-WER and DI-cpWER

The computed DI-cpWERs for the greedy algorithm and the exact algorithm are compared in Fig. 3.18, as a scatter plot of their difference over all systems and sessions from the CHiME-7 challenge submissions. The two algorithms barely differ on this data, by only 0.02 percentage points on average. They yield the exact same result for 86 % of the examples. A difference of

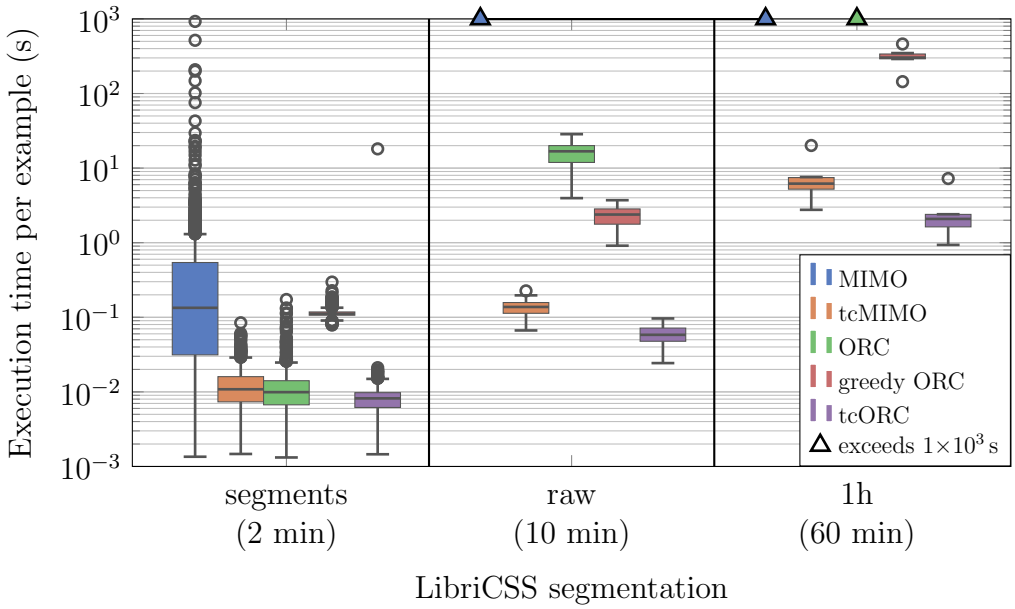


Figure 3.19: Execution time of the different algorithms for different segmentations of the LibriCSS dataset. *(Taken from [OC8])*

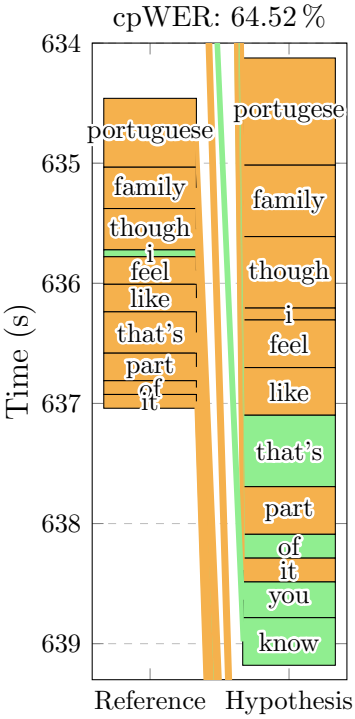
0.02 percentage points can be considered insignificant, as most publications report WERs with a lower precision. The result of the greedy algorithm provides an upper bound on the WER; it can never be smaller (better) than the exact computation. The greedy algorithm is thus a good approximation to the exact algorithm.

3.5.7 Execution Time

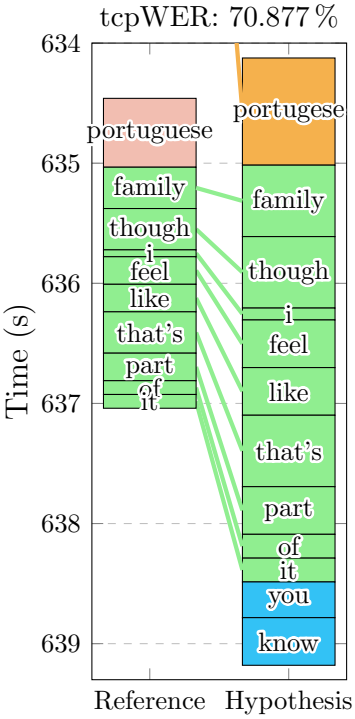
The execution times for the different algorithms are shown in Fig. 3.19 for different segmentations of the LibriCSS dataset and a two-output ($S = 2$) separation-first CSS pipeline. The different segmentations exhibit different temporal lengths and different numbers of speakers per example, where the segments have an average length of 2 min with on average 4.8 speakers, the examples from the raw segmentation have an average length of 10 min and 8 speakers, and the examples from the 1h segmentation, which combines multiple raw sessions, have an average length of 1 h and 8 speakers. All algorithms are implemented in C++ except for the greedy algorithm, which is implemented in Python using numpy.

On the short segments, all algorithms are feasible to compute in a reasonable time. The MIMO-WER quickly becomes infeasible already for the 10 min segments, where the (exact) ORC-WER is still feasible. For the long segments, both ORC-WER and MIMO-WER become infeasible. The time-constrained variants of ORC-WER and MIMO-WER are both feasible even for the long segments due to the pruning of the search space. Note that the execution time in this case heavily depends on the number of speakers that overlap within the range of the collar. The greedy algorithm is feasible for all segmentations.

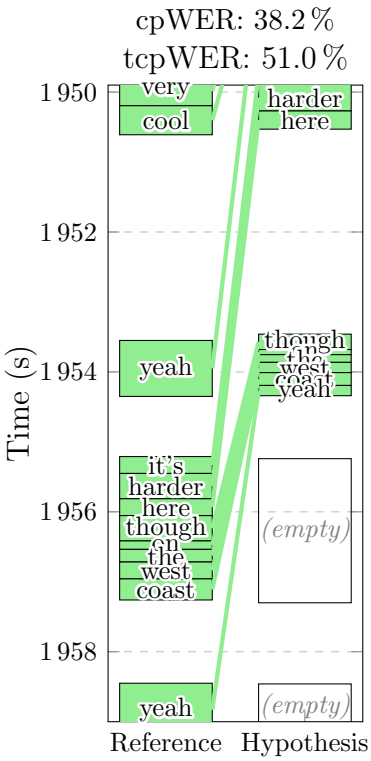
Note that the theoretical runtime of the algorithms compared here depends mainly on the number of speakers in the reference, and less on the number of system output streams, except for the MIMO-WER. The small number of system output streams used in this analysis has thus no significant impact on the measurements.



(a) Implausible matching produced by a non-time-constrained WER, such as cpWER or ORC-WER. The transcript is correct but words are matched as substitutions across large temporal distances. Obtained for an example from the CHiME-6 dataset.



(b) The same example as Fig. 3.20a, but with a time-constrained WER. The matching is now more plausible. "portugese" is not matched with "portuguese" because the matching is ambiguous. Obtained for an example from the DiPCo dataset.



(c) cpWER matching of decoupled diarization and recognition. Empty segments may result in substitutions instead of deletions. tcpWER would count these as deletions. Obtained for an example from the DiPCo dataset.

Figure 3.20: Case studies: Trace visualization for example excerpts from a single speaker from the DiPCo and CHiME-6 corpus. Time flows from top to bottom. Boxes represent word start and end times and matched words are connected with a line. Connecting lines may connect words that lie outside of the displayed excerpt. Matchings are color-coded as correct ■, substitution ■, deletion ■, and insertion ■. *(Taken from [OC8])*

3.5.8 Case Studies

These case studies illustrate the behavior and the issues of the time-constrained and non-time-constrained WERs mentioned in the discussions above in more detail. They specifically highlight the importance of the time-constraint for obtaining plausible matchings in the meeting scenario.

3.5.8.1 Case Study 1: Time-constrained Matching

Fig. 3.20a shows an example of an implausible matching produced by the (non-time-constrained) cpWER. Almost all words in this excerpt are recognized correctly at roughly the correct time. The cpWER, however, matches most of them as substitutions with other words that are temporally far apart and lie outside of the displayed excerpt. This happens because the cpWER could here trade substitution errors for insertion and deletion errors across large distances to reduce the overall error rate.

The alignment created by the tcpWER, as displayed for the same excerpt in Fig. 3.20b, creates a plausible matching where the correctly transcribed words are also recognized as correctly transcribed by the metric. This again highlights the importance of the time-constraint for the meeting scenario.

3.5.8.2 Case Study 2: Decoupled Diarization and Recognition

The third excerpt in Fig. 3.20c shows the non-time-constrained matching for an interesting case that appeared for one submission to the 7th CHiME challenge. The system treated diarization and speech recognition separately such that both the diarization and the speech recognition produce good results when analyzed separately. But, as shown in Fig. 3.20c, the transcribed speech was sometimes assigned to the wrong activity segment from the diarization. The DA-WER (defined in Section 3.2.5.3), which was the official scoring metric for the 7th CHiME challenge, produced a good score since it treats the diarization and speech recognition components only loosely coupled. The cpWER does not detect these errors because it is invariant to timestamp errors. The tcpWER, on the other hand, penalizes these errors when the wrong assignment leads to gaps that are larger than the collar. In this example, the WER increases significantly from 38.2 % to 51.0 % when the time constraint is applied, indicating issues in the temporal annotations.

3.6 Summary

This chapter presented a detailed overview over how to compute the WER in the meeting scenario. Several WER definitions were presented, including the ORC-WER, MIMO-WER, and DI-cpWER. The DI-cpWER is especially interesting because it allows for an analysis of how many errors are introduced by faulty diarization. In order to make the alignments and resulting WERs more plausible for long transcriptions, a temporal constraint was introduced to the computation of the Levenshtein distance. This constraint improves the plausibility of matchings by disallowing the matching of words across large temporal distances. Algorithms were presented for efficiently computing the WERs. For the ORC-WER and DI-cpWER, additionally a greedy approximation was proposed that is in many cases significantly faster than the exact algorithm and often finds the optimal solution. A visualization scheme for long

sequence alignments was presented and its effectiveness was exemplified in two case studies, where systematic system issues could be identified that were not visible in the metrics alone.

The WERs presented in this chapter will be used in the following chapters to analyze systems and evaluate the overall system performance. The alignment visualization tool proved to be useful throughout the developments of the systems presented in the following chapters.

All metrics and the visualization presented in this chapter are implemented and published in the open-source MeetEval toolkit[16]. The toolkit is well received in the community. The tcpWER was used as an official evaluation metric in the CHiME-8 DASR challenge [3] track of the 8th CHiME challenge. The NOTSOFAR-1 challenge [1] also used the tcpWER as the main ranking metric and the tcORC-WER as a complementary metric. The visualization tool is also well-received. It was presented in a Show-and-Tell Demo session at ICASSP 2024 and was already used in publications to provide a detailed analysis of the system behavior for selected examples [86].

[16] https://github.com/fgnt/meeteval

4 Robust Loss Functions for Meeting Separation

Related own publications

This chapter is based on [OC6], which introduced the Source-Aggregated Signal-to-Distortion Ratio (SA-SDR) as a loss function (here in Section 4.2) and extends the prior work with discussions about the scale-invariant SDR in Section 4.3 and about how the logarithm in the SDR affects the training in Section 4.2.2.

Speech separation in meeting scenarios extends over the conventional fully overlapped speech separation problem, where the uPIT training scheme (see Section 2.6.1) is commonly used. Since uPIT assumes a fixed number of K speakers in training and this exact number of speakers is active for the whole duration of the mixture, it does not apply to the meeting scenario without modification. The fully overlapped speech activity pattern is rarely found in realistic meeting scenarios where speakers frequently change turns. Even after segmenting the recording into smaller segments (see CSS and stitching in Section 2.6.2), the segments contain varying numbers of speakers. This variation in the number of speakers leads to multiple challenges when computing a loss in training: When the number of speakers is smaller than the number of outputs ($K < S$), it creates silent targets, i.e., no speech is active in at least one ground-truth signal such that $\mathbf{s} = \mathbf{0}$. Even if no target is silent, different target signals likely contain silent regions of different lengths which creates an imbalanced difficulty across streams and examples.

Both challenges create issues in the SDR, which is the most commonly used loss function, defined as

$$\mathcal{L}^{(\mathrm{SDR})}(\mathbf{s}, \hat{\mathbf{s}}) = -\mathrm{SDR}(\mathbf{s}, \hat{\mathbf{s}}) = -10 \log_{10} \left(\frac{\|\mathbf{s}\|^2}{\|\mathbf{s} - \hat{\mathbf{s}}\|^2} \right), \tag{4.1}$$

which measures how much the estimated signal $\hat{\mathbf{s}}$ is distorted from the original source signal $\mathbf{s}$ in relation to the energy of the original source signal $\mathbf{s}$. For computing the loss over multiple $K = S$ pairs of reference and estimated signals $\mathbf{s}_1, ..., \mathbf{s}_K$ and $\hat{\mathbf{s}}_1, ..., \hat{\mathbf{s}}_K$, we here assume that the number of system output streams matches the number of speakers and that the assignment problem is already solved[1]. The loss is in this case typically averaged across streams [50]. The averaged loss is here called averaged SDR (A-SDR). Using the notation from Eq. (2.11), where $\mathbf{S}$ and $\hat{\mathbf{S}}$ are matrices that contain the stacked signals $\mathbf{s}_1, ..., \mathbf{s}_K$ and

[1] This is the case for uPIT in a speaker-attributed way, as described in Section 2.6.1, where every stream represents a speaker. Another technique to achieve this in a speaker-agnostic way is presented in the next Chapter as Graph-PIT.

$\hat{\mathbf{s}}_1, ..., \hat{\mathbf{s}}_K$, respectively[2], it is written as:

$$\mathcal{L}^{(\text{A-SDR})}(\mathbf{S}, \hat{\mathbf{S}}) = \frac{1}{K} \sum_{k=1}^{K} \mathcal{L}^{(\text{SDR})}(\mathbf{s}_k, \hat{\mathbf{s}}_k) = -\frac{10}{K} \sum_{k=1}^{K} \log_{10} \left(\frac{\|\mathbf{s}_k\|^2}{\|\mathbf{s}_k - \hat{\mathbf{s}}_k\|^2} \right). \tag{4.2}$$

The A-SDR becomes unstable when the loss on one of the streams becomes unstable for silent targets or perfect reconstruction. For silent targets, where $\mathbf{s} = \mathbf{0}$, the loss diverges to

$$\lim_{\mathbf{s} \to \mathbf{0}} \mathcal{L}^{(\text{SDR})}(\mathbf{s}, \hat{\mathbf{s}}) = -\lim_{\mathbf{s} \to \mathbf{0}} \log_{10} \left(\frac{\|\mathbf{s}\|^2}{\|\mathbf{s} - \hat{\mathbf{s}}\|^2} \right) = +\infty. \tag{4.3}$$

For perfect reconstruction, where $\hat{\mathbf{s}} \to \mathbf{s}$, the loss diverges to

$$\lim_{\hat{\mathbf{s}} \to \mathbf{s}} \mathcal{L}^{(\text{SDR})}(\mathbf{s}, \hat{\mathbf{s}}) = -\lim_{\hat{\mathbf{s}} \to \mathbf{s}} \log_{10} \left(\frac{\|\mathbf{s}\|^2}{\|\mathbf{s} - \hat{\mathbf{s}}\|^2} \right) = -\infty. \tag{4.4}$$

The A-SDR also shows problematic behavior for imbalances in difficulty across streams, i.e., when the error of one stream is much smaller than the error of another stream: $\|\mathbf{s}_i - \hat{\mathbf{s}}_i\|^2 \gg \|\mathbf{s}_j - \hat{\mathbf{s}}_j\|^2$. This issue is discussed below in Section 4.2.1.

The following section first introduces approaches from the literature to address these issues. Section 4.2 then introduces the Source-Aggregated SDR (SA-SDR) [OC6] loss as an elegant way to address all problems mentioned above. Section 4.2.2 further discusses how the choice of the loss function influences the weighting of examples within a batch and across batches in training.

4.1 Conventional Losses for Meeting Separation

Most approaches to stabilize the SDR loss for speech separation in meeting data are based on modifying the SDR itself by either adding a small constant to the numerator, skewing the SDR curve, or removing the numerator completely. Most of the approaches only address one of the issues mentioned above. The following equations are defined for a single pair of reference signal $\mathbf{s}$ and estimated signal $\hat{\mathbf{s}}$, and are averaged across speakers similar to Eq. (4.2) when training.

Stabilizing with a small constant The most straightforward way to handle silent targets is to add a small value ϵ to the numerator of the SDR to prevent it from exploding for silent targets [OC5]:

$$\mathcal{L}^{(\epsilon\text{-SDR})}(\mathbf{s}, \hat{\mathbf{s}}) = -10 \log_{10} \left(\frac{\|\mathbf{s}\|^2 + \epsilon}{\|\mathbf{s} - \hat{\mathbf{s}}\|^2} \right). \tag{4.5}$$

The constant ϵ is typically chosen small enough to not influence the SDR for non-silent targets, but large enough to avoid numerical issues. While ϵ influences the value of the SDR and the weighting across streams, it does not alter the direction of the gradient for a single stream because the numerator disappears in the derivative.

[2] We here assume that each source signal spans the full recording, such that $\mathbf{s}_u = \breve{\mathbf{s}}_u$.

Skewing the SDR The skewed SDR[3] addresses the robustness for perfect reconstruction by skewing the SDR curve such that it is less steep for perfect reconstructions [87], here reformulated in a scale-dependent way:

$$\mathcal{L}^{(\text{skewed SDR})}(\mathbf{s}, \hat{\mathbf{s}}) = -10 \log_{10} \left(\frac{\|\mathbf{s}\|^2}{\|\mathbf{s} - \hat{\mathbf{s}}\|^2 + \nu \|\hat{\mathbf{s}}\|^2} \right), \tag{4.6}$$

which converges to

$$\lim_{\|\mathbf{s}-\hat{\mathbf{s}}\|^2 \to 0} \mathcal{L}^{(\text{skewed SDR})}(\mathbf{s}, \hat{\mathbf{s}}) = -10 \log_{10} \left(\frac{1}{\nu} \right) \tag{4.7}$$

for perfect reconstruction. The skewing factor ν controls how much the loss value is skewed for small reconstruction errors and how large its value becomes for a reconstruction error of 0. It is typically chosen small, e.g., $\nu = 0.3$. In its original work, it was used to reconstruct the mixture signal instead of silent targets and only applied for silent targets.[4]

Soft maximum The SDR can be made robust against perfect reconstruction and for imbalanced streams by introducing a soft maximum [88]:

$$\mathcal{L}^{(\text{tSDR})}(\mathbf{s}, \hat{\mathbf{s}}) = -10 \log_{10} \left(\frac{\|\mathbf{s}\|^2}{\|\mathbf{s} - \hat{\mathbf{s}}\|^2 + \tau \|\mathbf{s}\|^2} \right) \tag{4.8}$$

where $\tau = 10^{-\text{SDR}_{\max}/10}$ is a threshold that limits the minimum loss (for an error of 0) to $-\text{SDR}_{\max}$. The additional term $\tau \|\mathbf{s}\|^2$ in the denominator moves the log curve w.r.t. the error $\|\mathbf{s} - \hat{\mathbf{s}}\|^2$ such that an error of 0 coincides with $-\text{SDR}_{\max}$. In contrast to the skewed SDR, it does not otherwise change the shape of the function.

Log-MSE Removing the numerator from the SDR does not affect its gradients, but removes the instability for silent targets. Without the numerator, the SDR becomes the log-MSE [89]:

$$\mathcal{L}^{(\text{log-MSE})}(\mathbf{s}, \hat{\mathbf{s}}) = 10 \log_{10} \left(\|\mathbf{s} - \hat{\mathbf{s}}\|^2 \right). \tag{4.9}$$

To also make it robust against perfect reconstruction, a constant can be added to the argument of the logarithm, which leads to the log1p-MSE (log-one-plus-MSE) [OC18]:

$$\mathcal{L}^{(\text{log1p-MSE})}(\mathbf{s}, \hat{\mathbf{s}}) = 10 \log_{10} \left(\|\mathbf{s} - \hat{\mathbf{s}}\|^2 + 1 \right). \tag{4.10}$$

The log-MSE has the disadvantage that its value depends on the scaling of the signals (which gets cancelled out in the SDR). It is thus harder to interpret because a higher value does not necessarily mean a worse separation. A model selection based on the best log-MSE loss is thus discouraged.

[3] It was in [87] proposed in a different formulation in a scale-invariant way, i.e., such that the absolute scale of the estimated signal $\hat{\mathbf{s}}$ relative to the reference $\mathbf{s}$ does not influence the loss value. We here only consider scale-dependent losses, so it is reformulated without scale-invariance.

[4] The scale-dependent formulation in Eq. (4.6) encourages the estimate to become zero ($\hat{\mathbf{s}} \to \mathbf{0}$) through the additional term $\nu \|\hat{\mathbf{s}}\|^2$. The original scale-invariant version does not do this.

Extra loss for silence A different approach to handle silent targets explored in [59] is to add an extra loss term for silent targets that only operates on $\|\hat{\mathbf{s}}\|^2$ and that is only evaluated for silent targets. This approach solves the problem of a silent target, but does not solve the problem of imbalanced streams. Having an extra loss additionally introduces the complexity of switching between and tuning the weights of the two loss functions.

Skip loss of silent target Another approach is to skip the loss computation for silent targets [90] or select the best matching other target signal or the mixture [87] as a target for the silent stream. This approach solves the problem of computing a loss for silent targets and even seems to improve performance compared to modifying the loss function, perhaps because it removes the stark imbalance between the silent and non-silent targets completely. It is, however, not well suited for a full pipeline where inactivity has to be detected reliably because the silent streams are either unconstrained or trained to output high energy.

4.2 SA-SDR: An Elegant Stabilization of the SDR

We here propose to address these issues by, instead of modifying the single-stream SDR and averaging across speakers, aggregating the energies across speakers before computing the SDR. This leads to the SA-SDR:

$$\mathcal{L}^{(\text{SA-SDR})}(\mathbf{S}, \hat{\mathbf{S}}) = -\text{SA-SDR}(\mathbf{S}, \hat{\mathbf{S}}) = -10 \log_{10} \frac{\sum_{k=1}^{K} \|\mathbf{s}_k\|^2}{\sum_{k=1}^{K} \|\mathbf{s}_k - \hat{\mathbf{s}}_k\|^2}. \tag{4.11}$$

This formulation is equal to concatenating all signals before computing the SDR, i.e.,

$$\text{SA-SDR}(\mathbf{S}, \hat{\mathbf{S}}) = \text{SDR}\left(\begin{bmatrix} \mathbf{s}_1 \\ \vdots \\ \mathbf{s}_K \end{bmatrix}, \begin{bmatrix} \hat{\mathbf{s}}_1 \\ \vdots \\ \hat{\mathbf{s}}_K \end{bmatrix}\right), \tag{4.12}$$

where all signal vectors are column vectors and the notation of stacked signals in square brackets means concatenation into one large column vector. The SA-SDR is stable as long as at least one target signal is non-zero because neither the numerator nor the denominator becomes zero if not all signals are zero. It also eliminates the imbalance between streams by combining the stream energies into a single term.

4.2.1 Behavior for Imbalanced Streams

A single well-separated estimated signal can dominate the A-SDR and produce extremely high values even when the other estimated signals have a poor quality. The same is not true for the SA-SDR. In training, when one stream is better separated than another, say $\|\mathbf{s}_k - \hat{\mathbf{s}}_k\|^2 \gg \|\mathbf{s}_l - \hat{\mathbf{s}}_l\|^2$, the A-SDR focuses on the already better separated stream l while the SA-SDR minimizes the total distortions by focusing on all streams equally. This can be seen from the gradients of the losses. Over the course of training with an SGD-based optimizer, as it is typical for NN-based speech separation, the estimated signal $\hat{\mathbf{s}}$ is pushed into the direction of the negative gradient $-\nabla_{\hat{\mathbf{s}}}\mathcal{L}(\mathbf{s}, \hat{\mathbf{s}}) = -\frac{\partial}{\partial \mathbf{s}}\mathcal{L}(\mathbf{s}, \hat{\mathbf{s}})$, so that the training behavior can be assessed by analyzing the directions of the gradients. The gradient of the

loss ideally points away from $\mathbf{s}$, into the direction of $-(\mathbf{s} - \hat{\mathbf{s}})$. The norm of the gradient vector w.r.t. one estimated speaker signal determines how much the parameter update for the NN will focus on this network output compared to the other outputs. A larger vector norm means that a speaker has a larger impact.

The gradient for the l-th output $\mathbf{s}_l$ of the A-SDR depends only on the l-th output signal:

$$\nabla_{\hat{\mathbf{s}}_l} \mathcal{L}^{\text{A-SDR}}(\mathbf{S}, \hat{\mathbf{S}}) = -\frac{20}{K \ln 10} \frac{\mathbf{s}_l - \hat{\mathbf{s}}_l}{\|\mathbf{s}_l - \hat{\mathbf{s}}_l\|^2}, \tag{4.13}$$

where its norm is proportional to the inverse of the error:

$$\left\| \nabla_{\hat{\mathbf{s}}_l} \mathcal{L}^{\text{A-SDR}}(\mathbf{S}, \hat{\mathbf{S}}) \right\| = \frac{20}{K \ln 10} \frac{1}{\|\mathbf{s}_l - \hat{\mathbf{s}}_l\|} \propto \frac{1}{\|\mathbf{s}_l - \hat{\mathbf{s}}_l\|}. \tag{4.14}$$

The impact of one network output depends inversely on the error, so that the impact of the output with a low error is higher than the impact of output with a high error.

The gradient of the SA-SDR instead depends on all output signals:

$$\nabla_{\hat{\mathbf{s}}_l} \mathcal{L}^{(\text{SA-SDR})}(\mathbf{S}, \hat{\mathbf{S}}) = -\frac{20}{\ln 10} \frac{\mathbf{s}_l - \hat{\mathbf{s}}_l}{\sum_{k=1}^{K} \|\mathbf{s}_k - \hat{\mathbf{s}}_k\|^2}. \tag{4.15}$$

This gradient points into the same direction as $\nabla_{\hat{\mathbf{s}}_l} \mathcal{L}^{(\text{A-SDR})}(\mathbf{S}, \hat{\mathbf{S}})$, but is weighted differently. Looking at the norm of the gradient w.r.t. a single estimated signal $\left\| \nabla_{\hat{\mathbf{s}}_l} \mathcal{L}^{(\text{SA-SDR})}(\mathbf{S}, \hat{\mathbf{S}}) \right\|$ is not insightful. Instead, we look at the ratio of the norms of the gradients w.r.t. two different estimated signals.

One would expect that the norm of the gradients of the output stream with the worse quality are larger, i.e., $\left\| \nabla_{\hat{\mathbf{s}}_k} \mathcal{L}(\mathbf{S}, \hat{\mathbf{S}}) \right\| > \left\| \nabla_{\hat{\mathbf{s}}_l} \mathcal{L}(\mathbf{S}, \hat{\mathbf{S}}) \right\|$ if $\|\hat{\mathbf{s}}_k - \mathbf{s}_k\|^2 > \|\hat{\mathbf{s}}_l - \mathbf{s}_l\|^2$, because a larger error can be corrected on this stream. This is true for the SA-SDR:

$$\frac{\left\| \nabla_{\hat{\mathbf{s}}_k} \mathcal{L}^{(\text{SA-SDR})}(\mathbf{S}, \hat{\mathbf{S}}) \right\|^2}{\left\| \nabla_{\hat{\mathbf{s}}_l} \mathcal{L}^{(\text{SA-SDR})}(\mathbf{S}, \hat{\mathbf{S}}) \right\|^2} = \frac{\|\mathbf{s}_k - \hat{\mathbf{s}}_k\|}{\|\mathbf{s}_l - \hat{\mathbf{s}}_l\|} > 1, \text{ if } \|\hat{\mathbf{s}}_k - \mathbf{s}_k\|^2 > \|\hat{\mathbf{s}}_l - \mathbf{s}_l\|^2. \tag{4.16}$$

But, the A-SDR has the opposite behavior:

$$\frac{\left\| \nabla_{\hat{\mathbf{s}}_k} \mathcal{L}^{(\text{A-SDR})}(\mathbf{S}, \hat{\mathbf{S}}) \right\|^2}{\left\| \nabla_{\hat{\mathbf{s}}_l} \mathcal{L}^{(\text{A-SDR})}(\mathbf{S}, \hat{\mathbf{S}}) \right\|^2} = \frac{\|\mathbf{s}_l - \hat{\mathbf{s}}_l\|}{\|\mathbf{s}_k - \hat{\mathbf{s}}_k\|} < 1, \text{ if } \|\hat{\mathbf{s}}_k - \mathbf{s}_k\|^2 > \|\hat{\mathbf{s}}_l - \mathbf{s}_l\|^2. \tag{4.17}$$

The A-SDR gives a higher weight to the better separated stream while the SA-SDR gives a higher weight to the stream with the larger error. The SA-SDR thus leads to a better balance between the output streams.

4.2.2 Impact on the Weighting Across Examples

The gradients of the A-SDR and SA-SDR in training differ only in the weighting across the system output streams, where different weightings are created by the position of the logarithm in the computation of the loss. This concept can be generalized to other loss

functions as an explanation why logarithmic losses, like the SDR, generally work better for speech separation than linear or quadratic losses, like the Mean Squared Error (MSE)[5].

The logarithm not only has an impact on the weighting across streams, but also on the weighting across examples in a mini-batch and across mini-batches. The scale of the gradient of each example implicitly weights the example in the sum of gradients that accumulates in the parameters over the course of training for an SGD-based optimizer. This even happens when gradient normalization is employed, e.g, for ADAM [48] or RMSProp [91].

In fact, any (scalar) function applied after computing the error cannot change the direction of the gradient w.r.t. the estimate, but only its scale. The scale is determined by the derivative of the scalar function evaluated at the point of the error, for example

$$\nabla_{\hat{\mathbf{s}}} f(\mathcal{L}^{(\mathrm{MSE})}(\mathbf{s}, \hat{\mathbf{s}})) = \left.\frac{\partial f(x)}{\partial x}\right|_{x=\mathcal{L}^{(\mathrm{MSE})}(\mathbf{s},\hat{\mathbf{s}})} \nabla_{\hat{\mathbf{s}}} \mathcal{L}^{(\mathrm{MSE})}(\mathbf{s}, \hat{\mathbf{s}}), \tag{4.18}$$

which gives for the SDR, evaluated for a single speaker,

$$\nabla_{\hat{\mathbf{s}}} \mathcal{L}^{(\mathrm{SDR})}(\mathbf{s}, \hat{\mathbf{s}}) = \underbrace{\frac{10}{\ln 10 \cdot \|\mathbf{s} - \hat{\mathbf{s}}\|^2}}_{\left.\frac{\partial}{\partial x} 10 \log_{10}(x)\right|_{x=\|\mathbf{s}-\hat{\mathbf{s}}\|^2}} \nabla_{\hat{\mathbf{s}}} \mathcal{L}^{(\mathrm{MSE})}(\mathbf{s}, \hat{\mathbf{s}}). \tag{4.19}$$

The result of training with $\mathrm{detach}(\frac{10}{\ln 10 \cdot \|\mathbf{s}-\hat{\mathbf{s}}\|^2}) \mathcal{L}^{(\mathrm{MSE})}(\mathbf{s}, \hat{\mathbf{s}})$, where $\mathrm{detach}(\cdot)$ means that no gradient is propagated through the scaling factor, is the same as training with the SDR loss (SA-SDR, if computing the MSE across all speakers and the A-SDR if computing it for each speaker individually).

When summing over multiple of these terms explicitly (in case of the A-SDR) or implicitly (in the parameter updates of the optimizer), the different scales can indeed modify direction of the final gradient. Concluded from the fact that SDR-based losses outperform the MSE and that the SA-SDR outperforms the A-SDR (as shown in the experiments below), a weighting across examples can be beneficial while different weightings across output streams impact the model performance negatively.

Relation to curriculum learning The logarithm, specifically, scales the gradient inversely proportional to the error. This means that examples with a small error have a higher weight than examples with a large error. This is closely related to curriculum learning [92], where the difficulty of training examples is gradually increased over training. The SDR can thus be interpreted as a form of curriculum learning, where the loss is the MSE, the MSE is taken as a proxy for the example difficulty, and the examples are weighted inversely proportional to their difficulty.

The experiments below indicate that the weighting introduced by the logarithm, where the impact of hard examples that are poorly separated is reduced, is beneficial across examples. Note that both the A-SDR and the SA-SDR have a similar behavior w.r.t. the weighting across examples.

[5] This can be seen from the fact that most publications use a logarithmic loss for training a speech separation systems in the time domain [5–7, 89].

4.3 Scale-invariance

Many works on speech separation use scale-invariant losses, specifically the Scale-Invariant Signal-to-Distortion Ratio (SI-SDR) [5, 62]. Independent of the meeting scenario, it is often argued that the scale of the estimated signal does not contribute to the signal quality and should thus be ignored when computing the loss. The experiments in the remainder of this thesis, however, do not use scale-invariant losses. Here follows an explanation why we refrain from using a scale-invariant loss and why the SI-SDR may not be a good choice for the meeting scenario.

Let us first define how to obtain a scale-invariant version of the SDR. The SDR can be made scale-invariant by scaling either the estimated signal $\hat{\mathbf{s}}$ or the reference signal $\mathbf{s}$ such that their scale matches before computing the loss. The SI-SDR,

$$\mathcal{L}^{(\text{SI-SDR})}(\mathbf{s}, \hat{\mathbf{s}}) = -\text{SI-SDR}(\mathbf{s}, \hat{\mathbf{s}}) = -10 \log_{10} \left(\frac{\|\alpha \mathbf{s}\|^2}{\|\alpha \mathbf{s} - \hat{\mathbf{s}}\|^2} \right), \tag{4.20}$$

scales the source signal $\mathbf{s}$ with the scaling factor α is such that the denominator is minimized, i.e.

$$\alpha = \arg\min_{\tilde{\alpha}} \|\tilde{\alpha} \mathbf{s} - \hat{\mathbf{s}}\|^2 = \frac{\mathbf{s}^\mathsf{T} \hat{\mathbf{s}}}{\mathbf{s}^\mathsf{T} \mathbf{s}}. \tag{4.21}$$

This is the solution of the least-squares optimization for finding α.

Another approach to the scale invariance is the Optimal Scale-Invariant Signal-to-Distortion Ratio (OSI-SDR)[6] [93],

$$\mathcal{L}^{(\text{OSI-SDR})}(\mathbf{s}, \hat{\mathbf{s}}) = -\text{OSI-SDR}(\mathbf{s}, \hat{\mathbf{s}}) = -10 \log_{10} \left(\frac{\left\|\alpha^{(\text{OSI})} \mathbf{s}\right\|^2}{\left\|\alpha^{(\text{OSI})} \mathbf{s} - \hat{\mathbf{s}}\right\|^2} \right), \tag{4.22}$$

$$\alpha^{(\text{OSI})} = \arg\max_{\tilde{\alpha}} \frac{\|\tilde{\alpha} \mathbf{s}\|^2}{\|\tilde{\alpha} \mathbf{s} - \hat{\mathbf{s}}\|^2} = \frac{\mathbf{s}^\mathsf{T} \hat{\mathbf{s}}}{\hat{\mathbf{s}}^\mathsf{T} \hat{\mathbf{s}}}. \tag{4.23}$$

scales the source signal $\mathbf{s}$ such that the overall SDR is maximized. This variant has been used under the name *SI-SDR-SE* in [7].

These losses can be reformulated such that the estimated signal $\hat{\mathbf{s}}$ is scaled instead of the source signal $\mathbf{s}$. Both variants are equivalent. This concept of scale-invariance can also be applied to all other loss functions discussed so far, e.g., the log-MSE.

SI-SDR distorts the gradients of the estimated signal The scale-invariance, as included in the SI-SDR, distorts the direction of the gradients w.r.t. the estimated signal $\hat{\mathbf{s}}$. As it was discussed earlier, the gradient w.r.t. the estimated signal $\hat{\mathbf{s}}$ should point into the negative direction of the error, which is the case for the SDR:

$$\nabla_{\hat{\mathbf{s}}} \mathcal{L}^{(\text{SDR})}(\mathbf{s}, \hat{\mathbf{s}}) = -\frac{20}{\ln 10} \left(\frac{\mathbf{s} - \hat{\mathbf{s}}}{\|\mathbf{s} - \hat{\mathbf{s}}\|^2} \right). \tag{4.24}$$

[6] It is called Optimal Scale-Invariant Signal-to-Noise Ratio (OSI-SNR) in [93]. It is named OSI-SDR here to match the naming of the other loss function presented in this chapter.

From the idea of the SI-SDR, one would expect the same gradients, but with a re-scaled source signal $\alpha\mathbf{s}$ instead of the source signal $\mathbf{s}$. This gradient points into the direction of the error, i.e., it pushes the estimate towards the source signal, and it is zero for samples that are correctly estimated. But because α also depends on $\hat{\mathbf{s}}$, the direction of the gradient of SI-SDR is instead skewed[7]:

$$\nabla_{\hat{\mathbf{s}}}\mathcal{L}^{(\text{SI-SDR})}(\mathbf{s},\hat{\mathbf{s}}) = \nabla_{\hat{\mathbf{s}}}\left(10\log_{10}\|\alpha\mathbf{s}-\hat{\mathbf{s}}\|^2 - 10\log_{10}\left\|\frac{\mathbf{s}^\mathsf{T}\hat{\mathbf{s}}}{\mathbf{s}^\mathsf{T}\mathbf{s}}\mathbf{s}\right\|^2\right) \tag{4.25}$$

$$= -\frac{20}{\ln 10}\left(\frac{\alpha\mathbf{s}-\hat{\mathbf{s}}}{\|\alpha\mathbf{s}-\hat{\mathbf{s}}\|^2} + \frac{\mathbf{s}}{\mathbf{s}^T\hat{\mathbf{s}}}\right). \tag{4.26}$$

An additional term appears whose direction only depends on $\mathbf{s}$, but not on $\hat{\mathbf{s}}$. This means that even if the reconstruction is perfect, i.e., $\alpha\hat{\mathbf{s}} = \mathbf{s}$, and when ignoring the fact that the SI-SDR is undefined for this case, the gradients are non-zero. The OSI-SDR, on the other hand, has the expected gradients:

$$\nabla_{\hat{\mathbf{s}}}\text{OSI-SDR} = \frac{20}{\ln 10}\frac{\alpha^{(\text{OSI})}(\mathbf{s}-\alpha^{(\text{OSI})}\hat{\mathbf{s}})}{\|\mathbf{s}-\alpha^{(\text{OSI})}\hat{\mathbf{s}}\|^2}. \tag{4.27}$$

The effect of this distorted gradient on training is unknown; it may lead to worse performance or the introduced gradient noise may help in escaping local minima during training. It also influences the weighting across examples discussed in the previous sub-section. In Eq. (4.27), the scaling factor β accounts for differences in the scaling of the source signals, which may have an effect on the training, but it is unknown whether this effect is positive or negative.

Scale invariance for meeting separation with stitching The argument that the scale is unimportant may not hold for meeting scenarios where block-wise processing in the stitching process (Section 2.6.2.1) or further processing steps are employed. When using stitching, mismatching scales between adjacent windows introduce audible discontinuities in the stitched signal. Scale differences can also impact the error computation in the alignment process. For follow-up systems, the scale may also be important, e.g., for determining speech activity. We thus do not use it in the remainder of this thesis.

4.4 Experiments

To evaluate the different loss functions, experiments are conducted on utterance-level and meeting-level datasets.

4.4.1 Utterance-level Separation

To show that the SA-SDR does not degrade performance compared to the A-SDR on fully overlapped anechoic data, experiments are conducted on the WSJ0-2mix dataset. The large Dual-Path RNN (DPRNN)-based separator, described in Section A.1.2 is used and trained

[7] Only the α in the numerator is replaced with its definition here to show that the numerator here depends on $\hat{\mathbf{s}}$.

Table 4.1: Separation performance of models trained with A-SDR and SA-SDR on fully overlapped anechoic WSJ0-2mix data. Separation performance is evaluated with CI-SDR.
(Results taken from [OC6])

Loss	CI-SDR↑ in dB
no separation	0.2
A-SDR	17.8
SA-SDR	18.0

on the WSJ0-2mix dataset with uPIT using the different loss functions. The best checkpoint is chosen based on the validation loss. The results shown in Table 4.1 are taken from [OC6].

The model trained with SA-SDR performs slightly better than the model trained with A-SDR, but the performance is overall comparable. This is expected since the scenario is balanced, i.e., there is no silence and the energy and scale of the errors is expected to be similar across the separated streams and target signals for this dataset. Both SA-SDR and A-SDR weight both channels roughly equally in this case (see Section 4.2), which explains the similar performance.

4.4.2 Anechoic Meeting-level Separation With Stitching

The effect of the aggregation in the loss functions becomes visible in imbalanced scenarios. Experiments are conducted on artificial WSJ-based anechoic meetings generated with the MMS-MSG framework. Here, the small DPRNN-based separator, described in Section A.1.2, was used to speed up the experiments. Models were trained for a maximum of 600 000 iterations and the best performing checkpoints were selected based on the validation loss.

The results on the WSJ-based meeting data presented in Table 4.2 are taken from [OC6]. The performance of models trained with different loss functions is evaluated with multiple metrics. The WER and CI-SDR reflect the separation quality for downstream tasks on an utterance level. The attenuation ratio measures how well silence is reconstructed where no speaker is active, the VAER measures how well the speech activity can be predicted from the separated signals, and the SA-SDR measures the signal quality on a meeting level[8]. The column "#spk train" denotes the number of speakers that each model saw during training. For loss functions that permit silent target signals, both single- and two-speaker data was used. For loss functions that do not permit silent target signals, only two-speaker mixtures were used. The upper half shows the averaged loss variants (prefixed with "A-") and the lower half shows the source-aggregated variants (prefixed with "SA-"). The gradients of A-SDR and A-log-MSE only differ in a constant and they show similar performance. The A-log-MSE is slightly worse, perhaps because the log-MSE is not well suited for model selection, as discussed earlier in Chapter 4.

The results show that a stabilization which allows training on single- and multi-speaker examples is beneficial. The A-log1p-MSE improves over the A-logMSE and the A-ε-tSDR improves over the A-tSDR. Training on single-speaker examples including full silence on one

[8] Here, the non-convolutional SDR variant is used for non-reverberant data.

Table 4.2: Comparison of the separation performance of SDR variants on anechoic WSJ-based simulated meeting data. Averaged losses are prefixed with "A-" and source-aggregated losses with "SA-". Best numbers are **bold** and best numbers among conventional averaged SDRs are underlined. *(Adapted from [OC5])*

Loss	#spk train	Utt.-wise Metrics		Meeting-level Metrics		
		WER↓ in %	CI-SDR↑ in dB	att. ratio↑ in dB	VAER↓ in %	SA-SDR↑ in dB
no separation	—	48.1	7.3	0.0	65.6	0.0
A-SDR [5]	2	13.5	19.1	25.5	12.6	13.8
A-log-MSE [89]	2	13.1	19.5	18.3	13.2	14.8
A-log1p-MSE [OC18]	1+2	13.5	19.6	25.3	9.9	16.8
A-skewed-SDR [87]	2	15.6	18.7	24.7	12.5	10.1
A-tSDR [88]	2	13.6	18.8	21.1	14.0	13.3
A-ε-tSDR [OC5]	1+2	12.8	19.6	25.9	11.8	15.5
A-log-tMSE+$\mathcal{L}_0$ [59]	1+2	12.8	19.6	26.4	10.7	14.5
SA-SDR	1+2	12.5	19.8	30.3	9.7	16.1
SA-log-MSE	1+2	13.3	19.3	**31.5**	11.6	14.7
SA-log1p-MSE	1+2	15.1	18.7	25.1	11.4	15.7
SA-skewed-SDR	1+2	15.1	18.6	28.9	12.6	10.6
SA-tSDR	1+2	**12.2**	**19.9**	30.8	**8.2**	**17.9**
SA-ε-tSDR	1+2	12.8	19.6	27.5	9.1	16.3

stream is crucial for silence reconstruction during evaluation, which can be seen in the greatly improved attenuation ratio.

All source-aggregated losses outperform their respective averaged variants. The SA-tSDR achieves the best performance, with a 4 % relative reduction in WER from A-tSDR to SA-tSDR. The modifications that showed benefits in the averaged variants lose their benefit in the source-aggregated variants and degrade the performance here. The SA-log1p-MSE is, for example, worse than the SA-log-MSE and the SA-ε-tSDR is worse than the SA-tSDR.

We also observe that the source-aggregated loss variants yield a smoother and more stable training than the averaged losses. The more uniform weighting across streams seems to have a positive effect on the training behavior.

4.4.3 Reverberant Meeting-level Separation on LibriCSS

A sub-set of the losses are evaluated on the more realistic LibriCSS dataset in Table 4.3. For this experiment, the TF-GridNet-Small network architecture is used, as described in Section A.1.4. Performance was evaluated with the speaker-agnostic tcORC-WER on transcripts predicted with the NeMo [94] ASR model in the `stt_en_conformer_ctc_large` configuration[9]. Despite the increased difficulty, the WERs are overall better than the ones on the WSJ-based meetings due to the greatly improved network architecture for separation

[9] Available on HuggingFace at https://huggingface.co/nvidia/stt_en_conformer_ctc_large

and speech recognition. On this data, the source-aggregated losses again outperform their averaged counterparts, but, contrary to the anechoic case, the threshold in the tSDR does not improve the performance. The relative improvement in WER from the A-tSDR to the SA-tSDR is with 6 % larger than the relative improvement observed on the less realistic anechoic data. This indicates that the effect of the aggregation becomes larger for more realistic data.

The log Mean Absolute Error (log-MAE) was chosen in TS-SEP [36] over the SDR because it led to better silence reconstruction and overall performance. In Table 4.3, however, is not competitive. It only improves the silence reconstruction in the averaged case, but does not translate to the source-aggregated case well. It achieves a worse WER and a comparable attenuation ratio compared to the SA-SDR. This could be explained by the fact that the TS-SEP model outputs significantly more silence than the CSS model, so that the silence reconstruction is more important, while in our case it disturbs the performance.

Table 4.3: Comparison of the separation performance of SDR variants on LibriCSS data. Averaged losses are prefixed with "A-" and source-aggregated losses with "SA-". Best numbers are **bold** and best numbers among conventional averaged SDRs are underlined.

Loss	#spk train	tcORC-WER↓ in %	att. ratio↑ in dB
A-SDR	2	5.4	30.2
A-tSDR	2	5.1	30.7
A-log-MAE	2	22.0	**40.3**
SA-SDR	1+2	**4.8**	32.0
SA-tSDR	1+2	**4.8**	31.8
SA-log-MAE	1+2	5.1	33.1

4.5 Summary

In this section, we have discussed the issues with the commonly used SDR-based loss functions and introduced with the SA-SDR an elegant solution by aggregating signals differently while computing the loss. We have also discussed the relation of the SDR with curriculum learning as a possible explanation for the success of logarithmic losses in speech separation over the MSE. In experiments, we have shown that the SA-SDR outperforms the A-SDR in meeting-level separation tasks, especially when the training data is imbalanced. In the next chapter, the SA-SDR will be used to speed up the computations when solving the assignment problem on the signal level for training a source separator on meeting data directly.

5 Continuous Speech Separation With Graph-PIT

The meeting scenario described in Section 2.1 requires the separation of an arbitrary number of speakers and of arbitrary length. In classical systems, this is achieved with stitching, introduced earlier in Section 2.6.2.1, a block-wise processing scheme that assumes that the number of speakers in each block is small in order to separate a recording containing an overall large number of speakers.

Related own publications

This chapter presents the findings from [OC4, OC5] and [OC3] in a unified way. A short discussion about practical aspects is added in Section 5.5 that was not published before.

CSS can, however, be achieved in a more elegant way, by extending the uPIT training scheme such that it allows an arbitrary number of speakers K, independent of the number of output streams S, as long as never more than S speakers overlap at any point in time. With such a training scheme, a source separator can be trained to directly output overlap-free signals for an arbitrary number of speakers and makes the conventional stitching approach (see Section 2.6.2) unnecessary. We proposed this extension of uPIT as *Graph-PIT* in [OC3, OC5].

In this chapter, the basic idea of Graph-PIT from [OC3, OC5] will be introduced in Section 5.1. Computing the solution naively requires exponential compute with increasing numbers of ground-truth utterances $|\mathcal{U}|$. In order to obtain a more efficient algorithm, it is shown how a loss function can be decomposed to allow for splitting the problem into the computation of a score matrix $\mathbf{M}$ and solving the assignment problem based only on $\mathbf{M}$, for both uPIT and Graph-PIT in Section 5.2. This decomposition scheme was presented in [OC4]. Then, in Section 5.3, an algorithm based on graph coloring is described that solves the assignment problem with a complexity that is linear in the number of utterances, instead of exponential. Finally, the performance of the Graph-PIT training scheme is evaluated in Section 5.4 and practical considerations are discussed in Section 5.5.

5.1 Graph-PIT: Idea

Graph-PIT uses the same assumptions as CSS. It assumes that any separation onto S output streams is valid as long as no two ground-truth utterances overlap on any of these streams and each utterance appears continuously on these streams. Compared to uPIT, the constraint that $K = S$ is lifted to $K^{(\text{sim})} \leqslant S$, where $K^{(\text{sim})}$ is the number of concurrent speakers, i.e., the number of speakers that overlap at any given time. Instead of having a bijective mapping between speakers and output streams, now every utterance is mapped only to a single stream, where two overlapping utterances must be mapped to different streams. Note that two utterances that stem from different speakers can be assigned to the same stream

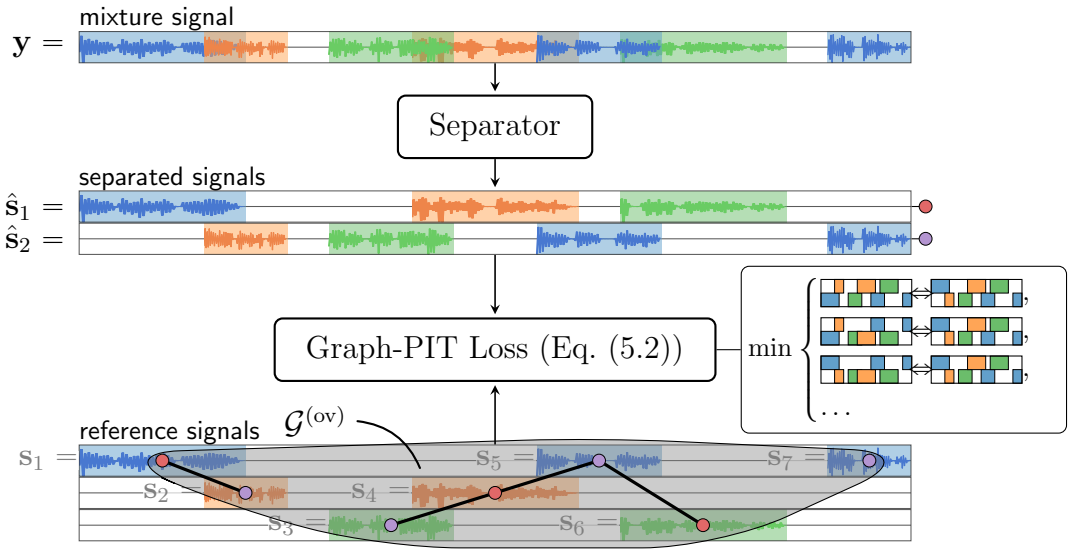


Figure 5.1: Visualization of the Graph-PIT processing scheme. The Graph-PIT loss finds the assignment that minimizes the loss. The graph is drawn on top of the reference signals. The optimal coloring is indicated with the node colors. *(Modified from [OC3])*

as long as they do not overlap and two utterances from the same speaker can be placed on different streams.

To train a neural network with this idea naively, the loss is computed for every eligible assignment $\mathbf{C}$, represented as an assignment matrix, that leads to non-overlapping streams. Then, only the assignment $\mathbf{C}^*$ that minimizes the loss is used for backpropagation:

$$\mathbf{C}^* = \arg\min_{\mathbf{C}\in\mathcal{C}} \mathcal{L}(\mathbf{SC}, \hat{\mathbf{S}}) \tag{5.1}$$

$$\mathcal{L}^{(\text{GraphPIT})}(\mathbf{S}, \hat{\mathbf{S}}) = \mathcal{L}(\mathbf{SC}^*, \hat{\mathbf{S}}) = \min_{\mathbf{C}\in\mathcal{C}} \mathcal{L}(\mathbf{SC}, \hat{\mathbf{S}}). \tag{5.2}$$

The function $\mathcal{L} : \mathbb{R}^{T\times S} \times \mathbb{R}^{T\times S} \to \mathbb{R}$ is a signal-level loss function that produces a scalar from a set of stacked reference and estimated signals (compare Eq. (2.11)). Multiplying the stacked source signals $\mathbf{S} = [\mathbf{s}_1, ..., \mathbf{s}_{|\mathcal{U}|}]^\mathsf{T} \in \mathbb{R}^{T\times|\mathcal{U}|}$ with the assignment matrix $\mathbf{C} = [\mathbf{c}_u]_{u\in\mathcal{U}} \in \mathbb{R}^{|\mathcal{U}|\times S}$ results in a matrix $\mathbf{SC} \in \mathbb{R}^{T\times S}$ of sums of utterance signals. The set

$$\mathcal{C} = \{\mathbf{C} : \mathbf{C} \text{ is an assignment matrix } \wedge \forall u, v : \mathbf{c}_u \neq \mathbf{c}_v \text{ if } o(u, v) = 1\} \tag{5.3}$$

is the set of all valid assignments $\mathbf{C}$ that yield non-overlapping target signals $\mathbf{SC}$, where $o(u, v)$ is the overlap function defined in Eq. (2.1) that is 1 if two utterances overlap and 0 otherwise. All assignment matrices in $\mathcal{C}$ may only assign two utterances u, v onto the same system output stream when they do not overlap, i.e., when $o(u, v) = 0$.

This formulation is similar to uPIT in Eq. (2.9), except that the minimum in Eq. (5.2) is not computed across permutation matrices $\mathbf{P}$ but across assignment matrices $\mathbf{C}$ and that utterance signals in $\mathbf{S}$ are not grouped by speakers.

The Graph-PIT training scheme is visualized in Fig. 5.1, where the best assignment is indicated with colors. A minimum is computed over all possible assignments of reference utterances to system output streams. The figure also shows the overlap graph $\mathcal{G}$, which is introduced below.

5.2 Efficient Decomposition of the Loss Function

The complexity for finding the optimal assignment $\mathbf{C}^*$ naively with Eq. (5.2) is exponential in the number of utterances $|\mathcal{U}|$ because there can be up to $S^{|\mathcal{U}|}$ assignments for $|\mathcal{U}|$ utterances. This complexity can be improved over the brute force variant in Eq. (5.2) if the loss function fulfills specific properties. To arrive at this more efficient formulation, first the simpler case of uPIT is discussed and then extended to Graph-PIT.

5.2.1 Efficient Computation of the Optimal Permutation for uPIT

Given the uPIT criterion with a loss function that operates on stacked signal matrices $\mathbf{S}$ and $\hat{\mathbf{S}}$ from Eq. (2.11), the computation of the optimal permutation matrix $\mathbf{P}^*$ can often be split into two steps in order to reduce the number of times that the loss has to be evaluated. This is the case if the loss function $\mathcal{L}(\mathbf{S}, \hat{\mathbf{S}})$ can be written in the form

$$\mathcal{L}^{(\text{uPIT})}(\mathbf{S}, \hat{\mathbf{S}}) = f\left(\min_{\pi \in \mathcal{P}(K)} \sum_{k=1}^{K} f^{(\text{score})}(\mathbf{s}_{\pi_k}, \hat{\mathbf{s}}_k), \mathbf{S}, \hat{\mathbf{S}}\right) \tag{5.4}$$

$$= f\left(\min_{\mathbf{P} \in \mathcal{P}(K)} \operatorname{tr}\Big(\mathbf{M}^{(\text{uPIT})}\mathbf{P}\Big), \mathbf{S}, \hat{\mathbf{S}}\right), \tag{5.5}$$

where $\operatorname{tr}(\cdot)$ denotes the trace of a (square) matrix, $f(\cdot, \cdot, \cdot)$ is a function strictly monotonically increasing in its first argument and $\mathbf{M}^{(\text{uPIT})} \in \mathbb{R}^{K \times K}$ is a score matrix computed from $\mathbf{S}$ and $\hat{\mathbf{S}}$ as $\mathbf{M}^{(\text{uPIT})} = [f^{(\text{score})}(\mathbf{s}_i, \mathbf{s}_j)]_{i,j}$. The monotony is required to ensure that the inner minimization yields the permutation matrix that minimizes the loss, so that $\min_{\mathbf{P}} f(\operatorname{tr}(\mathbf{MP}), \dots) = f(\arg\min_{\mathbf{P}} \operatorname{tr}(\mathbf{MP}), \dots)$. The function $f(\cdot, \cdot, \cdot)$ is further not allowed to solve the permutation problem in any way on its last two arguments $\mathbf{S}$ and $\hat{\mathbf{S}}$.

Now, first a score matrix $\mathbf{M}^{(\text{uPIT})}$ is computed in quadratic time $\mathcal{O}(K^2)$. Then, $\mathbf{P}^* = \arg\min_{\mathbf{P} \in \mathcal{P}(K)} \mathbf{M}^{(\text{uPIT})}\mathbf{P}$ is found based on $\mathbf{M}^{(\text{uPIT})}$ alone without evaluating $\mathcal{L}$ again. The outer function $f(\cdot, \cdot, \cdot)$ is then only applied once for the optimal permutation.

The minimum operation for finding $\mathbf{P}^*$ is equivalent to a linear sum assignment problem [OC4, 83, 84] with weights $\mathbf{M}^{(\text{uPIT})}$, which can be solved in $\mathcal{O}(K^3)$ polynomial time. The overall complexity can thus be reduced to being polynomial $\mathcal{O}(K^3)$ if the loss function fulfills Eq. (5.5).

Note that the decomposition becomes particularly simple for single-stream signal-level loss functions $\mathcal{L}^{(\text{single})}(\mathbf{s}, \hat{\mathbf{s}})$ from Eq. (2.9), where

$$\mathbf{M}^{(\text{uPIT})} = [\mathcal{L}^{(\text{single})}(\mathbf{s}_i^{(\text{spk})}, \hat{\mathbf{s}}_j)]_{ij}, \tag{5.6}$$

$$f(x, \mathbf{S}^{(\text{spk})}, \hat{\mathbf{S}}) = x. \tag{5.7}$$

This simple decomposition was used in [95] for training uPIT models with up to 20 speakers. In cases where the loss cannot be expressed as a sum over component-wise losses, such as for the SA-SDR introduced earlier in Section 4.2, often a decomposition according to Eq. (5.5) can be found to still compute the permutation efficiently. Examples for such decompositions for commonly used loss functions are presented later in this section, after introducing the decomposition for Graph-PIT.

5.2.2 Efficient Computation of the Optimal Assignment for Graph-PIT

A similar decomposition is possible for Graph-PIT, if the loss can be written as

$$\mathcal{L}^{(\text{Graph-PIT})}(\mathbf{S}, \hat{\mathbf{S}}) = f\left(\min_{\mathbf{C}\in\mathcal{C}} \sum_{s=1}^{S} \sum_{u\in\mathcal{U}} c_{u,s} f^{(\text{score})}(\breve{\mathbf{s}}_\pi, \hat{\mathbf{s}}_k), \mathbf{S}, \hat{\mathbf{S}}\right) \tag{5.8}$$

$$= f\left(\min_{\mathbf{C}\in\mathcal{C}} \operatorname{tr}\Big(\mathbf{M}^{(\text{Graph-PIT})}\mathbf{C}\Big), \mathbf{S}, \hat{\mathbf{S}}\right), \tag{5.9}$$

where the functions $f(\cdot,\cdot,\cdot)$ is again a scalar function that is monotonically increasing in its first argument and $\breve{\mathbf{s}}_u$ is the zero-padded ground-truth source signal for utterance u. Any interactions between the stream signals and ground-truth utterance signals only take place in $f^{(\text{score})}$, so that the optimal assignment $\mathbf{C}^* = \arg\min_{\mathbf{C}\in\mathcal{C}} \operatorname{tr}\Big(\mathbf{M}^{(\text{Graph-PIT})}\mathbf{C}\Big)$ is equal to the one found for the overall loss. Note that the score function $f^{(\text{score})}(\mathbf{s}, \hat{\mathbf{s}})$ operates on the full recording length, but for many loss functions only contributions are relevant where $\breve{\mathbf{s}}_u$ is non-zero, in which case the scores can be computed on parts of the signals only[1]. The indicator variable $c_{u,s} \in \{0, 1\}$ is an element from the assignment matrix $\mathbf{C}$ and indicates whether utterance u is assigned to stream s. As also for uPIT, the elements of the loss matrix $\mathbf{M}^{(\text{Graph-PIT})}$ are the values of $f^{(\text{score})}(\cdot,\cdot)$ and are computed only from pairwise comparisons of the utterance signals $\breve{\mathbf{s}}_u$ with the estimated signals $\hat{\mathbf{s}}_s$.

Compared to Eq. (5.5), the Graph-PIT version only differs in the computation of $\mathbf{M}^{(\text{Graph-PIT})}$. The functions $f(\cdot,\cdot,\cdot)$ and $f^{(\text{score})}(\cdot,\cdot)$ are the same for uPIT and Graph-PIT, i.e., any $f(\cdot,\cdot,\cdot)$ and $f^{(\text{score})}(\cdot,\cdot)$ found for uPIT can also be used for Graph-PIT and vice versa, only their arguments differ according to Eq. (5.5) compared to Eq. (5.9).

Given such a decomposition of a loss function, the assignment can be found based only on a loss matrix $\mathbf{M}^{(\text{Graph-PIT})}$ since minimizing $\operatorname{tr}\Big(\mathbf{M}^{(\text{Graph-PIT})}\mathbf{C}\Big)$ also minimizes $\mathcal{L}$. Finding such a decomposition trivially, i.e., by enumerating all possible assignments, has in general complexity $\mathcal{O}(S^{|\mathcal{U}|})$. This naive approach is infeasible for any non-trivial number of utterances. But, similar to uPIT, a more efficient algorithm exists for finding $\mathbf{C}^*$ based on $\mathbf{M}^{(\text{Graph-PIT})}$. It will be introduced in the next section. But before, decompositions for Graph-PIT with Eq. (5.9) are presented for common loss functions. Additional decompositions for the SA-SI-SDR and the SA-CI-SDR can be found in Section A.3.

5.2.3 Decomposition of Component-wise Additive Functions

Any loss based on a component-wise additive function, i.e., a function for which $g(\mathbf{x}) = \sum_i g(x_i)$ and $g(\mathbf{X}) = \sum_{ij} g(x_{ij})$, can be decomposed for Graph-PIT in the same way. This class of functions includes the core functions of commonly used losses, e.g., the L1-norm from the Mean Absolute Error (MAE) and log-MAE loss and squared L2-norm for the MSE,

[1] One example for this is the MSE, described later in Section 5.2.4.

log-MSE and SDR losses. Such a function $g(\cdot)$ can be decomposed as follows:

$$\mathcal{L}^{(\mathrm{GP}g)}(\mathbf{S}, \hat{\mathbf{S}}) = \min_{\mathbf{C}\in\mathcal{C}} \sum_{s=1}^{S} g\left(\mathbf{S}\mathbf{C} - \hat{\mathbf{S}}\right) \tag{5.10}$$

$$= \min_{\mathbf{C}\in\mathcal{C}} \sum_{s=1}^{S}\sum_{t=1}^{T} g\left(\sum_{u\in\mathcal{U}} c_{u,s} s_{u,t} - \hat{s}_{s,t}\right), \tag{5.11}$$

where the signals from $\mathbf{S}$ and $\hat{\mathbf{S}}$ are decomposed down to a sample level so that $s_{u,t}$ is t-th sample from the padded source signal $\breve{\mathbf{s}}_u$ and $\hat{s}_{s,t}$ is the t-th sample from the estimated stream signal $\hat{\mathbf{s}}_s$. The indicator variable $c_{u,s} \in \{0, 1\}$ is an element from the assignment matrix $\mathbf{C}$ that is one if the u-th utterance is assigned to the s-th stream and zero otherwise.

To get to a formulation that is compatible with the Graph-PIT decomposition in Eq. (5.9), the sum and the multiplication with $c_{u,s}$ has to be moved outside of $g(\cdot)$. It is known from the constraints on the overlap function $o(\cdot)$ that the inner-most sum $\sum_{u\in\mathcal{U}} c_{u,s} s_{u,t}$ contains at most one non-zero element. For all others, either $c_{u,s} = 0$ or $s_{u,t} = 0$ hold true. We can thus write

$$g\left(\sum_{u\in\mathcal{U}} c_{u,s} s_{u,t} - \hat{s}_{s,t}\right) = \begin{cases} g(s_{u',t} - \hat{s}_{s,t}), & \text{if } \exists u' : c_{u',s} s_{u',t} \neq 0 \\ g(-\hat{s}_{s,t}) & \text{otherwise.} \end{cases} \tag{5.12}$$

The first case happens when there is (exactly) one non-zero element in the sum, for index u' and for which $c_{u',s} = 1$. The second case happens when the sum is zero. We can further find that

$$\sum_{u\in\mathcal{U}} c_{u,s} g\left(s_{u,t} - \hat{s}_{s,t}\right) = \begin{cases} g(s_{u',t} - \hat{s}_{s,t}) + g(-\hat{s}_{s,t}) \cdot \left(\sum_{\substack{u\in\mathcal{U}, \\ u\neq u'}} c_{u,s}\right), & \text{if } \exists u' : c_{u',s} s_{u',t} \neq 0 \\ g(-\hat{s}_{s,t}) \cdot \left(\sum_{u\in\mathcal{U}} c_{u,s}\right) & \text{otherwise.} \end{cases} \tag{5.13}$$

$$= g\left(\sum_{u\in\mathcal{U}} c_{u,s} s_{u,t} - \hat{s}_{s,t}\right) + g(-\hat{s}_{s,t}) \cdot \left(\sum_{u\in\mathcal{U}} c_{u,s} - 1\right), \tag{5.14}$$

where the conditions of the cases are equal to those in Eq. (5.12) and it was used that $\sum_{\substack{u\in\mathcal{U}, \\ u\neq u'}} c_{u,s} = \sum_{u\in\mathcal{U}} c_{u,s} - 1$. We can now plug Eq. (5.14) into Eq. (5.11) to achieve a formulation where the sum and the multiplication with $c_{u,s}$ are outside the application of $g(\cdot)$:

$$\mathcal{L}^{(\mathrm{GP}g)}(\mathbf{S}, \hat{\mathbf{S}}) = \min_{\mathbf{C}\in\mathcal{C}} \sum_{s=1}^{S}\sum_{t=1}^{T} \left(\sum_{u\in\mathcal{U}} c_{u,s} g\left(s_{u,t} - \hat{s}_{s,t}\right) - g\left(-\hat{s}_{s,t}\right) \cdot \left(\sum_{u\in\mathcal{U}} c_{u,s} - 1\right)\right) \tag{5.15}$$

$$= \sum_{s=1}^{S}\sum_{t=1}^{T} g\left(-\hat{s}_{s,t}\right) + \min_{\mathbf{C}\in\mathcal{C}} \sum_{s=1}^{S}\sum_{u\in\mathcal{U}} c_{u,s} \sum_{t=1}^{T} \left(g\left(s_{u,t} - \hat{s}_{s,t}\right) - g(-\hat{s}_{s,t})\right) \tag{5.16}$$

$$= g\left(\hat{\mathbf{S}}\right) + \min_{\mathbf{C}\in\mathcal{C}} \sum_{s=1}^{S}\sum_{u\in\mathcal{U}} c_{u,s} \underbrace{\left(g\left(\breve{\mathbf{s}}_u - \hat{\mathbf{s}}_s\right) - g(-\hat{\mathbf{s}}_s)\right)}_{f^{(\mathrm{score})}(\breve{\mathbf{s}}_u, \hat{\mathbf{s}}_s)}. \tag{5.17}$$

The loss matrix $\mathbf{M} = \left[g\left(\breve{\mathbf{s}}_u - \hat{\mathbf{s}}_s\right) - g(-\hat{\mathbf{s}}_s)\right]_{u,s}$ is defined by the inner-most sum of the second term, $f^{(\mathrm{score})}(\breve{\mathbf{s}}_u, \hat{\mathbf{s}}_s) = g\left(\breve{\mathbf{s}}_u - \hat{\mathbf{s}}_s\right) - g(-\hat{\mathbf{s}}_s)$. The outer function $f(x, \mathbf{S}, \hat{\mathbf{S}}) = x + g(\hat{\mathbf{S}})$ from Eq. (5.9) can be found by looking at the surrounding expressions in Eq. (5.17).

Applied to the MAE with $g(x) = \|x\|$, this approach yields

$$\mathcal{L}^{(\text{GPMAE})}(\mathbf{S}, \hat{\mathbf{S}}) = \left\|\hat{\mathbf{S}}\right\|_1 + \min_{\mathbf{C}\in\mathcal{C}} \sum_{s=1}^{S} \sum_{u\in\mathcal{U}} c_{u,s} \left(\|\breve{\mathbf{s}}_u - \hat{\mathbf{s}}_s\|_1 - \|\hat{\mathbf{s}}_s\|_1\right) \tag{5.18}$$

and the SA-log-MAE can be obtained with[2].

$$\mathcal{L}^{(\text{GPSA-log-MAE})}(\mathbf{S}, \hat{\mathbf{S}}) = \log_{10} \mathcal{L}^{(\text{GPMAE})}(\mathbf{S}, \hat{\mathbf{S}}). \tag{5.19}$$

While this approach can be used to decompose a wide variety of loss functions, it is not always the most efficient one. The following subsections present more efficient decompositions for quadratic loss functions like the MSE, log-MSE and SDR and for the Binary Cross Entropy (BCE).

5.2.4 Decomposition of Quadratic Loss Functions

Loss functions where the inner term is quadratic, such at the MSE, log-MSE and SA-SDR, can be decomposed more elegantly. The MSE is here presented as the base of the decompositions for the other loss functions. The MSE is defined for a single pair of signals $\mathbf{s}_u$ and $\hat{\mathbf{s}}_u$ as

$$\mathcal{L}^{(\text{MSE})}(\mathbf{s}_u, \hat{\mathbf{s}}_u) = \|\mathbf{s}_u - \hat{\mathbf{s}}_u\|^2 = (\mathbf{s} - \hat{\mathbf{s}})^\mathsf{T}(\mathbf{s} - \hat{\mathbf{s}}), \tag{5.20}$$

where the squared norm is written as a scalar product to simplify further derivations. Applied to Graph-PIT (Eq. (5.2)), the loss becomes

$$\begin{aligned}
\mathcal{L}^{(\text{GPMSE})}(\mathbf{S}, \hat{\mathbf{S}}) &= \min_{\mathbf{C}\in\mathcal{C}} \operatorname{tr}\Big((\mathbf{S}\mathbf{C} - \hat{\mathbf{S}})^\mathsf{T}(\mathbf{S}\mathbf{C} - \hat{\mathbf{S}})\Big) && (5.21)\\
&= \min_{\mathbf{C}\in\mathcal{C}} \left(\operatorname{tr}\big(\mathbf{C}^\mathsf{T}\mathbf{S}^\mathsf{T}\mathbf{S}\mathbf{C}\big) - \operatorname{tr}\Big(\hat{\mathbf{S}}^\mathsf{T}\mathbf{S}\mathbf{C}\Big) - \operatorname{tr}\Big(\mathbf{C}^\mathsf{T}\mathbf{S}^\mathsf{T}\hat{\mathbf{S}}\Big) + \operatorname{tr}\Big(\hat{\mathbf{S}}^\mathsf{T}\hat{\mathbf{S}}\Big)\right) && (5.22)\\
&= \operatorname{tr}\big(\mathbf{S}^\mathsf{T}\mathbf{S}\big) + \operatorname{tr}\Big(\hat{\mathbf{S}}^\mathsf{T}\hat{\mathbf{S}}\Big) + 2\min_{\mathbf{C}\in\mathcal{C}} \operatorname{tr}\left(\underbrace{-\hat{\mathbf{S}}^\mathsf{T}\mathbf{S}}_{\mathbf{M}^{(\text{Graph-PIT})}}\ \mathbf{C}\right). && (5.23)
\end{aligned}$$

Here, it was used that $\operatorname{tr}\big(\mathbf{S}^\mathsf{T}\mathbf{S}\big) = \sum_{k=1}^{K} \|\mathbf{s}_k\|^2$ and that $\operatorname{tr}\big(\mathbf{S}^\mathsf{T}\big) = \operatorname{tr}(\mathbf{S})$. The assignment matrix $\mathbf{C}$ can be dropped in $\operatorname{tr}\big(\mathbf{C}^\mathsf{T}\mathbf{S}^\mathsf{T}\mathbf{S}\mathbf{C}\big)$ because it only maps non-overlapping utterances to the same stream such that every sample appears exactly once in the trace. From Eq. (5.23), we find

$$\begin{aligned}
f^{(\text{score,MSE})}(\mathbf{s}_u, \hat{\mathbf{s}}_s) &= \breve{\mathbf{s}}_u^\mathsf{T}\hat{\mathbf{s}}_s, && (5.24)\\
f^{(\text{MSE})}(x, \mathbf{S}, \hat{\mathbf{S}}) &= \operatorname{tr}\big(\mathbf{S}^\mathsf{T}\mathbf{S}\big) + \operatorname{tr}\Big(\hat{\mathbf{S}}^\mathsf{T}\hat{\mathbf{S}}\Big) + 2x. && (5.25)
\end{aligned}$$

Note that in $f^{(\text{score,MSE})}$, only samples from $\hat{\mathbf{s}}_s$ have a contribution where $\breve{\mathbf{s}}_u$ is non-zero, so that the score can be computed from the signal parts defined by the segment start and end times b_u and e_u without changing its value.

[2] Or alternatively by moving the logarithm also into the computation of $\mathbf{M}^{(\text{Graph-PIT})}$, but Eq. (5.19) applies the logarithm less often

SA-log-MSE The SA-log-MSE is similar to the MSE:

$$\mathcal{L}^{(\text{GP-SA-log-MSE})}(\mathbf{S}, \hat{\mathbf{S}}) = \log_{10} \mathcal{L}^{(\text{GPMSE})}(\mathbf{S}, \hat{\mathbf{S}}). \tag{5.26}$$

SA-SDR The SDR, in its standard definition, cannot be decomposed for Eq. (5.9), because order of the logarithm and the sum disallows such a rearrangement. The SA-SDR,

$$\mathcal{L}^{(\text{GP-SA-SDR})} = \min_{\mathbf{C}\in\mathcal{C}} -10 \log_{10} \frac{\operatorname{tr}\big(\mathbf{C}^\mathsf{T}\mathbf{S}^\mathsf{T}\mathbf{S}\mathbf{C}\big)}{\operatorname{tr}\Big((\mathbf{S}\mathbf{C} - \hat{\mathbf{S}})^\mathsf{T}(\mathbf{S}\mathbf{C} - \hat{\mathbf{S}})\Big)} \tag{5.27}$$

where the sum is moved inside the logarithm, can however be decomposed. The denominator in Eq. (5.27) is equal to the MSE in Eq. (5.23), so the MSE decomposition from Eq. (5.23) can be re-used:

$$\begin{aligned}
\mathcal{L}^{(\text{GP-SA-SDR})} &= 10 \log_{10} \mathcal{L}^{(\text{GPMSE})}(\mathbf{S}, \hat{\mathbf{S}}) - 10 \log_{10} \operatorname{tr}\big(\mathbf{S}^\mathsf{T}\mathbf{S}\big) && (5.28)\\
&= 10 \log_{10} \left(\operatorname{tr}\Big(\hat{\mathbf{S}}^\mathsf{T}\hat{\mathbf{S}}\Big) + 2 \min_{\mathbf{C}\in\mathcal{C}} \operatorname{tr}\Big(-\hat{\mathbf{S}}^\mathsf{T}\mathbf{S}\mathbf{C}\Big) \right) - 10 \log_{10} \left(\operatorname{tr}\big(\mathbf{S}^\mathsf{T}\mathbf{S}\big) + 1 \right). && (5.29)
\end{aligned}$$

The resulting expression for the loss matrix $\mathbf{M}^{(\text{Graph-PIT})}$ is equal to the MSE, so the assignment that minimizes the MSE also minimizes the negative SA-SDR. The outer function becomes $f(x, \mathbf{S}, \hat{\mathbf{S}}) = 10 \log_{10}(\operatorname{tr}\Big(\hat{\mathbf{S}}^\mathsf{T}\hat{\mathbf{S}}\Big) + 2x) - 10 \log_{10}(\operatorname{tr}\big(\mathbf{S}^\mathsf{T}\mathbf{S}\big) + 1)$.

5.2.5 Decomposition of Binary Cross Entropy

The BCE loss was used in [OC9] for training a neural diarization system with Graph-PIT. It is mentioned here for completeness, but not used in the remainder of this thesis. The BCE loss between the ground truth binary labels $a_1, ..., a_T$ and the estimation $\hat{a}_1, ..., \hat{a}_T$ is defined as

$$\mathcal{L}^{(\text{BCE})}(a, \hat{a}) = -\sum_{t=1}^{T} (a_t \log \sigma(\hat{a}_t) + (1 - a_t) \log(1 - \sigma(\hat{a}_t))), \tag{5.30}$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the sigmoid function, log is the natural logarithm and and t is the sample index. Eq. (5.30) can be simplified by plugging in the definition of the sigmoid:

$$\mathcal{L}^{(\text{BCE})}(a, \hat{a}) = \sum_{t=1}^{T} (\hat{a}_t + \log(1 + \exp(-\hat{a}_t))) - \sum_{t=1}^{T} a_t \hat{a}_t, \tag{5.31}$$

such that only one term depends on the ground truth activity a.

To apply Graph-PIT to the BCE loss, we again collect ground-truth targets across utterances in a matrix $\mathbf{A} = [a_{tu}]_{tu} \in \{0, 1\}^{T \times |\mathcal{U}|}$, zero-padded to match the full meeting length, similar to the speech signal matrix $\mathbf{S}$ for speech separation. The estimated activity is also collected into a matrix $\hat{\mathbf{A}} \in \mathbb{R}^{T \times S}$. Plugged into Eq. (5.2), this leads to

$$\mathcal{L}^{(\text{GPBCE})}(\mathbf{A}, \hat{\mathbf{A}}) = \sum_{t=1}^{T} (\hat{a}_t + \log(1 + \exp(-\hat{a}_t))) + \min_{\mathbf{C}\in\mathcal{C}(\mathcal{U})} \operatorname{tr}\Big(-\hat{\mathbf{A}}^\mathsf{T}\mathbf{A}\mathbf{C}\Big). \tag{5.32}$$

The minimum is only applied to the second term because the first term does not depend not on a. When implementing, the log-sum-exp expression in first term should be stabilized accordingly in terms of numerical effects.

The assignment matrix is, interestingly, the same as for the MSE in Eq. (5.23) but based on the activity instead of the speech signals.

5.3 Graph-PIT as a Vertex-Weighted Graph Coloring Problem

Finding the optimal assignment $\mathbf{C}^*$ is equivalent to a graph coloring problem, or more specifically, a *vertex-weighted graph coloring* problem, of the *overlap graph*, which is a graph that describes which utterances overlap in the ground-truth annotations. This section first introduces the necessary notation from graph theory, then describes the relation of Graph-PIT to graph coloring, and finally presents a dynamic programming algorithm that solves the vertex-weighted graph coloring problem for Graph-PIT efficiently.

5.3.1 Graph Theory

This section gives a quick overview of the graph concepts and notation required for understanding the remaining part of this chapter. It is by no means a general comprehensive description of graphs. The notation is kept consistent with the rest of this thesis and may not always be aligned with the notation found in the literature on graph theory.

5.3.1.1 Undirected Graphs

An (undirected cycle-free) Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a structure consisting of a set of vertices $\mathcal{V}$ (consisting of arbitrary elements) and a set of edges $\mathcal{E} \subseteq \{\{u, v\} : u, v \in \mathcal{V}, u \neq v\}$ that connect pairs of vertices. Vertices that are connected by an edge are called adjacent. The set $N_{\mathcal{G}}[v] = \{u : u \in \mathcal{V}, \exists e \in \mathcal{E} : u \in e\}$ is the set of neighbors of vertex v in $\mathcal{G}$, i.e., the set of vertices that are connected to v via an edge $e \in \mathcal{E}$. The subscript $\mathcal{G}$ is omitted if it is clear from the context. We denote with $\mathcal{G}[\mathcal{V}'] = (\mathcal{V}', \{\{u, v\} : \{u, v\} \in \mathcal{E}, u, v \in \mathcal{V}'\})$ the subgraph induced by $\mathcal{V}' \subseteq \mathcal{V}$ in $\mathcal{G}$, i.e., the graph that contains all vertices in $\mathcal{V}'$ and all edges from $\mathcal{G}$ that connect two vertices in $\mathcal{V}'$. A clique is a fully connected (sub)graph, i.e., a (sub)graph where all vertices are connected to each other. The clique number ω of a graph is the size of the largest clique in the graph.

5.3.1.2 Graph Vertex Coloring

The graph vertex coloring problem is concerned with finding an assignment (see Section 2.4) $\mathbf{C} \in \{0, 1\}^{|\mathcal{V}| \times S}$ that assigns each vertex in $\mathcal{V}$ a unique label (commonly called *color*) in $\{1, 2, \ldots, S\}$ under the condition that no two adjacent vertices have the same color. The coloring is here represented as an assignment matrix $\mathbf{C}$ with rows $\mathbf{c}_1, ..., \mathbf{c}_{|\mathcal{U}|}$ and elements $\mathbf{c}_u = [c_{u,1}, ..., c_{u,S}]$, where each row is a one hot vector such that $c_{u,s} = 1$ if utterance u is assigned color s and otherwise $c_{u,s} = 0$. Such an assignment $\mathbf{C}$ that uses S colors and does not assign two adjacent vertices the same color is called a *proper (S-)coloring* of the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and satisfies

$$\mathbf{c}_u \neq \mathbf{c}_v \quad \forall \{u, v\} \in \mathcal{E}. \tag{5.33}$$

The number S is the number of colors used in the coloring and a graph that can be properly colored with k colors (or less) is called k-colorable.

5.3.2 Vertex-Weighted (Vertex) Graph Coloring

We here extend the graph coloring problem to the *vertex-weighted (vertex) graph coloring* problem where each combination of color and vertex receives a weight and the goal is to find a proper coloring that minimizes the total weight of the coloring. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a number of colors S, and a weight function $w : \mathcal{V} \times \{0,1\}^S \to \mathbb{R}$ that assigns each combination of vertex and color a weight, the objective is to find the proper coloring $\mathbf{C}^*$ of $\mathcal{G}$ that uses at most S colors and minimizes the total weight, i.e.

$$\mathbf{C}^* = \arg\min_{\mathbf{C}} w(\mathbf{C}) = \arg\min_{\mathbf{C}} \sum_{v \in \mathcal{V}} w(v, \mathbf{c}_v), \quad \text{s.t. } \mathbf{C} \text{ is proper}, \tag{5.34}$$

where $w(\mathbf{C}) = \sum_{v \in \mathcal{V}} w(v, \mathbf{c}_v)$ denotes the total cost of the coloring $\mathbf{C}$ given w.

Graph-PIT is equivalent to the vertex-weighted graph coloring problem of the overlap-graph $\mathcal{G}^{(\text{ov})}(\mathcal{U})$, which we define for a set of utterances $\mathcal{U}$ as the graph

$$\mathcal{G}^{(\text{ov})}(\mathcal{U}) = (\mathcal{U}, \{\{u, v\} : o(u, v) = 1, \forall u, v \in \mathcal{U}\}), \tag{5.35}$$

that contains all utterances as vertices and edges between two utterances if they overlap. Here, $o(\cdot)$ is the overlap function introduced in Section 2.2 that is 1 if two utterances overlap and 0 otherwise. The argument $(\mathcal{U})$ is omitted where possible.

Fig. 5.1 visualizes such an overlap graph for an example. Each utterance in $\mathcal{U}$ is a vertex in $\mathcal{G}^{(\text{ov})}$, drawn as a circle in Fig. 5.1. An edge exists between two vertices in $\mathcal{G}^{(\text{ov})}$ when the corresponding utterances overlap in time. Edges are drawn as connecting thick lines between the vertices. The optimal coloring of $\mathcal{G}^{(\text{ov})}$ according to $\mathbf{C}^*$ in Eq. (5.34) is marked by the colors of the vertices. Each color corresponds to a system output stream and the coloring of a vertex represents the assignment of that utterance signal $\mathbf{s}_u$ to a network output stream $\hat{\mathbf{s}}_s$. The process of finding $\mathbf{C}^*$ with Eq. (5.34) is indicated by the minimum over a list of possible assignments in the figure.

The condition for a valid assignment in Eq. (5.3) is equivalent to the condition for a proper coloring in Eq. (5.33). Thus, an assignment $\mathbf{C}$ of utterances $\mathcal{U}$ to S output streams is a valid if and only if it is a S-coloring of the overlap graph $\mathcal{G}^{(\text{ov})}$, where the colors represent which stream an utterance is assigned to. The set of valid assignments $\mathcal{C}(\mathcal{U})$ is also the set of all proper S-colorings of $\mathcal{G}^{(\text{ov})}$.

5.3.2.1 Relation to Other Problems and Algorithms

The vertex-weighted graph coloring problem collapses to the minimum sum assignment problem [83] if the overlap graph is fully connected, i.e., if every vertex is connected to every other vertex. Then, any assignment is just a permutation of colors across vertices and it can be solved in polynomial time.

A similarly named problem is the cost vertex coloring problem [96], also called minimum sum coloring problem (MSCP) [97] or Disfavored Color or Favored Color Weighted Graph Coloring [98], assigns each color c (instead of vertex color pair) a weight w_c. The goal is to find a proper coloring that minimizes the sum of the weights of the colors assigned to the vertices, i.e., $\min_c \sum_{v \in \mathcal{V}} w_{c_v}$. This is similar to the vertex-weighted graph coloring problem, but the weight depends only on the color instead of both the vertex and the color.

The coloring at minimum cost [99] also has a similar name, but defines the costs as the difference between the colors of two adjacent vertices.

5.3.2.2 Complexity Bounds

The vertex-weighted graph coloring problem is in general NP-hard. Finding a single proper coloring is already NP-hard in general [100] and the vertex-weighted graph coloring problem is harder in that the solution is not any single proper coloring, but the one that at the same time minimizes the cost. The number of proper s-colorings of a graph is in general exponential. This can be seen for the example of a graph with no edges: Every vertex can be colored with every of the s colors, resulting in $s^{|\mathcal{V}|}$ possible proper colorings.

When the graph has a special structure, such as the overlap graph $\mathcal{G}^{(\mathrm{ov})}(\mathcal{U})$, the complexity can be reduced. It is shown that an overlap graph is a *chordal* graph in Section A.2. For chordal graphs, the (standard) graph coloring problem can be solved in polynomial time [101]. This is an indication that the vertex-weighted graph coloring problem for the overlap graph may also be solvable in polynomial time. That this is the case will be shown below when the complexity of the presented algorithm is analyzed.

5.3.3 A Dynamic Programming Algorithm for Solving the Vertex-Weighted Graph Coloring Problem

In this section, first an algorithm is presented that solves the vertex-weighted graph coloring problem, i.e., finding the optimal coloring $\mathbf{C}^*$ with the lowest cost according to Eq. (5.34) for a general graph. It is then shown that the complexity becomes polynomial in the number of segments given the special structure of an overlap graph.

For the development of the algorithm, the following additional notation is needed. We call two colorings *compatible* on a set of vertices $\mathcal{V}'$ if they yield the same coloring on the induced subgraph $\mathcal{G}[\mathcal{V}']$. This is denoted by $\mathbf{C}_1||_{\mathcal{V}'}\mathbf{C}_2$ if $\forall u \in \mathcal{V}' : \mathbf{c}_{u,1} = \mathbf{c}_{u,2}$. The set of all proper colorings of a set of vertices given a graph is denoted by $\mathcal{C}_{\mathcal{G}}(\mathcal{V})$, similar to Eq. (5.3).

Let

$$\mathcal{Q}^{\mathrm{all}} = \mathcal{C}_{\mathcal{G}}(\mathcal{V}) \tag{5.36}$$

be the set of all proper colorings of the vertices $\mathcal{V}$ with up to S colors, such that

$$\mathbf{C}^* = \underset{\mathbf{C}\in\mathcal{Q}^{\mathrm{all}}}{\arg\min} \sum_{v\in\mathcal{V}} w(v, \mathbf{c}_v) \tag{5.37}$$

is the solution to the vertex-weighted graph coloring problem in Eq. (5.34), where $\mathbf{c}_v$ is one column of $\mathbf{C}$. This set $\mathcal{Q}^{\mathrm{all}}$ can be constructed iteratively by traversing the graph in any order, starting with the empty set for $\mathcal{Q}_0 = \varnothing$ and extending the colorings of the already visited vertices by all possible colors for the current vertex with

$$\mathcal{Q}_u = \left\{ \begin{bmatrix} \mathbf{C} \\ \mathbf{e}_s^{(S)\mathsf{T}} \end{bmatrix} : \mathbf{C} \in \mathcal{Q}_{u-1}, s \in \{1, ..., S\}, \forall v \in N_{\mathcal{G}[\{1,...,u\}]}[u] : \mathbf{e}_s^{(S)\mathsf{T}} \neq \mathbf{c}_v \right\} \tag{5.38}$$

when visiting u, and $\mathcal{Q}^{\mathrm{all}} = \mathcal{Q}_{|\mathcal{U}|}$. The one-hot row vector $\mathbf{e}_s^{(S)\mathsf{T}}$ represents the color s and is appended to the bottom of the previous coloring $\mathbf{C}$. This set grows in general exponentially with the number of vertices.

It can be shown by induction that this constructs the full set of proper colorings $\mathcal{Q}_{|\mathcal{U}|} = \mathcal{C}_{\mathcal{G}}(\mathcal{V})$, or $\mathcal{Q}_u = \mathcal{C}_{\mathcal{G}}(\{1, ..., u\})$. The starting condition $\mathcal{Q}_1 = \{1, ..., S\} = \mathcal{C}_{\mathcal{G}}(\{1\})$ is true

because a single vertex is not constrained to a color but can be colored with any of the S available colors. When adding a vertex to the coloring with Eq. (5.38), the new vertex gets assigned all possible colors, i.e., any other color would result in an improper coloring. Thus, $\mathcal{Q}_{u+1} = \mathcal{C}_\mathcal{G}(\{1, ..., u+1\})$.

Since the goal of the optimization is to find the coloring with the smallest cost, elements can be removed that will not influence the choice of colors for future vertices while traversing the graph from $u = 1$ to $u = |\mathcal{U}|$. Any already visited vertices that have no neighbors later than the current vertex are irrelevant for the coloring of the remaining graph. Only the vertices in

$$\mathcal{F}_u = \{v : v < u \wedge N[v] \backslash \{1, ..., u-1\} \neq \varnothing\} \tag{5.39}$$

are relevant for coloring u or any later vertex, so that

$$\mathcal{Q}'_u = \left\{ \underset{\mathbf{C} \in \mathcal{Q}_{u-1}, s \in \{1,...,S\}: [\mathbf{C}^\mathsf{T}, \mathbf{e}_s^{(S)}]^\mathsf{T} || _{\mathcal{F}_u} \mathbf{C}'}{\arg\min} w([\mathbf{C}^\mathsf{T}, \mathbf{e}_s^{(S)}]^\mathsf{T}) : \mathbf{C}' \in \mathcal{C}_\mathcal{G}(\mathcal{F}_u) \right\} \tag{5.40}$$

are the relevant colorings. This set creates for each coloring $\mathbf{C}'$ of the relevant vertices in $\mathcal{F}_u$ all compatible colorings based on the colorings in $\mathcal{Q}_{u-1}$ from the previous step, and keeps only the one with the minimal cost. The size $|\mathcal{Q}'_u|$ is determined by the number of proper colorings of the subgraph $\mathcal{G}[\mathcal{F}_u]$ induced by $\mathcal{F}_u$.

Using Eq. (5.37) on $\mathcal{Q}'_{|\mathcal{U}|}$ leads to the same optimal solution as on $\mathcal{Q}^\text{all}$. This is true because the minimum operation in Eq. (5.40) will never remove the optimal coloring, which can be shown by contradiction. Let $\mathbf{C}^{(\text{opt})}$ be the optimal coloring of $\mathcal{U}$. If the partial coloring that leads to the optimal coloring would not be part of $\mathcal{Q}'$, i.e., $\mathbf{C}^{(\text{opt})}[\{1, ..., u\}] \notin \mathcal{Q}'_u$ for any $u \in \mathcal{U}$, then there would exist a coloring $\mathbf{C}$ of $\{1, ..., u\}$ with $\mathbf{C}[\mathcal{F}_u] = \mathbf{C}^{(\text{opt})}[\mathcal{F}_u]$ and $\mathbf{C}[\{1, ..., u\} \backslash \mathcal{F}_u] \neq \mathbf{C}^{(\text{opt})}[\{1, ..., u\} \backslash \mathcal{F}_u]$ and, because of the additivity of the costs, $\sum_{v \in \{1,...,u\} \backslash \mathcal{F}_u} w(v, \mathbf{c}_v^{(\text{opt})}) > \sum_{v \in \{1,...,u\} \backslash \mathcal{F}_u} w(v, \mathbf{c}_v)$, so that $\mathbf{C}^{(\text{opt})}$ disappears in the minimum operation. But since the coloring of $v > u$ is independent of the coloring of $\{1, ..., u\} \backslash \mathcal{F}_u$, and a $\mathbf{C}$ is better on $\{1, ..., u\} \backslash \mathcal{F}_u$ than $\mathbf{C}^{(\text{opt})}$, the coloring $\mathbf{C}^{(\text{opt})}$ would not be optimal anymore, which is a contradiction.

For overlap graphs If $\mathcal{G} = \mathcal{G}^{(\text{ov})}(\mathcal{U})$, i.e., $\mathcal{G}$ is the overlap graph of the utterances $\mathcal{U}$, the computation of the set $\mathcal{F}_u$ is simplified. Let us first assume that the utterances $u \in \mathcal{U}$ are ordered by their begin time, such that $u < u' \Leftrightarrow b_u \leqslant b_{u'}$. The overlap graph is defined by the utterances and the overlap function $o(u, u')$, which has the following property (see Section 2.2), for the three utterances[3] $v, u, w \in \mathcal{U}$:

$$v < u < w \wedge o(v, w) = 1 \Rightarrow o(v, u) = 1. \tag{5.41}$$

From the definition of the overlap graph, we know that $o(v, u) = 1 \Rightarrow v \in N[u] \wedge u \in N[v]$, so that this property can be rewritten as:

$$v < u < w \wedge w \in N[v] \Rightarrow u \in N[v]. \tag{5.42}$$

[3] The names are deliberately chosen to be different from the alphabet such that they match the definition of $\mathcal{F}_u$, where $v < u$.

From the definition of $\mathcal{F}_u$, let now $w \in N[v] \backslash \{1, ..., u-1\}$. It then follows for $\mathcal{F}_u$, using Eq. (5.42):

$$\mathcal{F}_u = \{v : v < u \wedge \exists w \geqslant u : w \in N[v]\} \tag{5.43}$$
$$= \{v : v < u \wedge v \in N[u]\}. \tag{5.44}$$

When $\mathcal{G}$ is an overlap graph, the set $\mathcal{F}_u$ is thus the sub-set of the neighborhood of u that has already been visited. The algorithm is outlined in Algorithm 3 using pseudo code for the case when $\mathcal{G}$ is an overlap graph.

Algorithm 3 Dynamic programming algorithm for Graph-PIT

for $u = 1$ to $|\mathcal{U}|$ **do**
  ▷ *Find all neighbors of the current utterance that are already visited* ◁
  $\mathcal{F}_u \leftarrow \{v : v \in N[u] \wedge v < u\}$
  ▷ *The two loops below iterate in total exactly once over $\mathcal{Q}'_{u-1}$ because the compatibility condition partitions the colorings into disjoint sets.* ◁
  for $\mathbf{C}' \in \mathcal{C}_\mathcal{G}(\mathcal{F}_u)$ **do**
    for $\mathbf{C} \in \mathcal{Q}'_{u-1}$ if $\mathbf{C}||_{\mathcal{F}_u}\mathbf{C}'$ **do**
      $\mathbf{C}^{(\text{best})} \leftarrow$ null
      for $s \in \{s : 1 \leqslant s \leqslant S \wedge \mathbf{e}_s^{(S)} \notin \mathbf{C}[\mathcal{F}_u]\}$ **do**
        if $w([\mathbf{C}^\mathsf{T}, \mathbf{e}_s^{(S)}]^\mathsf{T}) < w(\mathbf{C}^{(\text{best})})$ **then**
          $w^{(\text{best})} \leftarrow [\mathbf{C}^\mathsf{T}, \mathbf{e}_s^{(S)}]^\mathsf{T}$
      $\mathcal{Q}_u \leftarrow \mathbf{C}^{(\text{best})}$
return $\mathbf{C}^* = \arg\min_{\mathbf{C} \in \mathcal{Q}'_{|\mathcal{U}|}} w(\mathbf{C})$

Complexity The algorithm, as expected for an NP-hard problem, has an exponential complexity for arbitrary graphs $\mathcal{G}$ because no guarantees can be made for the size of $\mathcal{F}_u$, which is only trivially bound by u. Subsequently the size of $\mathcal{Q}'_u$ may grow exponentially with u. One trivial optimization is to solve the assignment problem for each connected component of the graph $\mathcal{G}$, i.e., for different sets of vertices that are not connected by paths through $\mathcal{G}$, independently. The complexity then only depends on the number of utterances within a connected component instead of the total number of utterances.

The problem is, however, not NP-hard if the graph is an overlap graph, as already indicated earlier. If $\mathcal{G} = \mathcal{G}^{(\text{ov})}(\mathcal{U})$, then $|\mathcal{F}_u| \leqslant |N[u]| < \omega \leqslant S$ is bound by the number of neighbors of u in the overlap graph, which is bound by the clique number ω, which is itself, by the CSS assumption, bound by S. The size of $|\mathcal{Q}'_u| = |\mathcal{C}_\mathcal{G}(\mathcal{F}_u)| = \frac{S!}{(S-|\mathcal{F}|)!} \leqslant S!$ is the number of proper colorings of a clique of size $|\mathcal{F}_u|$ with S colors. The overall complexity is thus bound by $\mathcal{O}(|\mathcal{U}|S!)$, which is linear in the number of utterances. Splitting at the boundaries of connected components has no effect here since it is implicitly done by the algorithm when the graph is an overlap graph.

5.4 Experiments

To validate the proposed Graph-PIT training scheme, it is applied to different datasets and compared to the conventional uPIT training scheme. Data is selected with increasing

difficulty, starting with the simulated clean anechoic meetings in Section 5.4.1 and then on simulated reverberant meetings in Section 5.4.2. At the end of this section, the execution time of the proposed assignment algorithm is profiled in Section 5.4.3 in order to quantify if it is applicable during training time.

5.4.1 Application to Anechoic Meetings

As the first case-study, the Graph-PIT and uPIT training schemes are applied to simulated clean anechoic meetings. The experiments are taken from [OC3].

5.4.1.1 Training and Evaluation Procedure

The small DPRNN model (see Section A.1.2) with $S = 2$ output streams is used. The model is trained on segments of lengths between 2 s and 64 s with the SA-tSDR loss, as introduced in Chapter 4, and a soft threshold of 25 dB. Any segments with more than 2 speakers are discarded for uPIT training since the loss cannot handle them. The amount of data discarded increases with the segment length, but stays below 80 % for 16 s segments. When training with Graph-PIT, the decomposition described in Section 5.2.4 is used and no segments have to be discarded.

For CSS-style evaluation on full meetings, the stitching scheme described in Section 2.6.2.1 is used. The future and history contexts are kept constant at $T_\mathrm{f} = T_\mathrm{h} = 1\,\mathrm{s}$ and the payload T_c is varied between 0.4 s and 14 s to match the desired segment length. The permutation is solved based on the MSE on the time-domain signals. Additionally, experiments are performed without stitching to show that Graph-PIT generalizes to long signals without segmentation or a window-wise processing approach.

Meeting data is generated with the MMS-MSG toolkit [OC2] with five to eight speakers per meeting, sampled uniformly, and a length of 120 s per meeting. The overlap is sampled uniformly between 20 % and 40 %. In total, 36 h of training data and 1 h each of development and test data were generated. The test data is based on the `train_si284` sub-set of WSJ and the test and development sets are based on `test_eval92` and `cv_dev93`, respectively. These experiments use a sample rate of 8 kHz.

5.4.1.2 Graph-PIT SA-CI-SDR as a Performance Measure

In order to compute a signal-level metric in the meeting scenario, we extend the CI-SDR metric (see Section 2.8.3) with Graph-PIT to the SA-CI-SDR, which solves the assignment problem and evaluates the signal quality of the full recording. The CI-SDR is an utterance-level metric and can only be applied to meeting-level evaluation in an utterance-wise manner according to Section 2.8.2 with the already mentioned drawbacks.

The resulting metric is called SA-CI-SDR, and it is defined as

$$\text{SA-CI-SDR} = \max_{\mathbf{C}\in\mathcal{C}} 10\log_{10} \frac{\sum_s^S \left\| \sum_{u\in\mathcal{U}} c_{u,s}\mathbf{a}_{u,s} * \breve{\mathbf{s}}_u \right\|^2}{\sum_s^S \left\| \sum_{u\in\mathcal{U}} c_{u,s}\mathbf{a}_{u,s} * \breve{\mathbf{s}}_u - \hat{\mathbf{s}}_s \right\|^2}, \tag{5.45}$$

where $\mathbf{a}_{u,s}$ is again the filter that compensates for the transfer function between the speech source and the recording device. This filter is optimized for each utterance u separately w.r.t. the corresponding estimated signal with Eq. (2.13). The decomposition for the SA-CI-SDR for use with the Graph-PIT algorithm is presented in Section A.3.2.

The SA-CI-SDR is a meeting-level metric that can be computed without the need of a full meeting transcription pipeline. It is thus a good candidate for the evaluation of speech separation systems in meeting scenarios and is used in the experiments below. But, it should be noted that a single assignment error of a single utterance can propagate through a connected component of the overlap graph to other utterances and cause penalties also for other utterances that are well separated but whose assignment does not match the graph constraints. For an example, we go back to Fig. 5.1 where the utterances $\mathbf{s}_3$, $\mathbf{s}_4$, $\mathbf{s}_5$ and $\mathbf{s}_6$ form a connected component, with $\mathbf{s}_3$ overlapping $\mathbf{s}_4$, $\mathbf{s}_4$ overlapping $\mathbf{s}_3$ and $\mathbf{s}_5$, and $\mathbf{s}_6$ overlapping $\mathbf{s}_5$. If the system output would split the second utterance $\mathbf{s}_4$ in half and would place the second half on the second output stream, it would have to put the following blue utterance ($\mathbf{s}_5$) on the first stream and the green one ($\mathbf{s}_6$) on the second stream. Even if the separation would be otherwise perfect, there is no valid solution for the Graph-PIT problem with $S = 2$ output streams that would allow both green utterances ($\mathbf{s}_3$ and $\mathbf{s}_6$) to be placed on the same stream. The Graph-PIT SA-CI-SDR would thus treat at least one of them as an error, which may cause a huge penalty, even though these utterances may be reconstructed correctly.

5.4.1.3 Impact of the Stitcher Segment Size

Fig. 5.2 Shows the impact of the stitcher's segment size on the separation performance, measured with ORC-WER. The top part of Fig. 5.2 shows the ORC-WER for different segment sizes. The bottom part shows the distribution of the number of speakers in the stitched segments. The red line indicates the number of segments used for uPIT training.

The performance of all uPIT models decreases with larger larger segment sizes, i.e., larger numbers of speakers in a segment. None of the uPIT models generalizes to the case without stitching. The average performance, however, increases with increasing training segment size T_{Tr} up to the point where too many segments are discarded. This indicates that larger contexts improve separation performance and modifying the model such that it is able to handle longer contexts is beneficial.

The Graph-PIT model behaves similarly to the uPIT model for small training segment sizes T_{Tr}. In these cases, almost no training data is discarded for uPIT which also means that the assignments determined by Graph-PIT and uPIT are identical for most training examples. With larger training segment sizes, the Graph-PIT models outperform uPIT and the larger the training segment size is, the better it generalizes to longer segments, even beyond the trained segment size. With large enough training segment sizes, the Graph-PIT models are able to generalize to the case without stitching.

This same trend, that Graph-PIT outperforms uPIT, was also observed on preliminary experiments where uPIT was trained on two-speaker data only, i.e., where no training examples had to be discarded.

5.4.1.4 Reducing the Computational Cost of Stitching

The bottom plot in Fig. 5.2 shows the amount of computations introduced by the stitching process in relation to the payload window. This overhead arises from the fact that the overlapping signal parts have to be processed twice. Since the history and future context were kept constant at $T_{\mathrm{f}} = T_{\mathrm{h}} = 1\,\mathrm{s}$, this overhead decreases with increasing segment size. For common segment sizes below 4 s, where uPIT models show their best performance, the

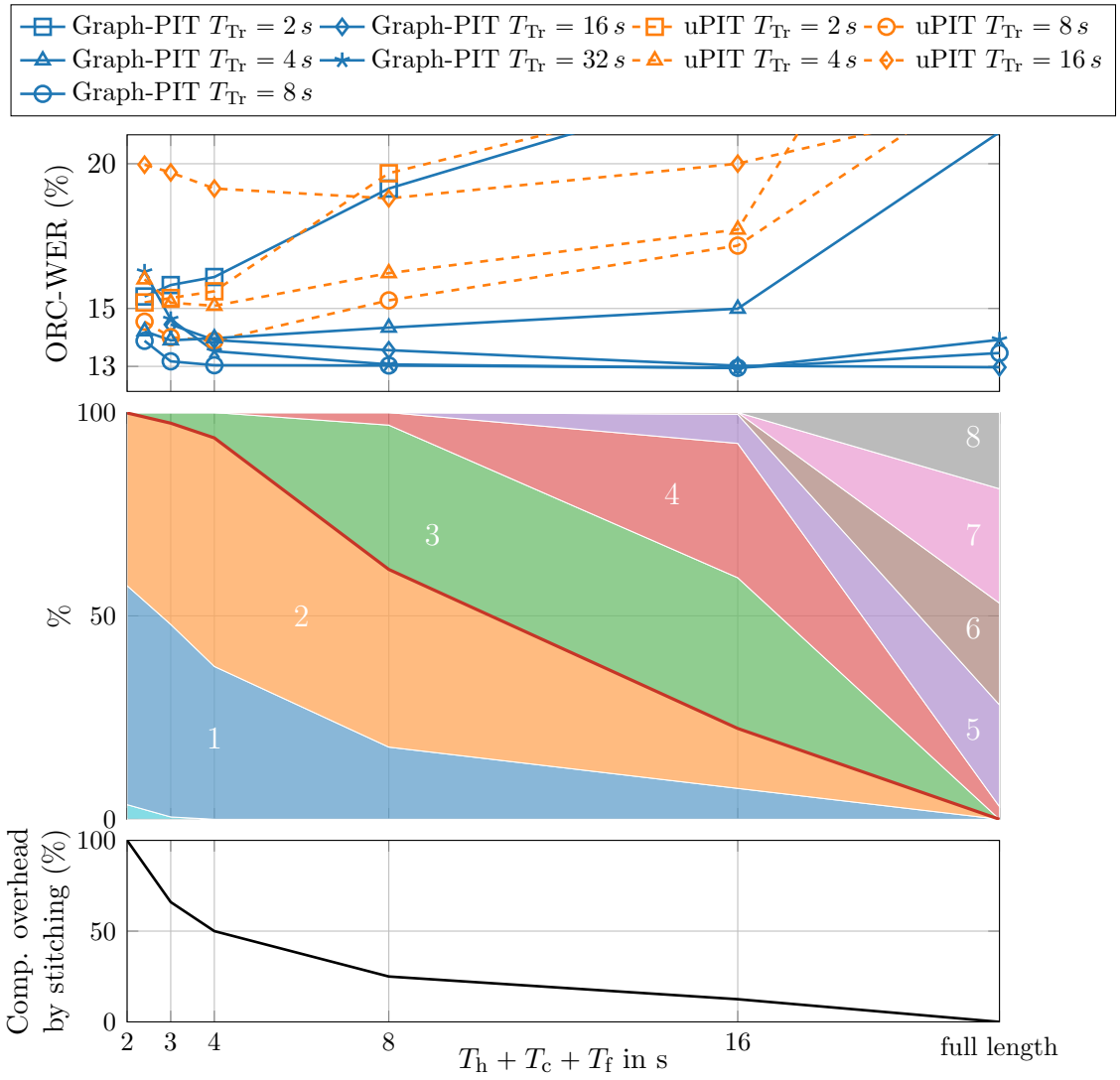


Figure 5.2: Analysis of the separation performance of Graph-PIT and uPIT models over segment size. *Top:* ORC-WER plotted over the segment size for stitching. Lower is better. *Middle:* Distribution of the number of speakers in the stitched windows. *Bottom:* Amount of computations introduced by overlapping windows in the stitching process. *(Modified from [OC3])*

overhead is larger than 50 %. Using Graph-PIT with larger segment sizes, this overhead can be reduced significantly up to the point where stitching can be completely avoided.

5.4.1.5 Detailed Comparison With Different Metrics

Table 5.1 shows a detailed comparison with additional metrics. We here specifically use the SA-CI-SDR as a signal-level metric to judge the quality of the separated signals, the VAER (defined at the end of Section 2.8.5), after a simple energy-based VAD, as an alternative to the DER when no speaker labels are estimated to judge how well the speech activity is preserved, and the utterance-wise WER and ORC-WER for judging the impact on downstream tasks. Only the best stitching configuration is shown for each training configuration.

The same general trend as in Fig. 5.2 can be observed: Increased training segment sizes

Table 5.1: Comparison of models trained with uPIT and Graph-PIT, evaluated with different metrics on anechoic WSJ-based simulated meeting data. Best numbers per column are set in bold and best numbers without stitching are underlined. For each training configuration, numbers are reported for evaluation without stitching (—) and the best stitching configuration. *(Adapted from [OC3])*

Training Scheme	T_{Tr} in s	Stitcher in s $T_{\text{h}} + T_{\text{c}} + T_{\text{f}}$	SA-CI-SDR ↑ in dB	VAER ↓ in %	utt.-wise WER ↓ in %	ORC-WER ↓ in %
no sep.	—	—	0.0	65.6	38.1	77.7
uPIT	2	—	7.9	48.6	23.3	25.0
		1+0.4+1	15.4	7.9	15.4	15.2
	4	—	8.6	48.5	29.4	32.2
		1+2+1	16.5	7.7	15.1	15.1
	8	—	8.3	27.1	22.6	23.2
		1+2+1	17.2	6.6	14.9	13.9
	16	—	8.5	22.9	22.3	22.1
		1+14+1	9.2	18.5	20.0	20.0
Graph-PIT	2	—	7.5	49.2	24.5	26.0
		1+0.4+1	15.1	12.8	16.1	15.4
	4	—	13.8	23.9	21.0	21.1
		1+6+1	17.4	7.2	14.7	14.3
	8	—	14.1	39.9	20.6	20.8
		1+6+1	18.1	19.0	14.3	13.2
	16	—	<u>18.6</u>	10.5	<u>14.0</u>	<u>13.0</u>
		1+14+1	**18.7**	12.2	14.2	13.0
	32	—	18.0	<u>8.3</u>	14.3	13.9
		1+14+1	18.5	**6.4**	**13.6**	**12.9**

generally leads to better performance for Graph-PIT and enables the use of longer stitcher segment sizes. Graph-PIT beats uPIT in all metrics. The signal quality, speech activity estimation and WER all improve with Graph-PIT and larger training segment sizes.

5.4.1.6 Pre-training With uPIT

The Graph-PIT assignment problem is more complex than the uPIT assignment problem. From this arises the hypothesis that a training curriculum would be beneficial, starting with uPIT to teach the model basic speech separation and continuing with Graph-PIT to make it generalize to larger numbers of speakers.

Table 5.2 shows the results of training with different training schemes. Only models with a fixed training segment size of $T_{\text{Tr}} = 8\,\text{s}$ are shown because the curriculum approach "uPIT → Graph-PIT" yielded the best results with this segment size. The curriculum model was pre-trained with 600 000 uPIT training steps and then continued with Graph-PIT for 600 000 steps while the uPIT and Graph-PIT models were trained for 600 000 steps each. Only the best stitcher configuration is shown for each training configuration.

The results show the performance of the curriculum "uPIT → Graph-PIT" model between

Table 5.2: Comparison of training schemes on anechoic WSJ-based simulated meetings for $T_{\mathrm{Tr}} = 8\,\mathrm{s}$. *(Taken from [OC3])*

Training Scheme	Stitcher in s $T_{\mathrm{h}} + T_{\mathrm{c}} + T_{\mathrm{f}}$	SA-CI-SDR ↑ in dB	ORC-WER ↓ in %
uPIT	—	8.3	23.2
	1+2+1	17.2	13.9
uPIT → Graph-PIT	—	11.2	20.5
	1+6+1	**17.9**	14.6
Graph-PIT	—	14.1	20.8
	1+6+1	17.8	**13.2**

the "uPIT" and "Graph-PIT" models when separation is performed without stitching. The curriculum model seems to have learned the uPIT behavior in the beginning and was unable to learn the Graph-PIT generalization well enough during fine-tuning. Training with Graph-PIT from scratch yields the best performance while being simpler and faster than the pre-training with uPIT.

5.4.2 Application to Artificially Reverberated Meetings

To evaluate whether the Graph-PIT training generalizes to reverberant data, it is evaluated on artificially reverberated simulated meetings based on utterances from the WSJ dataset. The mixtures are kept as similar as possible to the anechoic meetings used in the previous section for a fair comparison.

5.4.2.1 Data

The clean source data and speech activity are the same as in the previous experiments on clean data, but with added reverberation. Speaker and microphone positions are randomly sampled without constraints on minimum angular distances between speakers. The room impulse responses are generated in the same way as [21] with the image method [102]. The room dimensions are sampled randomly between 7.6 m by 5.6 m and 8.4 m by 6.4 m. The sound decay time (T60) is sampled between 200 ms and 500 ms.

5.4.2.2 Model and Training Procedure

For the experiments on reverberant data, a Bidirectional LSTM (BLSTM)-based source separator working in the Short-term Fourier Transformation (STFT)-domain is chosen, as described in Section A.1.3. Frequency-domain separation was shown to be more effective than time-domain approaches for reverberant data [OC10]. The general training procedure is similar to the previous section. For the ground truth target signals, the early reverberation signal $\check{\mathbf{s}}^{(\mathrm{early})}$ is used, i.e., the clean signal convolved with the first 50 ms of the room impulse response, to eliminate the reverberation tail. 90 % of the single-speaker segments were discarded to increase the amount of overlap in the training data.

Table 5.3: Separation performance on simulated WSJ-based and artificially reverberated meetings using a BLSTM-based separation network. *(Taken from [OC3])*

Training Scheme	Stitcher in s $T_\mathrm{h} + T_\mathrm{c} + T_\mathrm{f}$	SA-CI-SDR↑ in dB	ORC-WER↓ in %
no separation	—	−0.4	74.9
uPIT	—	5.3	35.6
	1+1+1	9.5	28.9
Graph-PIT	—	8.2	29.9
	1+6+1	**9.8**	**27.7**

5.4.2.3 Results

The separation performance for the models trained on reverberant data is shown in Table 5.3. As expected, the overall performance is degraded compared to the anechoic scenario. Still, similar tendencies can be observed: The Graph-PIT model outperforms the uPIT model for larger segment sizes and better generalizes to processing without stitching.

5.4.2.4 Challenges on Reverberant Data

Training with Graph-PIT on reverberant data is more challenging than training on anechoic data. The reverberation tail blurs the segment boundaries such that they are not clearly defined and thus harder to detect by the model. In training, this can lead to ambiguities where tiny variations in the utterance positions can dramatically change the overlap graph. This may confuse the model during training and lead to unstable behavior. It was found that overall more overlap is required for training compared to the anechoic case, which is the reason why 90 % of the single-speaker training segments were discarded.

5.4.3 Algorithm Execution Time

To quantify whether solving the Graph-PIT assignment problem during training time is feasible, i.e., does not increase the training time significantly, the execution time of the dynamic programming algorithm described in Section 5.3.3 is profiled and compared with alternative approaches from [OC4]. The analysis is taken from [OC4] and the results are shown in Fig. 5.3. For this analysis, a toy dataset is generated with a three-output separator ($S = 3$) and a variable number of utterances $|\mathcal{U}|$. All simulated utterances have a length of 2 s and an overlap of 0.5 s with the next utterance. Silence is intentionally excluded in the dataset because it can trivially be removed by cutting in silence and solving the problem individually for each connected component of the overlap graph. Note that this would improve the runtime of all algorithms except the dynamic programming algorithm, which already performs this optimization implicitly. Only the number of utterances in one connected component of the overlap graph is of interest.

The benchmark is performed on a single CPU core. All Graph-PIT algorithms are implemented in Python and use the `numpy` library for matrix operations. The numbers

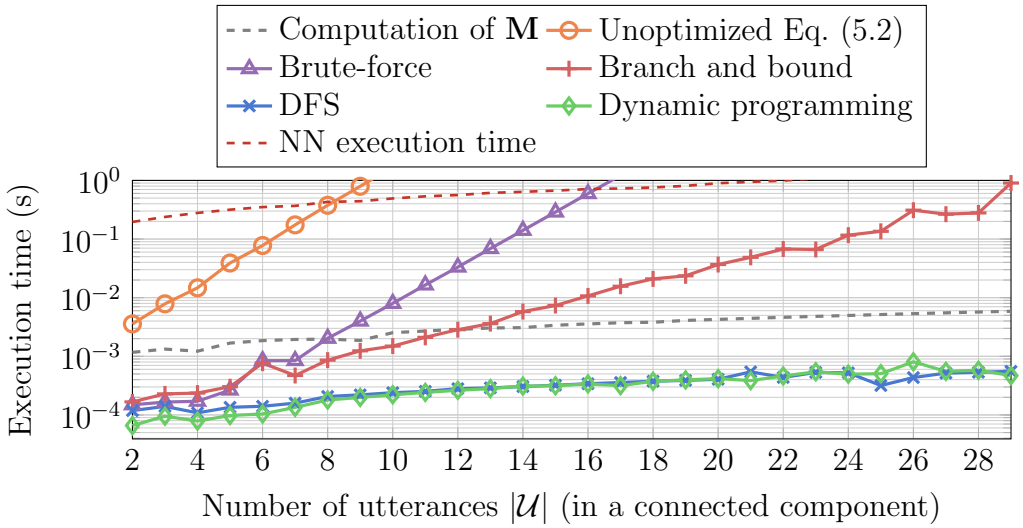


Figure 5.3: Comparison of different assignment algorithms for different sizes of connected components on toy data. All algorithms use the pre-computed score matrix $\mathbf{M}^{(\text{Graph-PIT})}$, whose computation time is shown as a gray dashed line. The red dashed line indicates the execution time of the forward and backward steps of a typical separation network architecture. *(Taken from [OC4])*

displayed in Fig. 5.3 are the average execution time of 500 runs for each algorithm and each number of utterances.

5.4.3.1 Reference Algorithms

Other algorithms are taken from [OC4] as references and shortly described in the following.

Brute-force The brute-force algorithm computes Eq. (5.2) by enumerating all proper colorings, but uses a pre-computed score matrix $\mathbf{M}^{(\text{Graph-PIT})}$ and the decomposition from Eq. (5.9). Its runtime is in general exponential in the number of utterances, which can also be observed in Fig. 5.3 where its runtime appears linear in the logarithmic scale.

Greedy Depth-First-Search (DFS) A greedy and not necessarily optimal solution can be found by leveraging an (incomplete) Depth-First Search (DFS) algorithm in the space of partial solutions. Utterances are assigned sequentially so that each newly selected color adds the smallest possible cost while respecting the constraints imposed by the overlap graph. If the next utterance cannot be assigned given the overlap graph, the algorithm backtracks and tries to assign the next utterance to a different stream. This algorithm has a best-case complexity of $\mathcal{O}(|\mathcal{U}|S)$ and an exponential worst-case complexity.

Branch-and-bound search The optimal solution can be found by continuing the DFS search until the best solution is found. During the search partial solutions with costs higher than the current best are discarded as they can never be extended to a lower overall cost. The complexity lies between the greedy DFS and the brute-force algorithm.

5.4.3.2 Results

The execution time of the dynamic programming algorithm is sub-exponential (sub-linear in the logarithmic plot) w.r.t. the number of utterances, which corresponds to the theoretical analysis in Section 5.3.3. Its runtime is on-par with a greedy DFS algorithm, which has a linear best-case complexity. Both algorithms have a negligible runtime compared to the time required to execute a typical separation network (dashed red line) or to compute the score matrix $\mathbf{M}^{(\text{Graph-PIT})}$ (gray dashed line).

All other reference algorithms show an exponential runtime (linear in the logarithmic plot) and are thus significantly slower than the dynamic programming algorithm.

It can be concluded that optimally solving the Graph-PIT assignment problem with the dynamic programming algorithm does not significantly contribute to the training time.

5.5 Discussion

Graph-PIT is an elegant solution for the assignment problem in the training phase of a speech separator. It is, however, not the only approach and using it is not always straightforward, as outlined below.

5.5.1 Related Work

A similar approach to Graph-PIT named Group-PIT was developed simultaneously in [41]. Instead of solving the permutation problem for arbitrarily complex speaking patterns (like Graph-PIT), they limited the scenario to two output streams and chose to generate training data such that the overlap graph comprises only a single connected component with exactly $S!$ many proper colorings. By this, they reduce the problem again to a permutation problem across streams, where every stream can now contain multiple speakers.

This approach has the same advantages as uPIT: It limits the number of possible assignments to $S!$, which makes the training in general more stable compared to Graph-PIT. On the other hand, it requires training data that is specifically generated for this purpose while Graph-PIT can be applied to any data with arbitrary overlap patterns. One example is training data with silence, which is not considered in Group-PIT (but can fundamentally be integrated) and is naturally supported by Graph-PIT.

5.5.2 Practical Considerations

While Graph-PIT is an elegant extension of uPIT to the meeting scenario, it comes with some practical challenges. Most importantly, training is problematic if overlaps are not obvious or if a single violation of the overlap graph leads to a large penalty. In the following, some practical considerations are discussed when training models with Graph-PIT.

Length of training data To see a benefit of Graph-PIT over uPIT, the training data must be long enough. Short training examples lead to the same assignment as uPIT. The training segments should cover cases with more than S speakers and graphs with more than $S!$ proper colorings. The experiments show that longer training segments lead to better performance for Graph-PIT.

Receptive field The model architecture should provide a receptive field that is large enough to cover one connected component of the overlap graph. Otherwise, it cannot solve the assignment problem consistently because information about (the chain of) overlapping utterances is lost while processing a chain of overlapping utterances that exceeds the receptive field. This is the reason for choosing Recurrent Neural Network (RNN)-based architectures in this thesis over Convolutional Neural Networks (CNNs).

Difficulty of training data Realistic meetings often contain single-speaker regions where different speakers take turns without overlaps. These cases are particularly easy for Graph-PIT, which can simply pass through the input to one output stream and silence the other streams. It is important that the training data is balanced such that it does not contain too many of these too easy cases, so that the model is forced to use all its outputs. Training data can be filtered (as done in the reverberant experiments in Section 5.4.2) to remove single-speaker segments.

Ambiguities in the training data Ambiguous situations in the training data, e.g., when utterances touch but do not overlap, can lead to a degraded performance. In these cases, it is unclear from the signals alone whether the utterances overlap, e.g., the uttered speech may not overlap, but the reverberation tails overlap. It is then ambiguous whether an edge should exist between the corresponding vertices in the overlap graph. The model in general prefers assigning utterances to the same stream, likely because an identity operation is easier to realize than assigning parts of the mixture signal to different streams. In ambiguous cases, this behavior may leadto a good performance for some examples while for others the overlap graph is violated, resulting in a (large) penalty. This creates conflicting gradients and should be avoided during training.

5.6 Summary

In this chapter, the Graph-PIT training scheme was introduced. It was shown that the assignment problem in Graph-PIT is equal to a weighted graph coloring problem and an efficient algorithm was introduced to solve it. Experiments were conducted on simulated anechoic and reverberated meetings. It was here shown that Graph-PIT works well on clean data, but training it on reverberated data is more challenging. Still, Graph-PIT was shown to outperform uPIT for larger segment sizes across experiments and it was shown that the model can generalize to longer segments than it was trained on.

6 Separation-first Diarization-last Pipeline for Meeting Transcription

After having described the evaluation metrics for meeting transcription and having designed a sub-system for speech separation, this chapter describes a full separation-first diarization-last meeting transcription pipeline built on top of the CSS-style speech separator described in previous chapters. This chapter is largely based on [OC1].

Related own publications

This chapter is based on [OC1] and extends it with additional experiments and a larger literature comparison in Section 6.4.4.

Most meeting transcription pipelines outlined in Section 2.3.2 perform speech recognition as the last step, after separation and diarization, so that the ASR depends on all previous steps and disallows the use of linguistic information in the earlier stages. We here look at the diarization-last pipeline, where diarization is performed after speech recognition, such that information obtained from the ASR's transcript can improve the diarization.

An overview of the pipeline is shown in Fig. 6.1. The audio is first processed with the stitching scheme described in Section 2.6.2.1, where the mixture is segmented into overlapping segments of equal size, each segment is processed with the separator, and the segments are stitched together to form S continuous signals. Each separated stream is then segmented using a VAD, described in detail in Section 6.1, to obtain initial candidates for speaker turns. Each VAD segment is processed by the ASR system, which outputs a transcript and the start and end times of each word. This transcript information is used to further refine the segmentation, which is described in detail in Section 6.2. Then, one speaker embedding vector is extracted for each refined segment and a simple k-means clustering is performed to group the segments into one cluster per speaker. We assume that the number of speakers is known in advance, which is a common assumption in clustering-based diarization systems.

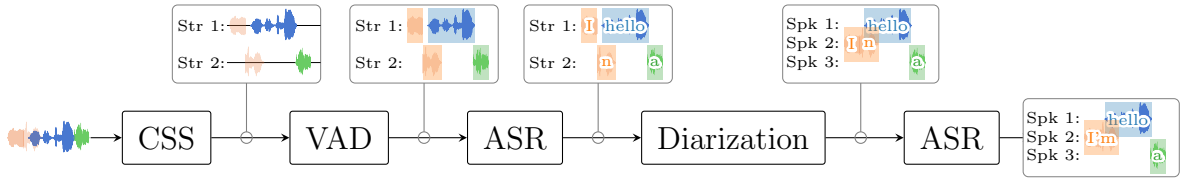


Figure 6.1: Overview of the transcription-supported diarization pipeline. The diarization uses word-level timing information from the ASR output. It can fix segmentation errors in the CSS output ("I" and "n" → "In"). The second ASR pass can fix further recognition errors ("In" → "I'm"). *(Modified from [OC11])*

6.1 Cross-stream Energy-based Voice Activity Detection (VAD)

The first step after separation in the diarization pipeline is to find segments that likely contain continuous speech and exclude non-speech. This is here done with an energy-based VAD that uses the signals from both streams to detect speech activity. Compared to conventional energy-based VADs computed on only a single stream, we can here improve the detection of false alarms by using information from all output streams.

First, both streams are transformed into the STFT domain and the energy is computed for each frame, excluding frequencies unimportant for speech. The obtained energy estimates are smoothed using dilation, erosion, dilation and again erosion. The kernel sizes for the dilation and erosion operations are chosen such that small gaps are closed, short energy spikes are removed, and the energy is smoothed such that energy regions appear wider after the smoothing.

Based on these smoothed energy estimates, the ratio of the frame-wise energies between each stream and the sum of all other streams is computed. This ratio indicates whether a significant amount of speech is present on one stream compared to the other streams, which helps in compensating leakage errors.

A frame is classified as speech if three conditions are met: First, the combined energy across all streams must be above a first threshold $\tau^{(\text{total})}$, relative to the total energy observed across all streams. Second, the energy in the frame must exceed a second threshold, $\tau^{(\text{stream})}$, relative to the maximum energy observed across all frames. Third, the ratio, or difference in the logarithmic domain, of this frame's energy to the sum of all other streams must be above a third threshold $\tau^{(\text{ratio})}$.

The first condition ensures that the frame overall contains a significant amount of energy in relation to the total energy of the mixture. While on first sight, this first condition seems to be redundant to the second condition, it catches cases where one stream contains significantly more energy than the other streams, e.g., when one estimated stream contains no speech at all. The second condition ensures that the frame contains a significant amount of energy in relation to the other frames on the same stream, which is a condition often used for energy-based VAD on a single stream. The third condition makes it less likely that a frame is classified as speech when there is speech on the other streams, which helps to reduce leakage errors.

After individual frames are classified as speech or non-speech, adjacent frames are combined into segments and corresponding start and end times b and e are computed for each segment.

6.2 Transcription-supported Sub-segmentation

The segments obtained by the VAD are not guaranteed to be speaker-homogeneous. Especially in discussions where speakers only overlap slightly and pauses are small, multiple speaker turns may be classified as a single speech segment by the VAD. For clustering-based diarization, however, speaker-homogeneous segments are required.

Conventionally, speaker turns are either detected by VAD alone or by clustering of short equally sized segments. With a VAD, speaker turns may be missed when pauses between speaker turns are short. If using equally sized segments, it is not guaranteed to hit the

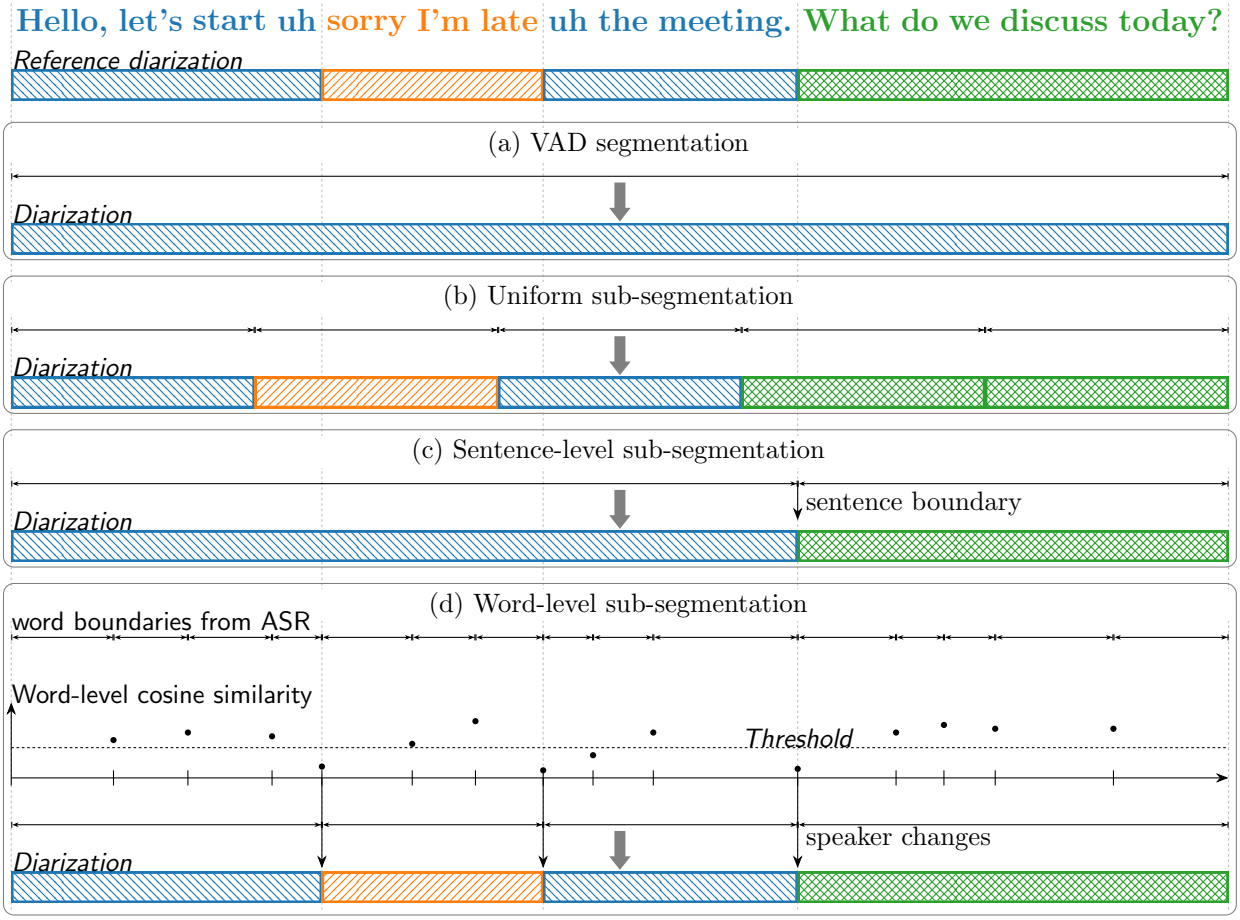


Figure 6.2: Visualization of the different sub-segmentation schemes. On top is the transcript of an overlap-free continuous speech segment that contains speech of three speakers. Below are the different sub-segmentation schemes. Different colors represent different speakers.
(Taken from [OC1])

exact speaker turn, especially when the speakers take turns rapidly. This has an impact on both the timestamp accuracy and on the embedding vector quality for clustering, which decreases when multiple speakers are active. For the segment size, a trade-off must be taken between embedding vector quality, which generally increases with segment length, and timing accuracy, which decreases with segment length.

Instead, we can use the ASR system to obtain better candidates for the speaker turns. An example is visualized in Fig. 6.2, where three speakers are active but non-overlapping. The VAD segmentation in Fig. 6.2a misses the speaker changes. The uniform segmentation in Fig. 6.2b detects speaker changes (here assuming that the clustering makes no errors), but the detected changes are not aligned with the true speaker changes.

6.2.1 Sentence-level Segmentation

Assuming that speaker turns appear likely at sentence boundaries, information from the ASR can be used to sub-segment the audio at positions where the ASR predicts a sentence

boundary. We use the Whisper [79] ASR system for this purpose, which predicts punctuation marks and generated timestamps for word boundaries. The segment size is generally longer than for the uniform sub-segmentation, which improves the quality of embedding vectors as the segments are longer and are more likely to contain only a single speaker. As visualized in Fig. 6.2c, this segmentation detects speaker changes at sentence boundaries, but it may miss speaker turns when no sentence boundary is predicted.

6.2.2 Word-level Embeddings for Segmentation

Although the speaker changes are likely to occur at sentence boundaries, this is not guaranteed. Speaker changes may happen in the middle of a sentence, especially in heated discussions, or the ASR system may fail to predict the end-of-sentence tag. To improve segmentation in these cases, we can use word-level d-vectors to detect speaker turns. This procedure is depicted in Fig. 6.2d.

First, the word boundaries, i.e., the start and end times, are obtained from the ASR system. Then, one d-vector is extracted for each word. These d-vectors have a low quality since they are extracted from short segments with an overall low amount of speech content, but their quality is generally enough to detect speaker changes compared to d-vectors from neighboring words with an equally low quality. This is because neighboring words share a similar acoustic environment and are likely to be similar if the speaker is similar even if the temporal length is not enough to extract a globally robust speaker representation.

To further improve the detection of speaker turns, the word-level embeddings are not compared to each other directly, but the d-vectors are averaged over a short time window. Each center point between two words is treated as a candidate for a potential speaker change point. For each candidate, a score is computed as the similarity between the average d-vectors of a left and right context around the candidate position. The lower the score, the more the d-vectors differ, and the more likely a speaker change is. A candidate is selected as a speaker change point if the score is lowest among all candidates within a context, whose size is determined on the development dataset, before and after the candidate and the score is below a threshold.

6.3 Speaker Embedding Clustering

Single-speaker segments obtained with one of the sub-segmentation schemes described above are clustered similarly to a conventional clustering-based diarization system. For each segment, a speaker embedding vector is extracted and the embedding vectors are clustered using the k-means clustering algorithm. The number of speakers is assumed to be known for the clustering.

6.3.1 Embedding Vector Extraction

As an embedding extractor, a pre-trained d-vector extractor is used[1]. It is based on the ResNet34 architecture and trained on VoxCeleb [104] with MUSAN [105] augmentation and reverberated with simulated RIRs. On VoxCeleb1-O, it achieves an Equal Error Rate (EER) of 1.06 % and on VoxCeleb1-H, it achieves an EER of 2.12 % [103].

[1] https://huggingface.co/boeddeker/d_vector_yellow_inherent_elephant. Also used in [OC1, 103]

Table 6.1: Speaker-agnostic performance of the separation and ASR sub-systems on LibriCSS-raw without diarization. VAD is applied to all systems prior to ASR. WERs are reported in %. *: asclite WER. *(Adapted from [OC1])*

Model	ASR	ORC-WER↓	DI-cpWER↓
No separation	Whisper	26.6	94.8
TF-GridNet-Large [OC1]	Whisper	6.8	13.9
TF-GridNet-Large-v2	Whisper	4.8	14.4
Clean signals	Whisper	3.5	3.5
MTRNNT [38]	Integrated	23.6	—
t-SOT TT [37]	Integrated	7.6*	—
TS-SEP [36]	ESPnet	—	5.8

6.4 Application to the LibriCSS Dataset

The pipeline is evaluated on the LibriCSS-raw dataset [14]. These experiments are taken from [OC1] and extended.

6.4.1 Model Architecture and Training

The experiments presented in Sections 6.4.2 and 6.4.3 use the TF-GridNet-Large model as a source separator, described in Section A.1.4. These experiments are taken from [OC1] and extended with evaluations using additional metrics, i.e., the ORC-WER and the time-constrained DI-cpWER (DI-tcpWER). In Section 6.4.4, numbers are reported with the TF-GridNet-Large-v2 model that uses the modified architecture from the small model from the earlier chapters and is trained on additional data, generated from the DiPCo and CHiME-6 datasets. See Section A.1.4 for details. All speech separation models are trained with uPIT on fully overlapped two-speaker mixtures and single-speaker mixtures of 4 s length generated with the MMS-MSG toolkit [OC2]. Dynamic mixing is used to increase the diversity of the training data. Training examples for the small model were augmented with noise from the MUSAN dataset. The model with the best loss on a validation set, which was generated similarly to the training data, was selected for evaluation. The batch-size was set to 2, limited by the available GPU memory.

The VAD parameters are optimized on the development set to minimize the ORC-WER. The thresholds are set to $\tau^{(\text{total})} = \tau^{(\text{stream})} = -35\,\text{dB}$ and $\tau^{(\text{ratio})} = 10$. For the d-vector-based segmentation, the context size for the left and right context is set to six words and the similarity threshold for detecting a speaker change is set to 0.2.

6.4.2 Speaker-agnostic Evaluation Without Diarization

To first assess the separation performance alone, without the impact of the diarization pipeline, the output of the ASR system is scored with the ORC-WER and DI-cpWER in Table 6.1. Here, no time-constrained error rates are used because the referenced prior work did not use them.

Table 6.2: Comparison of different sub-segmentation schemes on LibriCSS-raw. The pipeline uses TF-GridNet-Large for separation, an energy-based VAD, Whisper and k-means clustering with a known number of speakers. WERs are reported in %. *(cpWER taken from [OC1])*

Sub-segmentation	cpWER↓	tcpWER↓	DI-tcpWER↓	tcORC-WER↓	DER↓
—	14.8	15.4	14.8	6.9	13.2
uniform 4s	9.8	10.0	7.7	8.2	10.7
uniform 2s	12.4	13.0	6.9	9.2	12.4
sentence	7.9	7.9	7.5	6.9	9.7
word	7.6	7.7	7.2	7.2	9.5
sentence + word	7.2	7.3	6.7	6.9	9.4

As a (pessimistic) baseline, the mixture is processed with the Whisper ASR system without separation. Only VAD is applied to divide the mixture into manageable segments. This yields an ORC-WER of 26.6 % and a DI-cpWER of 94.8 %. The high DI-cpWER can be explained by the fact that many utterance-merge errors are present due to the missing separation. The top-line is Whisper applied to the clean signals which yields an ORC-WER and a DI-cpWER of 3.5 %. The clean signals were directly taken from the LibriCSS dataset and weighted according to the number of times they appear in the LibriCSS dataset. ORC-WER and DI-cpWER are here equal because no segmentation errors are present and the differences caused by possible matches across utterances were small enough.

Using the separated signals, the ORC-WER drops to 6.8 % and the DI-cpWER to 13.9 %. The ORC-WER is best among prior works that reported ORC-WER and beats the numbers reported with sclite in [37]. The asclite WER reported for t-SOT TT is not directly comparable to the ORC-WER, but it in general tends to be over-optimistic because it does not have the utterance-consistency constraint from the ORC-WER (see Section 3.2.5). It is thus safe to assume that the TF-GridNet + Whisper experiment also beats [37]. The DI-cpWER, however, is worse than the one reported by TS-SEP [36]. This is mainly because TS-SEP has a speaker-attributed output so utterance-merge errors are less likely than in a CSS-style separation output. How these utterance-merge errors can be corrected in the follow-up diarization is discussed in the next subsection.

6.4.3 Speaker-attributed Evaluation With Diarization

Table 6.2 shows the speaker-attributed results evaluated with cpWER and tcpWER on LibriCSS comparing different sub-segmentation approaches. All systems here use the same embedding extractor and k-means clustering approach. Time-constrained metrics were chosen because they prefer more plausible matchings. The cpWER is reported for comparison with other systems from the literature that do not use the tcpWER, especially with [OC1].

Clustering performance The DI-tcpWER is reported in combination with the tcpWER to determine whether the performance can be improved by improving the segmentation or the clustering. When the DI-tcpWER is close to the tcpWER, performance cannot be improved by clustering alone, but the segmentation has to be improved. This is the case for the first row without sub-segmentation, where the VAD frequently combines multiple speaker turns into

one segment so that they cannot be separated by clustering. With uniform sub-segmentation, the DI-tcpWER improves due to the larger number of segments and smaller segment sizes[2], but the cpWER decreases when the segment size is chosen too small. The greatly reduced DI-tcpWER indicates that fewer segments contain multiple speakers, but the increase in tcpWER and the larger difference between tcpWER and DI-tcpWER indicates that the clustering performance is degraded with smaller segment sizes. This is expected since it is known that the quality of the d-vectors decreases with small segment sizes. For the proposed sentence- and word-level sub-segmentations, the difference between tcpWER and DI-tcpWER is small. This indicates a good clustering performance, which can be explained by having longer segments than the uniform segmentations but fewer segments with multiple speakers than the plain VAD segmentation. The sentence- and word-level sub-segmentation schemes thus successfully improve the speaker purity of the segments without reducing the segment length too much.

The combined segmentation (sentence + word) shows a better performance than the isolated ones because the word-level sub-segmentation is sometimes inaccurate at sentence boundaries where the embedding vectors may be corrupted by artifacts from the separation stage. Here, the sentence-level sub-segmentation hits the speaker changes more reliably while the word-level segmentation detects additional speaker changes within sentences. These insights were obtained using the alignment visualization approach described in Section 3.4.

Analysis of utterance-merge and utterance-split errors As discussed in Section 3.2.4, the ratio of utterance-merge and utterance-split errors are roughly reflected in the difference of DI-(t)cpWER and (tc)ORC-WER, where utterance-merge errors increase the DI-(t)cpWER and utterance-split errors increase the (tc)ORC-WER. The VAD segmentation shows a low tcORC-WER but a significantly higher DI-tcpWER, which indicates that utterance-merge errors are more prominent than utterance-split errors. With uniform sub-segmentation the DI-tcpWER improves with smaller segment sizes while the tcORC-WER increases. This indicates that the uniform sub-segmentation removes utterance-merge errors, but (together with clustering errors) adds utterance-split errors. The sentence-level sub-segmentation yields the same tcORC-WER as no sub-segmentation, but improves the DI-tcpWER. This indicates that the sentence-level sub-segmentation removes utterance-merge errors without adding (many) utterance-split errors. Using just the word-level sub-segmentation increases the tcORC-WER slightly while using both sentence and word-level sub-segmentation improves it again. Here, the clustering performance likely improves, which reduces the number of utterance-spilt errors.

Diarization The DER in Table 6.2 is computed with a collar of 0 to avoid the issues for overlapping speech discussed in Section 2.8.5. The DER shows the same tendencies as the cpWER and tcpWER. While the uniform sub-segmentation cannot significantly improve the DER, the sentence- and word-level sub-segmentations lead to a significant reduction in DER. Note that the system was optimized for speech recognition, not for diarization. Especially the smoothing operations were chosen such that they tend to over-estimate speech

[2] It should be noted that the DI-cpWER can always be improved by using smaller segments up to one segment per word. But, the performance is then not accurately reflected in the metric, especially w.r.t. utterance consistency.

Table 6.3: Single-channel LibriCSS(-raw) literature comparison WERs are reported in %. Systems may not be fairly comparable because of different test data choices[3]. *(Extended from [OC1])*

System	DER↓	cpWER↓
SC [106]	11.3	–
RPN [107]	9.5	–
SC + TS-VAD + ESPnet [36]	5.7	9.3
SC + TS-SEP + ESPnet [36]	15.7	7.8
Transcribe-to-Diarize [19]	7.9	11.6
DCF-DS + E2E ASR [108]	–	6.3
DCF-DS + E2E-SSL ASR [108]	–	4.4
CSS-AD [OC1]	9.4	7.2
CSS-AD + ESPnet [OC1]	9.4	6.2
CSS-AD (TF-GridNet-Large-v2)	9.0	5.5

activity for better ASR results [36]. Different hyperparameter choices could improve the DER performance.

6.4.4 Literature Comparison

Table 6.3 shows a comparison of the proposed CSS-AD pipeline with other systems from the literature. Numbers using the TF-GridNet-Large (if not specified) and TF-GridNet-Large-v2 are reported for the CSS-AD pipeline. The meeting transcription systems that the CSS-AD pipeline is compared against are briefly introduced below.

The first two systems from the literature are pure diarization systems that do not perform separation or speech recognition. They are given as references for the diarization performance. No cpWER was reported in the original publications. The spectral clustering (SC) system [106] uses a sliding window of 1.5 s with 50 % overlap, extracts an x-vector [109] for every segment and applies an overlap-aware spectral clustering algorithm to obtain speaker labels. The Region Proposal Network (RPN) [107] uses a neural network to predict speaker turns and a k-means clustering to group segments from the same speaker. TS-VAD [35] is another target-speaker diarization system that first extracts embedding vectors for each speaker by clustering short-term embeddings from the mixture with spectral clustering and then uses a neural network to predict the speech activity for each speaker. The reported cpWER uses no separation. Speaker segments are simply cut from the mixture based on the diarization information and fed into ASR.

The following systems perform either separation or speaker-attributed ASR to obtain per-speaker transcripts. Not all of these systems provide diarization information in their output, so that the DER is not available for all systems. TS-SEP [36] follows the same approach as TS-VAD, but instead of only predicting the speech activity, it also predicts

[3] As discussed in Section 2.5.1.4, some publications report on the full LibriCSS dataset while other choose only a sub-set (sessions 1-8) for testing and use session 0 for development. It is often not indicated which subset the numbers are reported on. In our experiments, results on session 0 tend to be slightly better than the average across the other sessions. Our numbers are reported excluding session 0.

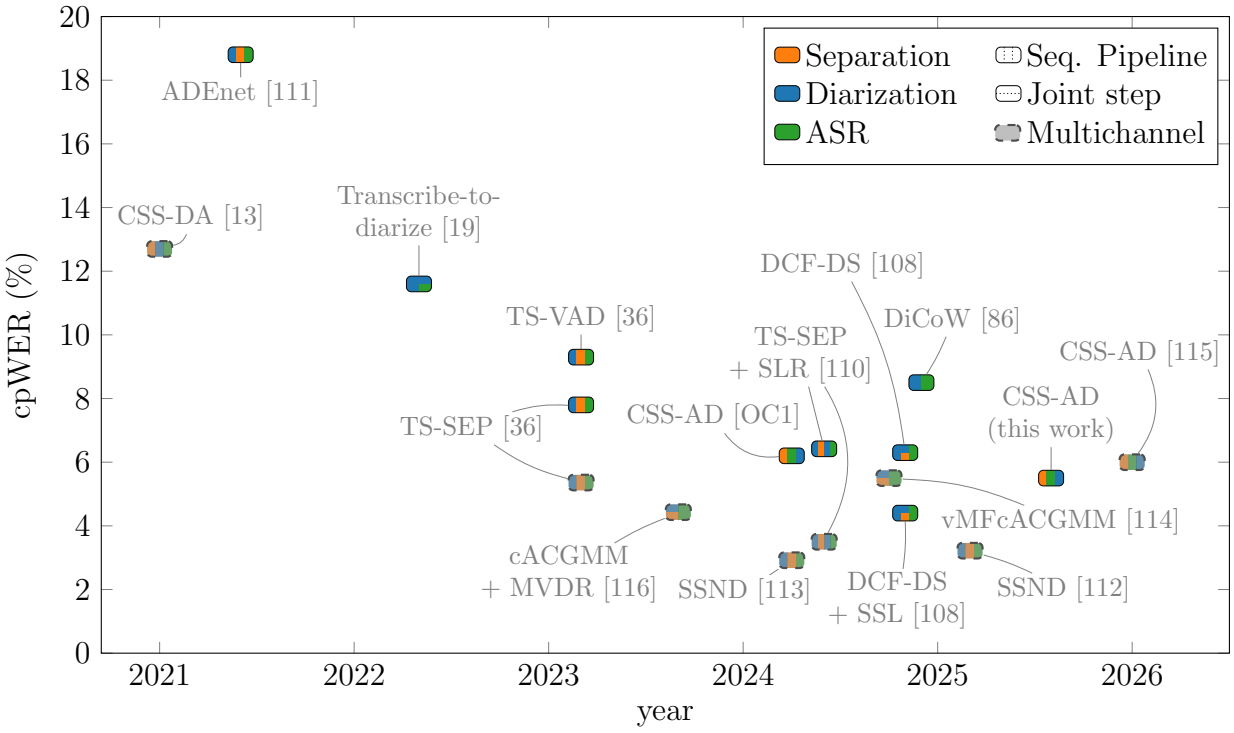


Figure 6.3: Literature comparison of single- and multi-channel systems for which the cpWER was reported on LibriCSS(-raw). The pipeline architecture is indicated by the marker coloring. Only the lowest cpWER from each publication is reported. Systems using multiple microphone channels are drawn in lighter colors and with a dashed border. Uses the month and year of publication as indicated by the publisher.

the separated speech signals. Transcribe-to-Diarize [19] follows a diarization-last approach. Similarly to TS-VAD, embedding vectors extracted from the mixture are used to apply a target-speaker ASR system which directly predicts the transcription for a speaker. Time annotations obtained from the ASR system are used to determine speech regions of each speaker. DCF-DS is another pipeline that refines an initial diarization estimate with a combined system for diarization and separation. It additionally applies dedicated speech enhancement between the separation and the speech recognition.

The presented DER and cpWER numbers show that the CSS-AD pipeline is competitive for both diarization and speech recognition. A second pass of the ASR, here with a speech recognizer taken from the ESPnet toolkit[10][4], significantly improves the final performance. This is both because the second pass has a better segmentation at the input than the first pass and because the ESPnet model was explicitly trained on LibriSpeech data. Note that the ESPnet model does not provide word-level timestamps and can thus not be used for the first ASR pass. In terms of cpWER, only the recently published DCF-DS beats it, and only in its most complex configuration. The DCF-DS model is significantly more complex, has a larger number of parameters and is trained on a larger training dataset. Also for DER, the numbers are competitive, though not the lowest. This is perhaps because the hyper-parameters of the CSS-AD pipeline are optimized for ASR performance (low cpWER) and not for diarization,

[4] The pre-trained model from https://huggingface.co/espnet/simpleoier_librispeech_asr_train_asr_conformer7_wavlm_large_raw_en_bpe5000_sp is used.

as it was also done for TS-SEP.

A wider range of systems is displayed in Fig. 6.3, where the performance in terms of cpWER is plotted over the publication date. Only publications are included that report cpWER and only the best number from each publication is plotted. The acronyms are not described in detail here, rather details can be taken from the referenced sources. Different pipeline structures are indicated with the coloring of the markers, where different colors indicate the sub-problems. A vertically divided marker means sequential processing and a horizontally divided block means joint processing of the sub-problems. Multi-channel systems are also included to increase the number of systems displayed; they are drawn in lighter colors and with a dashed border.

Fig. 6.3 nicely highlights the point from Section 2.3.2 that no pipeline is clearly superior to another pipeline. The plot shows a collection of vastly different architectures, but the only visible trend is that newer systems perform better. While most pipelines apply diarization in their first processing stage, these systems do not generally outperform pipelines that start with separation. Multi-channel pipelines, by design, can exploit more information and thus generally perform better than single-channel pipelines, but the trend is the same.

No pipeline that starts with speech recognition as its first processing step. There have been works that perform ASR directly on the streams, namely the SOT systems [18, 37] and the Multi-turn RNN-T (MT-RNN-T) [38]. But, they have not been evaluated with diarization and thus no cpWERs are available. An ORC-WER of 23.6 % was reported for MT-RNN-T, which would result in cpWER $\geqslant$ 23.6 %, which is not competitive with any other pipeline, perhaps because the training of a large joint system is more challenging and requires more data than training smaller subsystems individually.

The proposed CSS-AD pipeline achieved state-of-the-art results among single-channel systems when it was published in 2024, but was since beaten by DCF-DS [108]. As discussed above, DCF-DS is a complex cascaded system that includes an additional speech enhancement stage.

6.5 Summary

In this chapter, a complete meeting transcription pipeline was presented that builds on the CSS-style speech separation system developed earlier as the first processing step. It uses a separation-first diarization-last approach, where the diarization uses information from the speech recognition to improve the segmentation. The pipeline was evaluated on the LibriCSS dataset, where it achieved state-of-the-art results in terms of ORC-WER and cpWER.

7 Conclusions

As a conclusion to this work, the four main contributions are summarized, namely the extension of word error rates to the meeting scenario, the robust SA-SDR loss function and Graph-PIT for training a speech separator on meeting data, and the transcription-supported diarization pipeline for meeting transcription. Implementations of the core contributions are available in open-source packages, linked below.

7.1 Main Contributions

Word error rates The first main contribution of this work is the introduction and discussion of several WER-based metrics for the evaluation of meeting transcription systems. Based on the conventional WER, we have introduced a time constraint that significantly improves the temporal alignment. We have analyzed its behavior and concluded that it improves the interpretation of the WER for long word sequences in meeting transcription problems. We have devised ways for solving the assignment problem when computing a WER, namely the ORC-WER, MIMO-WER and DI-cpWER, for scenarios where either no speaker attribution is present or where speaker attribution errors should not be considered in the evaluation. We have analyzed how the proposed variants react to different transcription errors. For the speaker-agnostic WERs, we have designed efficient algorithms for computing a WER in cases where it was previously infeasible. We finally introduced a visualization scheme that helps developers analyze their meeting transcription systems to find out what specific errors the systems made. All WERs and the visualization tool are implemented in the open-source MeetEval toolkit[1].

SA-SDR The second contribution is the SA-SDR, an elegant way to robustify SDR-based loss functions for speech separation. Especially in meeting scenarios, silent targets (no speech is active) and a highly imbalanced difficulty across examples and target signals is common. The SA-SDR was introduced to mitigate the instabilities caused by these target signal properties. It was analyzed and shown theoretically that the produced gradients are better balanced across streams. Experiments showed that this also leads to an improvement in actual model training. An implementation of the SA-SDR loss is available in the padertorch toolkit[2].

Graph-PIT As the third contribution, the conventional uPIT training scheme for speech separation was extended to the meeting scenario. By allowing multiple speakers to appear on the same system output, as long as no two speakers overlap on any system output, a training scheme could be achieved that allows a model to be trained to directly process

[1] https://github.com/fgnt/meeteval
[2] https://github.com/fgnt/padertorch/blob/master/padertorch/ops/losses/regression.py

meeting data, without sub-segmentation. Compared to the previous uPIT and stitching approach, the context that the model sees can be increased significantly and the separation performance can be improved while at the same time reducing the computational complexity that was previously required for stitching. The Graph-PIT loss implementation for PyTorch is published as an open-source package[3].

Transcription-supported diarization The final contribution is a diarization-last pipeline for meeting transcription that makes use of information obtained from an ASR system to improve the quality of the diarization step. This pipeline was set into perspective comparing to other pipelines, and it was finally concluded that it is not yet decided in which order diarization and ASR, and speech separation should be solved. An extensive comparison of systems across publications showed that many different pipeline structures were used to advance the state of the art.

7.2 Future Work

7.2.1 Evaluation Metrics

For system analysis, the developer experience during system development could be improved further by better guiding the system designer towards problematic regions. This requires heuristics for estimating error types that were disregarded as out-of-topic in this thesis, where we focused only on measures that can be detected automatically in a deterministic, fully specified and transparent way derived from the simple definition of the WER. Heuristics that have already been proposed for more classical speech recognition scenarios based on error sequences or linguistic content could be generalized to the meeting scenario to provide such a guide.

What is also left open is a reliable signal-level metric for meeting-level speech separation. While the SA-CI-SDR with the assignment solved using Graph-PIT provides some insights, the discussed drawbacks make it problematic for system ranking and blind application across different systems. Further development in this area is left for future work.

7.2.2 Meeting Transcription

While the methods presented in this thesis show promising results, they have only been evaluated on completely or partially simulated data. A task that remains is testing and robustifying the systems further w.r.t. distortions seen in real recordings, such as in the AMI [47], CHiME-6, DiPCo or NOTSOFAR datasets.

Related to this, as a more general problem for speech separation, especially for the meeting scenario, is the lack of realistic paired training data. Most NN-based speech separation techniques require paired data of mixtures and the corresponding clean speech signals from each speaker for training. This paired training data is impossible to obtain in realistic scenarios and instead is simulated, as also done in this thesis. Incorporating speech foundation models for speech separation could potentially leverage real recorded mixtures for speech separation in realistic meeting scenarios. Foundation models are trained in unsupervised or semi-supervised ways and therefore circumvent the problem of needing paired training data.

[3] https://github.com/fgnt/graph_pit

A Appendix

A.1 Neural Network Architectures and Model Configurations

This subsection gives a short overview over the NN architectures for NN-based speech separation systems used in this thesis.

A.1.1 Common Configuration

All models are trained using loss functions described in Chapter 4 and Chapter 5. For clean data, the target signals are the padded sources $\check{\mathbf{s}}_u$ and for reverberant data, the target signal is chosen as the early reflections (including the direct signal) $\check{\mathbf{s}}^{(\text{early})}$. All models are optimized with the Adam optimizer [48] with an initial learning rate of $1{\times}10^{-3}$. For evaluation, the checkpoints are selected from the training that minimize the loss on the development data. Some models incorporate a learning rate schedule, which is described for each model individually if applied. The total number of training iterations is capped in order to limit the experiment turnaround times. The model architectures and configurations for each model used in the thesis is detailed below.

A.1.2 Dual-Path RNN for Anechoic Speech Separation

The DPRNN first transforms the input signal into a latent feature space using a learned one-dimensional convolutional layer. The transformed input signal is segmented into chunks and RNNs are applied alternately across the time and the chunk axis. The resulting output of the stack of RNN layers is interpreted as masks (one for each speaker) in the latent feature space and multiplied with the latent feature representation of the input signal, resulting in K masked latent representations. The masked signals are then transformed back to the time domain using a learned transposed convolutional layer. Even though the original work used a learning rate schedule, we did not observe a meaningful improvement with it in our experiments.

The two RNNs essentially process the data on different resolutions, facilitating information transform across long distances. The DPRNN has been extended to a Multi-Path RNN (MPRNN) [OC12], extending the hierarchy to more than two resolutions. The MPRNN showed improved performance for very long sequences, but we stick to the more widely used DPRNN here.

A.1.2.1 Large Model

The large DPRNN configuration is copied from the original work [6]. The model has as an encoder a one-dimensional convolutional layer with a kernel size of 16 and a stride

of 8 mapping to a 64-dimensional feature space. The transformed signal is chunked into overlapping segments of 100 frames with a shift of 50 frames.

The core separation network consists of a projection layer followed by six dual-path blocks, where each dual-path block contains one inter-chunk and one intra-chunk network. The projection layer maps from 64 to 64 dimensions. Both the inter-chunk network and intra-chunk network are BLSTMs with identical configuration followed by a linear and a normalization layer. The BLSTMs map to 128 dimensions and the linear layer projects back to 64 dimensions. Skip connections are added around the inter-chunk and intra-chunk networks. A final projection layer projects the 64-dimensional output to 128 dimensions, which is then split into two 64-dimensional masks, one for each output stream. The decoder is a transposed convolutional layer that mirrors the encoder with a kernel size of 16 and a stride of 8.

The network has overall 2.6 M parameters.

A.1.2.2 Small Model

To speed up the experiments, some experiments were performed with a shallower model, comprising only three blocks instead of six. All other parameters are identical to the large model. The number of parameters is reduced to 1.3 M and the training time is roughly halved.

A.1.3 BLSTM for Reverberant Speech Separation

A simple BLSTM-based masking network is used for the initial experiments on reverberant data. The mixture signal is transformed into the STFT domain using a STFT. The network predicts a mask in the STFT domain from the absolute of the STFT with a stack of three BLSTM [117] layers with 600 units each, followed by a single fully connected layer that maps to 514 units. This output is split into $K = 2$ masks, each is multiplied with the mixture's STFT signal and transformed back to the time domain using the iSTFT. The STFT shift and size are 128 and 512, respectively and the Blackman window is used. The network has 23.5 M parameters.

A.1.4 TF-GridNet for Reverberant Speech Separation

The TF-GridNet network architecture [7, 44] works in the STFT domain, as well. The input signal is transformed into the STFT domain using an STFT and real and imaginary parts are concatenated. The multi-channel features from [44] are not used because we only use single-channel signals. For the sample rate of 16 kHz, we use a STFT size of 1024 and shift of 256 and the Blackman window. Then, the signal is mapped to a D-dimensional latent feature space using a 1×1 convolutional layer. Then B blocks are stacked.[1] Each block consists of a global attention module, a frequency-wise cross-frame module and a frame-wise cross-frequency module.

The attention uses a multi-head scaled dot-product attention across all time frames with L heads. To compute the attention keys, queries and values, the embeddings are fed through

[1] Note that this is the notation from [7, 44] and may not be compatible with the notation in the rest of this thesis.

a point-wise convolutional layer that maps to a dimension of E, a PReLU activation function and a normalization, and then stacked. The cross-frame and cross-frequency modules both consist of an unfolding operation across I embeddings with a stride of J, which essentially stacks neighboring embedding vectors to obtain a longer context, a BLSTM with a hidden dimension of H for recursive modelling and a 1D transposed convolution to invert the unfolding operation and map back to D dimensions. The only difference between them is that the cross frame module is recursive across the frames and the cross-frequency module processes all embeddings within a frame across the frequency dimension. Each sub-net contains a skip connection from its input to its output.

TF-GridNet performs spectral mapping at the output, where the network output is directly interpreted as the STFT representation of the separated signals instead of as a mask and multiplied with the mixture STFT. Also note that, while both the DPRNN and TF-GridNet apply a recurrent network alternately along different dimensions of a high-dimensional latent representation, the direction in which the RNNs are applied are significantly different. The DPRNN alternates across a high- and a low-resolution RNN while the TF-GridNet alternates between the time and the frequency direction.

During training, we employ a fixed learning rate schedule that halves the learning rate three times over the course of a training.

A.1.4.1 Original Large Model (TF-GridNet-Large)

The large model is chosen similarly to the original paper, using $B = 4$ blocks, an embedding dimension of $D = 48$, $L = 4$ attention heads with a dimension of $E = 2$ for the keys and queries, a unfold kernel size $I = 4$ and stride $J = 1$ an Long-Short-Term Memory (LSTM) size of $H = 192$.

A second large model is introduced for the final experiment that employs all improvements made to the small model introduced below. It has a slightly smaller size, due to some architectural changes, and achieves a significantly better performance. It is equal to the modified small model brought back up to the original size, and was trained on a larger training dataset, including training data from the CHiME-6 and DiPCo datasets. It is the last row in Table A.1

A.1.4.2 Optimized Small Model (TF-GridNet-Small)

To speed up the experiments, the model size was reduced to $B = 2$ and the hidden dimension of the BLSTM was reduced to $H = 89$. To increase the performance, Dynamic Range Compression (DRC) [118] was introduced, taking the square root of the amplitude in the STFT domain and reverting it after the separation before computing the iSTFT. Additionally, the BLSTM in the frame-wise cross-frequency module was replaced with a Transform Average Concatenate (TAC) [119] module so that no recursive processing is required in this module. Replacing the BLSTM in the cross-frame module degraded the performance significantly in preliminary experiments while it did not impact the performance in the frame-wise cross-frequency module. Combining these architectural changes, the model size could be reduced to 0.99 M parameters while improving the performance.

The effect of these modifications is analyzed in Table A.1. All models in this table were trained on the same artificially generated mixtures from the LibriSpeech dataset, including one- and two-speaker mixtures of LibriSpeech utterances that were artificially reverberated.

Table A.1: Performance of TF-GridNet architectures on LibriCSS under different modifications.

Model configuration	#Param.	tcORC-WER↓	Used in
Large	5.8 M	6.8	[OC1],Chapter 6
Reduced size ($B = 2$ and $H = 89$)	1.2 M	11.4	
+ DRC	1.2 M	8.6	
+ TAC (cross-frame & cross-freq.)	0.8 M	8.3	
+ TAC (cross-freq.)	0.99 M	6.7	
+ MUSAN augmentation	0.99 M	5.1	Section 4.4.3
Large-v2 (DRC + TAC (cross-freq.) + MUSAN + DiPCo + CHiME-6)	4.6 M	4.8	Chapter 6

Training was performed using uPIT and the SA-SDR loss function. The separated signals were transcribed with the Whisper speech recognizer in the `large-v2` configuration in order to compute tcORC-WER scores. Only reducing the model size brings the number of parameters down by a factor of almost 5 to 1.2 M parameters while the performance in terms of tcORC-WER is reduced to 11.4 %. The dynamic range compression brings it down to 8.6 %. Using the TAC layer reduces the model size further down to 0.99 M parameters and reduces the tcORC-WER to 6.7 %, comparable to the large model. When additional noise augmentation with MUSAN noise is employed, the performance can further be improved to 5.1 %. These changes additionally stabilized the training. This is the model architecture used in Section 4.4.3.

A.1.4.3 Optimized Large Model (TF-GridNet-Large-v2)

The final line in Table A.1 shows the performance when incorporating the same modifications into the large model and extending the training data with mixtures generated from the single-speaker regions of the DiPCo and CHiME-6 datasets. This model is named "TF-GridNet-Large-v2" and is used to obtain the final performance in Chapter 6.

A.2 Proof That Every Overlap Graph is Chordal

A graph $\mathcal{G}$ is chordal if and only if every induced subgraph has at least one simplical vertex, where a vertex v is simplical if and only if the subgraph $\mathcal{G}[N[v]]$ induced by its neighborhood $N[v]$ is complete [120], i.e., a fully connected graph. We show that every overlap graph $\mathcal{G}^{(\mathrm{ov})}$ is chordal by showing that every induced subgraph of $\mathcal{G}^{(\mathrm{ov})}$ is again an overlap graph and then showing that every overlap graph has at least one simplical vertex.

Every induced subgraph of an overlap graph is an overlap graph. Let $\mathcal{G}^{(\mathrm{ov})}(\mathcal{V}) = (\mathcal{V}, \mathcal{E})$ be an overlap graph and $\mathcal{V}' \subseteq \mathcal{V}$ be a subset of its vertices. The graph $\mathcal{G}^{(\mathrm{ov})}(\mathcal{V}') = (\mathcal{V}', \mathcal{E}')$ is the overlap graph constructed from the vertices $\mathcal{V}'$ and the edges $\mathcal{E}' = \{\{u, v\} : u, v \in \mathcal{V}', o(u, v)\}$. The induced subgraph $\mathcal{G}^{(\mathrm{ov})}(\mathcal{V})[\mathcal{V}']$ can be expressed just with the vertices

$\mathcal{V}'$ and the edges $\mathcal{E}'$:

$$\mathcal{G}^{(\mathrm{ov})}(\mathcal{V})[\mathcal{V}'] = (\mathcal{V}', \{\{u,v\} \in \mathcal{E} : u \in \mathcal{V}' \wedge v \in \mathcal{V}'\}) \tag{A.1}$$
$$= (\mathcal{V}', \{\{u,v\} : u \in \mathcal{V}' \wedge v \in \mathcal{V}' \wedge o(u,v) = 1\}) \tag{A.2}$$
$$= \mathcal{G}^{(\mathrm{ov})}(\mathcal{V}'), \tag{A.3}$$

where the definition of the overlap graph from Eq. (5.35). This proves that every induced subgraph of an overlap graph is again an overlap graph.

Every overlap graph has at least one simplical vertex. A simplical vertex v is a vertex whose neighborhood $N[v]$ induces a complete subgraph. Let $\mathcal{G}^{(\mathrm{ov})} = (\mathcal{V}, \mathcal{E})$ be an overlap graph and let v_{last} be the vertex corresponding to the utterance with the latest begin time in $\mathcal{G}^{(\mathrm{ov})}$. Then, any two utterances in the neighborhood of v_{last} overlap:

$$\forall v \in N[v_{\mathrm{last}}] : b_v \leqslant b_{v_{\mathrm{last}}} \wedge e_v \geqslant b_{v_{\mathrm{last}}} \tag{A.4}$$
$$\Rightarrow \forall u, v \in N[v_{\mathrm{last}}] : b_u < b_{v_{\mathrm{last}}} < e_v \tag{A.5}$$
$$\Rightarrow \forall u, v \in N[v_{\mathrm{last}}] : o(u,v) = 1 \tag{A.6}$$
$$\Rightarrow \forall u, v \in N[v_{\mathrm{last}}] : \{u,v\} \in \mathcal{E}. \tag{A.7}$$

From this follows that the neighborhood $N[v_{\mathrm{last}}]$ induces a complete subgraph and thus v_{last} is simplical.

A.3 Additional Decompositions for Graph-PIT

In the main text, only the most general decompositions were presented. Here, two more decompositions of the SA-SI-SDR and the SA-CI-SDR are shown.

A.3.1 SA-SI-SDR

The SI-SDR is defined for a single pair of ground-truth and estimated signals $\mathbf{s}$ and $\hat{\mathbf{s}}$ as (compare Eq. (4.20))

$$\mathcal{L}^{(\text{SI-SDR})}(\mathbf{s}, \hat{\mathbf{s}}) = -10 \log_{10} \frac{\|\alpha \mathbf{s}\|^2}{\|\alpha \mathbf{s} - \hat{\mathbf{s}}\|^2}, \tag{A.8}$$

where the scaling factor

$$\alpha = \arg\min_{\tilde{\alpha}} \|\tilde{\alpha}\mathbf{s} - \hat{\mathbf{s}}\|^2 = \frac{\mathbf{s}^\mathsf{T}\hat{\mathbf{s}}}{\mathbf{s}^\mathsf{T}\mathbf{s}} \tag{A.9}$$

scales the reference signal to match the scale of the estimated signal, i.e., to minimize the error. It is constant over the full utterance.

When applied to Graph-PIT, we assume that the scaling factor α is constant within an utterance but can change between utterances[2], e.g., due to speaker movement. For Graph-PIT SA-SI-SDR, we thus estimate one scaling factor for every combination of utterance and

[2] This assumption is arbitrary but plausible; the scaling factor can also be optimized across multiple utterances, e.g., a single scaling factor for all utterances of the same speaker. This would, however, involve a more complex optimization.

stream:

$$\alpha_{u,s} = \frac{\breve{\mathbf{s}}_u^\mathsf{T} \hat{\mathbf{s}}_s}{\breve{\mathbf{s}}_u^\mathsf{T} \breve{\mathbf{s}}_u}. \tag{A.10}$$

The zeros in the padded source signal $\breve{\mathbf{s}}_u$ mask out the parts of the estimated signal outside of the boundaries of the u-th utterance.

To formulate the SA-SI-SDR for Graph-PIT, we now write out the squared norm of the matrix multiplication as $\|\mathbf{SC}\|^2 = \sum_{s=1}^{S} \|\sum_{u\in\mathcal{U}} \breve{\mathbf{s}}_u c_{u,s}\|^2$, where $c_{u,s}$ is an element of the assignment matrix that is 1 if utterance u is assigned to stream s and 0 otherwise. Also writing out the squared norm as $\|\mathbf{s}\|^2 = \mathbf{s}^\mathsf{T}\mathbf{s}$, we obtain:

$$\mathcal{L}^{\text{(GP-SA-SI-SDR)}}(\mathbf{S}, \hat{\mathbf{S}}) = \min_{\mathbf{C}\in\mathcal{C}} -10\log_{10} \frac{\sum_{s=1}^{S} \left\|\sum_{u\in\mathcal{U}} c_{u,s}\alpha_{u,s}\breve{\mathbf{s}}_u\right\|^2}{\sum_{s=1}^{S} \left\|\sum_{u\in\mathcal{U}} c_{u,s}\alpha_{u,s}\breve{\mathbf{s}}_u - \hat{\mathbf{s}}_s\right\|^2} \tag{A.11}$$

$$= \min_{\mathbf{C}\in\mathcal{C}} -10\log_{10} \frac{\sum_{s=1}^{S} \sum_{u\in\mathcal{U}} \sum_{u'\in\mathcal{U}} c_{u,s}c_{u',s}\alpha_{u,s}\alpha_{u',s}\breve{\mathbf{s}}_u^\mathsf{T}\breve{\mathbf{s}}_{u'}}{\sum_{s=1}^{S} \left(\sum_{u\in\mathcal{U}} \sum_{u'\in\mathcal{U}} c_{u,s}c_{u',s}\alpha_{u,s}\alpha_{u',s}\breve{\mathbf{s}}_u^\mathsf{T}\breve{\mathbf{s}}_{u'} - 2\sum_{u\in\mathcal{U}} c_{u,s}\alpha_{u,s}\breve{\mathbf{s}}_u^\mathsf{T}\hat{\mathbf{s}}_s + \hat{\mathbf{s}}_s^\mathsf{T}\hat{\mathbf{s}}_s\right)}. \tag{A.12}$$

In the numerator and denominator, we can use the fact that either $\breve{\mathbf{s}}_u^\mathsf{T}\breve{\mathbf{s}}_{u'} = 0$ (if the utterances do not overlap) or $c_{u,s}c_{u',s} = 0$ (if the utterances overlap) to remove the sum over u':

$$\mathcal{L}^{\text{(GP-SA-SI-SDR)}}(\mathbf{S}, \hat{\mathbf{S}}) = \min_{\mathbf{C}\in\mathcal{C}} -10\log_{10} \frac{\sum_{s=1}^{S} \sum_{u\in\mathcal{U}} c_{u,s}\alpha_{u,s}^2\breve{\mathbf{s}}_u^\mathsf{T}\breve{\mathbf{s}}_u}{\sum_{s=1}^{S} \left(\sum_{u\in\mathcal{U}} \left(c_{u,s}\alpha_{u,s}^2\breve{\mathbf{s}}_u^\mathsf{T}\breve{\mathbf{s}}_u - 2c_{u,s}\alpha_{u,s}\breve{\mathbf{s}}_u^\mathsf{T}\hat{\mathbf{s}}_s\right) + \hat{\mathbf{s}}_s^\mathsf{T}\hat{\mathbf{s}}_s\right)}. \tag{A.13}$$

Plugging in $\alpha_{u,s}\breve{\mathbf{s}}_u^\mathsf{T}\breve{\mathbf{s}}_u = \breve{\mathbf{s}}_u^\mathsf{T}\hat{\mathbf{s}}_s$ from Eq. (A.10) and using $\log(a/b) = -\log(b/a)$, we get:

$$\mathcal{L}^{\text{(GP-SA-SI-SDR)}}(\mathbf{S}, \hat{\mathbf{S}}) = \min_{\mathbf{C}\in\mathcal{C}} -10\log_{10} \frac{\sum_{s=1}^{S} \sum_{u\in\mathcal{U}} c_{u,s}\alpha_{u,s}\breve{\mathbf{s}}_u^\mathsf{T}\hat{\mathbf{s}}_s}{\sum_{s=1}^{S} \left(\sum_{u\in\mathcal{U}} c_{u,s}\alpha_{u,s}\breve{\mathbf{s}}_u^\mathsf{T}\hat{\mathbf{s}}_s + \hat{\mathbf{s}}_s^\mathsf{T}\hat{\mathbf{s}}_s\right)} \tag{A.14}$$

$$= 10\log_{10} \left(\frac{\sum_{s=1}^{S} \hat{\mathbf{s}}_s^\mathsf{T}\hat{\mathbf{s}}_s}{\min_{\mathbf{C}\in\mathcal{C}} \sum_{s=1}^{S} \sum_{u\in\mathcal{U}} c_{u,s}\alpha_{u,s}\breve{\mathbf{s}}_u^\mathsf{T}\hat{\mathbf{s}}_s} - 1 \right). \tag{A.15}$$

Now, only the denominator of the inner fraction depends on the assignment matrix $\mathbf{C}$, so the score matrix can be found as

$$\mathbf{M} = [\alpha_{u,s}\breve{\mathbf{s}}_u^\mathsf{T}\hat{\mathbf{s}}_s]_{u,s} = \left[\frac{\breve{\mathbf{s}}_u^\mathsf{T}\hat{\mathbf{s}}_s}{\breve{\mathbf{s}}_u^\mathsf{T}\breve{\mathbf{s}}_u} \breve{\mathbf{s}}_u^\mathsf{T}\hat{\mathbf{s}}_s \right]_{u,s}. \tag{A.16}$$

Instead of computing the scalar product across the full meeting length T with the padded source signals, the same result is obtained by cutting the respective parts from the estimated signal and computing the scalar product only on the region where the source signal is not zero-padded. The outer function is

$$f(x, \mathbf{S}, \hat{\mathbf{S}}) = 10\log_{10} \left(\frac{\left\|\hat{\mathbf{S}}\right\|^2}{x} - 1 \right). \tag{A.17}$$

A.3.2 SA-CI-SDR

The SA-CI-SDR is defined in Eq. (5.45) as

$$\text{SA-CI-SDR} = \max_{\mathbf{C}\in\mathcal{C}} 10\log_{10} \frac{\sum_s \left\|\sum_u c_{u,s}\mathbf{a}_{u,s} * \breve{\mathbf{s}}_u\right\|^2}{\sum_s \left\|\sum_u c_{u,s}\mathbf{a}_{u,s} * \breve{\mathbf{s}}_u - \hat{\mathbf{s}}_s\right\|^2}, \tag{A.18}$$

with $\mathbf{a}_{u,s} = \arg\min_{\tilde{\mathbf{a}}} \left\|\tilde{\mathbf{a}} * \breve{\mathbf{s}}_u - \hat{\mathbf{s}}_s\right\|^2$. The signal parts in $\hat{\mathbf{s}}_s$ that lie outside of the boundaries of the convolved utterance, i.e., starting at b_u and ending at $e_u + \dim(\mathbf{a}) - 1$, are constant w.r.t. the optimization of $\mathbf{a}$ (since $\breve{\mathbf{s}}$ is 0 there) and thus have no effect on $\mathbf{a}$. The full convolution is truncated to the length of T. Note that the SA-CI-SDR is formulated as a performance measure (higher value is better) and not as a loss (lower value is better), so that the goal is here to maximize its value.

We can write the discrete convolution $\mathbf{a} * \breve{\mathbf{s}}$ as a matrix multiplication $\mathbf{T}\mathbf{a}$, where $\mathbf{T} = \text{toeplitz}(\breve{\mathbf{s}}) \in \mathbb{R}^{T\times\dim(\mathbf{a})}$ is a padded toeplitz matrix created from $\breve{\mathbf{s}}$, so that vector gradient rules can be used for the derivative of the convolution. The least-squares minimization then becomes

$$\mathbf{a}_{u,s} = \arg\min_{\tilde{\mathbf{a}}} \left\|\mathbf{T}_u\tilde{\mathbf{a}} - \hat{\mathbf{s}}_s\right\|^2 \tag{A.19}$$

$$\frac{\mathrm{d}\left\|\mathbf{T}_u\mathbf{a}_{u,s} - \hat{\mathbf{s}}_s\right\|^2}{\mathrm{d}\mathbf{a}_{u,s}} = 2\mathbf{T}_u^\mathsf{T}\mathbf{T}_u\mathbf{a}_{u,s} - 2\mathbf{T}_u^\mathsf{T}\hat{\mathbf{s}}_s \overset{!}{=} 0 \tag{A.20}$$

$$\Rightarrow \mathbf{T}_u^\mathsf{T}\mathbf{T}_u\mathbf{a}_{u,s} = \mathbf{T}_u^\mathsf{T}\hat{\mathbf{s}}_s. \tag{A.21}$$

Plugging Eq. (A.21) into Eq. (A.18) yields a solution very similar to the SA-SI-SDR in Eq. (A.15):

$$\text{SA-CI-SDR} = \max_{\mathbf{C}\in\mathcal{C}} 10\log_{10} \frac{\sum_s \left\|\sum_u c_{u,s}\mathbf{T}_u\mathbf{a}_{u,s}\right\|^2}{\sum_s \left\|\sum_u c_{u,s}\mathbf{T}_u\mathbf{a}_{u,s} - \hat{\mathbf{s}}_s\right\|^2} \tag{A.22}$$

$$= -10\log_{10}\left(\frac{\sum_s \hat{\mathbf{s}}_s^\mathsf{T}\hat{\mathbf{s}}_s}{\max_{\mathbf{C}\in\mathcal{C}}\sum_{u,s} c_{u,s}\mathbf{a}_{u,s}^\mathsf{T}\mathbf{T}_u^\mathsf{T}\hat{\mathbf{s}}_s} - 1\right) \tag{A.23}$$

$$= -10\log_{10}\left(\frac{\sum_s \hat{\mathbf{s}}_s^\mathsf{T}\hat{\mathbf{s}}_s}{\max_{\mathbf{C}\in\mathcal{C}}\sum_{u,s} c_{u,s}(\mathbf{a}_{u,s} * \breve{\mathbf{s}}_u)^\mathsf{T}\hat{\mathbf{s}}_s} - 1\right) \tag{A.24}$$

The decomposition is similar to the one for SA-SI-SDR, but with

$$\mathbf{M} = \left[(\mathbf{a}_{u,s} * \breve{\mathbf{s}}_u)^\mathsf{T}\hat{\mathbf{s}}_s\right]_{u,s}. \tag{A.25}$$

Symbols and Notation

Vectors are denoted with lower-case bold letters (e.g., $\mathbf{s}$) and matrices are denoted with upper-case bold letters (e.g., $\mathbf{S}$). Row and column vectors are not discriminated in the notation. Sets are denoted with a calligraphic font (e.g., $\mathcal{C}, \mathcal{P}$). Estimated values, i.e., at the system output, are denoted with a hat (e.g., $\hat{\mathbf{s}}$) and zero-padded signals with a breve symbol (e.g., $\breve{\mathbf{s}}$). If the same concept appears in different places with slight differences, it is discriminated with a superscript text in parenthesis, e.g., $\mathbf{L}$, $\mathbf{L}^{(\mathrm{ORC})}$. Most notation is introduced where it is first needed.

Basic operations

$[\mathbf{a}, \mathbf{b}], \begin{bmatrix}\mathbf{a}\\ \mathbf{b}\end{bmatrix}$	Concatenation or stacking of the two vectors or matrices $\mathbf{a}$ and $\mathbf{b}$ along the first or second dimension
$\dim(\mathbf{a})$	The dimension of vector $\mathbf{a}$, i.e., the number of elements in $\mathbf{a}$
$(\cdot)^{\mathsf{T}}$	Vector / matrix transpose
$\nabla_{\mathbf{x}} f(\mathbf{x}) = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$	The gradient or vector derivative of the (scalar-valued) function $f(\mathbf{x})$ w.r.t. $\mathbf{x}$
$\mathbf{a}_{1,\ldots,N} = (\mathbf{a}_1, ..., \mathbf{a}_N)$	Shorthand notation for a tuple of vectors with consecutive indices
$\mathcal{O}(\cdot)$	Big $\mathcal{O}$ notation for complexity, denoting the upper limiting complexity class
$\|\cdot\|_1, \|\cdot\|, \|\cdot\|^2$	L1 norm, L2 norm and squared L2 norm for vectors and matrices
$\mathrm{tr}(\cdot)$	Matrix trace

Meeting transcription All signals are represented as column vectors of samples. All estimated quantities are marked with a hat $\hat{(.)}$.

$\mathcal{U}$	The set of ground-truth utterances
K, k	The number of speakers and the corresponding index
S, s	The number of system outputs (streams) or the estimated number of speakers and the corresponding index

$\hat{\mathcal{U}}$	The estimated set of utterances
$u \in \mathcal{U}$	An utterance index
$b_u, \hat{b}_u$	The ground-truth and estimated begin time (in samples or seconds, depending on the context) of the u-th ground-truth or estimated utterance
$e_u, \hat{e}_u$	The ground-truth and estimated end time (in samples or seconds, depending on the context) of the u-th ground-truth or estimated utterance
$\ell_u, \hat{\ell}_u$	The ground-truth and estimated speaker labels. The estimated speaker labels may not represent the true speaker identity but may represent only the system output stream index.
$\mathbf{t}_u, \hat{\mathbf{t}}_u$	The ground-truth and estimated transcript for utterance u, as a row vector representing a sequence of words
$t_{u,i}, \hat{t}_{u,i}$	The i-th word in the ground-truth or estimated transcript for utterance u
$\mathbf{y}$	The observed speech mixture signal
$\mathbf{n}$	The noise signal
$\mathbf{h}_u$	The Room Impulse Response for reverberating utterance u
$\mathbf{s}_u$	Ground-truth source signal for utterance $u \in \mathcal{U}$ of length $\dim(\mathbf{s}_u) = e_u - b_u$
$\breve{\mathbf{s}}_u$	The ground-truth source signal for utterance $u \in \mathcal{U}$ padded to the full recording length $\dim(\mathbf{y})$
$\hat{\mathbf{s}}_u$	Estimated separated signal for estimated utterance $u \in \hat{\mathcal{U}}$
$\mathbf{S}, \hat{\mathbf{S}}$	Padded and stacked ground-truth source signals and estimated signals
$o(u_1, u_2)$	The overlap function that indicates whether the two utterances u_1 and u_2 overlap temporally
$<_{\mathrm{u}}$	The utterance order relation, where $u_1 <_{\mathrm{u}} u_2$ if u_1 comes before u_2

Word error rates

$C_{\mathrm{D}}, C_{\mathrm{I}}, C_{\mathrm{S}}, C_{\mathrm{C}}$	Cost for a deletion, insertion, substitution or correct match
$C_{\mathrm{C/S}}$	Cost for a match, either a correct match or a substitution
$(\cdot)_{<i}, (\cdot)_{\geqslant i}$	The sub-vector of the first i elements or the last i elements, respectively
$\mathrm{tail}(\mathbf{a})$	The last element from $\mathbf{a}$

$\text{head}(\mathbf{a})$	All but the last element from $\mathbf{a}$
$\mathbf{L}$	The Levenshtein matrix or high-dimensional Levenshtein tensor
$l_{i,j}, l_{i,s,\mathbf{h}}$	One entry of the Levenshtein matrix or tensor
c	The collar for the time-constrained WER
$\mathbf{t}_{\ell,k}^{(\text{cat})}, \hat{\mathbf{t}}_{\hat{\ell},s}^{(\text{cat})}$	The concatenation of the transcripts of all utterances or estimated utterances
$\mathbf{t}_{k}^{(\text{spk})}, \hat{\mathbf{t}}_{s}^{(\text{str})}$	The concatenation of the transcripts of all utterances of speaker k or with estimated speaker or stream label s

uPIT, graph theory and Graph-PIT

$\boldsymbol{\pi}, \mathbf{P}$	A permutation vector and permutation matrix, used interchangeably where applicable
$\boldsymbol{\pi}^*, \mathbf{P}^*$	The optimal permutation vector or matrix
$\mathcal{P}(K)$	The set of all $K!$ permutation vectors or matrices of size K
$\mathcal{G}$	An undirected graph
$\mathcal{V}$	The set of vertices of a graph
$\mathcal{E}$	The set of edges of a graph
$N[v]$	The neighborhood of a vertex v
$\mathbf{C}$	A coloring and assignment matrix
$\mathbf{C}^*$	The optimal coloring
$\mathcal{C}_\mathcal{G}$	The set of all proper vertex colorings of graph $\mathcal{G}$
$\mathbf{c}$	A row of $\mathbf{C}$ representing a color as a one-hot row vector
$c_{u,s}$	An element of $\mathbf{C}$ that is 1 if utterance u is assigned to stream s and 0 otherwise
$\mathbf{M}$	The score or loss matrix
α, β	The scale-invariance factors in the SI-SDR and OSI-SDR
$f, f^{(\text{score})}$	The decomposed functions required for efficient computation of the uPIT permutation and Graph-PIT assignment
ω	The clique number of a graph
$\Xi(\mathcal{G})$	The chromatic number of the graph $\mathcal{G}$, i.e., the minimum number of colors required to properly color $\mathcal{G}$
τ	A threshold

List of Figures

List of Tables

Acronyms

A-SDR averaged SDR.

ASR Automatic Speech Recognition.

BCE Binary Cross Entropy.

BLSTM Bidirectional LSTM.

CHiME Computational Hearing in Multi-source Environments.

CI-SDR Convolution-Invariant SDR.

CNN Convolutional Neural Network.

cpWER Concatenated minimum-Permutation WER.

CSS Continuous Speech Separation.

DA-WER Diarization-Aware WER.

DER Diarization Error Rate.

DFS Depth-First Search.

DI-cpWER Diarization Invariant cpWER.

DI-tcpWER time-constrained DI-cpWER.

DiPCo Dinner Party Corpus.

DPRNN Dual-Path RNN.

DRC Dynamic Range Compression.

EER Equal Error Rate.

GSS Guided Source Separation.

log-MAE log Mean Absolute Error.

LSTM Long-Short-Term Memory.

MAE Mean Absolute Error.

MIMO-WER Multiple Input Multiple Output WER.

MIMO-WER time-constrained MIMO-WER.

MPRNN Multi-Path RNN.

MSE Mean Squared Error.

MT-RNN-T Multi-turn RNN-T.

NN Neural Network.

ORC-WER Optimal Reference Combination WER.

OSI-SDR Optimal Scale-Invariant Signal-to-Distortion Ratio.

RIR Room Impulse Response.

RNN Recurrent Neural Network.

RPN Region Proposal Network.

SA-SDR Source-Aggregated Signal-to-Distortion Ratio.

SDR Signal-to-Distortion Ratio.

SGD Stochastic Gradient Descent.

SI-SDR Scale-Invariant Signal-to-Distortion Ratio.

SOT Serialized Output Training.

STFT Short-term Fourier Transformation.

TAC Transform Average Concatenate.

tcORC-WER Time-Constrained ORC-WER.

tcpWER Time-Constrained minimum-Permutation WER.

uPIT Utterance-level Permutation Invariant Training.

VAD Voice Activity Detection.

VAER Voice Activity Error Rate.

WER Word Error Rate.

Bibliography

Peer-reviewed publications with own contributions

[OC1] **T. von Neumann**, C. Boeddeker, T. Cord-Landwehr, M. Delcroix, and R. Haeb-Umbach, "Meeting recognition with continuous speech separation and transcription-supported diarization," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing Workshops (ICASSPW)*, 2024, pp. 775–779 (cited on pp. 1, 9, 103, 105–108, 110, 111, 119).

[OC2] T. Cord-Landwehr, **T. von Neumann**, C. Boeddeker, and R. Haeb-Umbach, "MMS-MSG: A multi-purpose multi-speaker mixture signal generator," in *International Workshop on Acoustic Signal Enhancement (IWAENC)*, 2022 (cited on pp. 14, 93, 107).

[OC3] **T. von Neumann**, K. Kinoshita, C. Boeddeker, M. Delcroix, and R. Haeb-Umbach, "Segment-less continuous speech separation of meetings: Training and evaluation criteria," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 31, pp. 576–589, 2023 (cited on pp. 16–18, 21, 81, 82, 93, 95–98).

[OC4] **T. von Neumann**, C. Boeddeker, K. Kinoshita, M. Delcroix, and R. Haeb-Umbach, "Speeding up permutation invariant training for source separation," in *Speech Communication; 14th ITG-Symposium*, 2021 (cited on pp. 16, 81, 83, 98, 99).

[OC5] **T. von Neumann**, K. Kinoshita, C. Boeddeker, M. Delcroix, and R. Haeb-Umbach, "Graph-PIT: Generalized permutation invariant training for continuous separation of arbitrary numbers of speakers," in *Interspeech*, 2021 (cited on pp. 18, 21, 22, 70, 78, 81).

[OC6] **T. von Neumann**, K. Kinoshita, C. Boeddeker, M. Delcroix, and R. Haeb-Umbach, "SA-SDR: A novel loss function for separation of meeting style data," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022 (cited on pp. 24, 69, 70, 77).

[OC7] **T. von Neumann**, C. Boeddeker, K. Kinoshita, M. Delcroix, and R. Haeb-Umbach, "On word error rate definitions and their efficient computation for multi-speaker speech recognition systems," 2023 (cited on pp. 29, 30, 42, 43, 53, 58).

[OC8] **T. von Neumann**, C. Boeddeker, M. Delcroix, and R. Haeb-Umbach, "Word error rate definitions and algorithms for long-form multi-talker speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 33, pp. 3174–3188, 2025 (cited on pp. 29, 30, 39, 40, 42, 44, 46, 47, 54, 59, 61–66).

[OC9] K. Kinoshita, **T. von Neumann**, M. Delcroix, C. Boeddeker, and R. Haeb-Umbach, "Utterance-by-utterance overlap-aware neural diarization with graph-pit," in *Interspeech*, 2022, pp. 1486–1490 (cited on p. 87).

[OC10] T. Cord-Landwehr, C. Boeddeker, **T. von Neumann**, C. Zorilă, R. Doddipatla, and R. Haeb-Umbach, "Monaural source separation: From anechoic to reverberant environments," 2022 (cited on p. 97).

[OC11] P. Vieting, S. Berger, **T. von Neumann**, C. Boeddeker, R. Schlüter, and R. Haeb-Umbach, "Error analysis in a modular meeting transcription system," in *Speech Communication; 16th ITG-Symposium*, 2025 (cited on p. 103).

[OC12] K. Kinoshita, **T. von Neumann**, M. Delcroix, T. Nakatani, and R. Haeb-Umbach, "Multi-path RNN for hierarchical modeling of long sequential data and its application to speaker stream separation," in *Interspeech*, 2020, pp. 2652–2656 (cited on p. 116).

[OC13] C. Boeddeker, T. Cord-Landwehr, **T. von Neumann**, and R. Haeb-Umbach, "An initialization scheme for meeting separation with spatial mixture models," in *Interspeech*, 2022, pp. 271–275.

[OC14] P. Vieting, S. Berger, **T. von Neumann**, C. Boeddeker, R. Schlüter, and R. Haeb-Umbach, "Combining TF-GridNet and mixture encoder for continuous speech separation for meeting transcription," in *IEEE Spoken Language Technology Workshop (SLT)*, 2024, pp. 155–162.

Non-Peer-reviewed publications with own contributions

[OC15] C. Boeddeker, T. Cord-Landwehr, **T. von Neumann**, and R. Haeb-Umbach, "Multi-stage diarization refinement for the CHiME-7 DASR scenario," in *International Workshop on Speech Processing in Everyday Environments (CHiME)*, 2023, pp. 51–56 (cited on pp. 1, 8, 61).

[OC16] **T. von Neumann**, C. Boeddeker, M. Delcroix, and R. Haeb-Umbach, "MeetEval: A toolkit for computation of word error rates for meeting transcription systems," in *International Workshop on Speech Processing in Everyday Environments (CHiME)*, 2023, pp. 27–32 (cited on pp. 29–31, 33, 34, 36, 37).

Pre-graduation publications with own contributions

[OC17] **T. von Neumann**, K. Kinoshita, M. Delcroix, S. Araki, T. Nakatani, and R. Haeb-Umbach, "All-neural online source separation, counting, and diarization for meeting analysis," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 91–95 (cited on p. 4).

[OC18] **T. von Neumann** et al., "Multi-talker ASR for an unknown number of sources: Joint training of source counting, separation and ASR," in *Interspeech*, 2020, pp. 3097–3101 (cited on pp. 71, 78).

[OC19] L. Drude, **T. von Neumann**, and R. Haeb-Umbach, "Deep attractor networks for speaker re-identification and blind source separation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 11–15.

[OC20] **T. von Neumann** et al., "End-to-end training of time domain audio separation and recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 7004–7008.

Other References

[1] A. Vinnikov et al., "NOTSOFAR-1 challenge: New datasets, baseline, and tasks for distant meeting transcription," in *Interspeech*, 2024, pp. 5003–5007 (cited on pp. 1, 14, 38, 68).

[2] S. Cornell et al., "The CHiME-7 DASR challenge: Distant meeting transcription with multiple devices in diverse scenarios," in *International Workshop on Speech Processing in Everyday Environments (CHiME)*, 2023 (cited on pp. 1, 8, 14, 60).

[3] S. Cornell et al., "The CHiME-8 DASR challenge for generalizable and array agnostic distant automatic speech recognition and diarization," in *International Workshop on Speech Processing in Everyday Environments (CHiME)*, 2024 (cited on pp. 1, 68).

[4] J. R. Hershey, Z. Chen, J. Le Roux, and S. Watanabe, "Deep clustering: Discriminative embeddings for segmentation and separation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 31–35 (cited on pp. 1, 13, 15).

[5] Y. Luo and N. Mesgarani, "TaSNet: Time-domain audio separation network for real-time, single-channel speech separation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 696–700 (cited on pp. 1, 13, 74, 75, 78).

[6] Y. Luo, Z. Chen, and T. Yoshioka, "Dual-path RNN: Efficient long sequence modeling for time-domain single-channel speech separation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 46–50 (cited on pp. 1, 74, 116).

[7] Z.-Q. Wang, S. Cornell, S. Choi, Y. Lee, B.-Y. Kim, and S. Watanabe, "Tf-gridnet: Making time-frequency domain models great again for monaural speaker separation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023 (cited on pp. 1, 74, 75, 117).

[8] R. Prabhavalkar, T. Hori, T. N. Sainath, R. Schlüter, and S. Watanabe, "End-to-end speech recognition: A survey," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 32, pp. 325–351, 2023 (cited on pp. 1, 8).

[9] D. Povey et al., "The kaldi speech recognition toolkit," in *IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2011 (cited on pp. 1, 12, 29, 35, 36, 59).

[10] S. Watanabe et al., "ESPnet: End-to-end speech processing toolkit," in *Interspeech*, 2018, pp. 2207–2211 (cited on pp. 1, 14, 29, 111).

[11] D. O'Shaughnessy, "Speaker diarization: A review of objectives and methods," *Applied Sciences*, vol. 15, no. 4, 2025 (cited on pp. 1, 19).

[12] H. Bredin, "Pyannote.audio 2.1 speaker diarization pipeline: Principle, benchmark, and recipe," in *Interspeech*, 2023, pp. 1983–1987 (cited on pp. 1, 60).

[13] D. Raj et al., "Integration of speech separation, diarization, and recognition for multi-speaker meetings: System description, comparison, and analysis," in *IEEE Spoken Language Technology Workshop (SLT)*, 2021, pp. 897–904 (cited on pp. 1, 8, 9, 111).

[14] Z. Chen et al., "Continuous speech Separation: Dataset and Analysis.," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 7284–7288 (cited on pp. 1, 8, 9, 13, 14, 17, 18, 21, 35, 107).

[15] N. Kamo et al., "NTT multi-speaker ASR system for the DASR task of CHiME-7 challenge," in *International Workshop on Speech Processing in Everyday Environments (CHiME)*, 2023, pp. 45–50 (cited on pp. 1, 8, 61).

[16] K. Deng, X. Zheng, and P. Woodland, "The university of cambridge system for the CHiME-7 DASR task," in *International Workshop on Speech Processing in Everyday Environments (CHiME)*, 2023, pp. 73–76 (cited on pp. 1, 8, 61).

[17] T. Prisyach et al., "STCON system for the CHiME-7 challenge," in *International Workshop on Speech Processing in Everyday Environments (CHiME)*, 2023, pp. 87–92 (cited on pp. 1, 8, 61).

[18] N. Kanda, Y. Gaur, X. Wang, Z. Meng, and T. Yoshioka, "Serialized output training for end-to-end overlapped speech recognition," in *Interspeech*, 2020, pp. 2797–2801 (cited on pp. 1, 43, 112).

[19] N. Kanda et al., "Transcribe-to-diarize: Neural speaker diarization for unlimited number of speakers using end-to-end speaker-attributed ASR," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 8082–8086 (cited on pp. 1, 110, 111).

[20] S. Watanabe et al., "CHiME-6 challenge: Tackling multispeaker speech recognition for unsegmented recordings," in *International Workshop on Speech Processing in Everyday Environments (CHiME)*, 2020 (cited on pp. 2, 8, 9, 14, 29, 38, 40).

[21] L. Drude, J. Heitkaemper, C. Boeddeker, and R. Haeb-Umbach, "SMS-WSJ: Database, performance measures, and baseline recipe for multi-channel source separation and recognition," 2019. arXiv: 1910.13934 [cs.SD] (cited on pp. 6, 97).

[22] J. Heymann, L. Drude, and R. Haeb-Umbach, "A generic neural acoustic beamforming architecture for robust multi-channel speech processing," *Computer Speech & Language*, 2017 (cited on p. 6).

[23] C. Boeddecker, J. Heitkaemper, J. Schmalenstroeer, L. Drude, J. Heymann, and R. Haeb-Umbach, "Front-end processing for the CHiME-5 dinner party scenario," in *International Workshop on Speech Processing in Everyday Environments (CHiME)*, 2018, pp. 35–40 (cited on pp. 8, 61).

[24] X. Fang et al., "A deep analysis of speech separation guided diarization under realistic conditions," in *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2021, pp. 667–671 (cited on p. 8).

[25] T. Menne, I. Sklyar, R. Schlüter, and H. Ney, "Analysis of deep clustering as preprocessing for automatic speech recognition of sparsely overlapping speech," in *Interspeech*, 2019, pp. 2638–2642 (cited on p. 8).

[26] M. Delcroix, K. Zmolikova, K. Kinoshita, A. Ogawa, and T. Nakatani, "Single channel target speaker extraction and recognition with speaker beam," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 5554–5558 (cited on pp. 8, 9).

[27] R. Wan et al., "The USTC-NERCSLIP systems for CHiME-7 challenge," in *International Workshop on Speech Processing in Everyday Environments (CHiME)*, 2023, pp. 13–18 (cited on pp. 8, 9, 61).

[28] L. Ye, H. Lu, G. Cheng, Y. Chen, Z. Shang, and X. Li, "The IACAS-Thinkit system for CHiME-7 challenge," in *International Workshop on Speech Processing in Everyday Environments (CHiME)*, 2023, pp. 23–26 (cited on p. 8).

[29] T. J. Park et al., "The CHiME-7 challenge: System description and performance of NeMo team's DASR system," in *International Workshop on Speech Processing in Everyday Environments (CHiME)*, 2023 (cited on p. 8).

[30] B. Mu et al., "The NPU system for DASR task of CHiME-7 challenge," in *International Workshop on Speech Processing in Everyday Environments (CHiME)*, 2023, pp. 63–66 (cited on p. 8).

[31] M. Karafiat et al., "BUT CHiME-7 system description," in *International Workshop on Speech Processing in Everyday Environments (CHiME)*, 2023, pp. 67–72 (cited on p. 8).

[32] K. Žmolíková et al., "Speakerbeam: Speaker aware neural network for target speaker extraction in speech mixtures," *IEEE Journal of Selected Topics in Signal Processing (JSTSP)*, vol. 13, no. 4, pp. 800–814, 2019 (cited on p. 8).

[33] Q. Wang et al., "VoiceFilter: Targeted voice separation by speaker-conditioned spectrogram masking," in *Interspeech*, 2019, pp. 2728–2732 (cited on p. 8).

[34] T. Yoshioka, H. Erdogan, Z. Chen, X. Xiao, and F. Alleva, "Recognizing overlapped speech in meetings: A multichannel separation approach using neural networks.," in *Interspeech*, 2018, pp. 3038–3042 (cited on pp. 9, 17, 18).

[35] M. He, D. Raj, Z. Huang, J. Du, Z. Chen, and S. Watanabe, "Target-speaker voice activity detection with improved i-vector estimation for unknown number of speaker," in *Interspeech*, 2021, pp. 3555–3559 (cited on pp. 9, 61, 110).

[36] C. Boeddeker, A. S. Subramanian, G. Wichern, R. Haeb-Umbach, and J. Le Roux, "TS-SEP: Joint diarization and separation conditioned on estimated speaker embeddings," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 32, pp. 1185–1197, 2024 (cited on pp. 9, 30, 35, 36, 46, 54, 59, 79, 107, 108, 110, 111).

[37] N. Kanda et al., "Streaming multi-talker ASR with token-level serialized output training," 2022, pp. 3774–3778 (cited on pp. 10, 43, 107, 108, 112).

[38] I. Sklyar, A. Piunova, X. Zheng, and Y. Liu, "Multi-turn RNN-T for streaming recognition of multi-party speech," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 8402–8406 (cited on pp. 10, 29, 30, 42, 53, 107, 112).

[39] A. Bronkhorst, "The cocktail party phenomenon: A review of research on speech intelligibility in multiple-talker conditions," *Acta Acustica united with Acustica*, vol. 86, pp. 117–128, Jan. 2000 (cited on p. 10).

[40] S. Haykin and Z. Chen, "The cocktail party problem," *Neural Computation*, vol. 17, no. 9, pp. 1875–1902, 2005 (cited on p. 10).

[41] W. Zhang et al., "Separating long-form speech with group-wise permutation invariant training," in *Interspeech*, 2022, pp. 5383–5387 (cited on pp. 12, 18, 100).

[42] J. Garofalo, D. Graff, D. Paul, and D. Pallett, "Csr-i (wsj0) complete," *Linguistic Data Consortium*, 2007 (cited on pp. 12, 33).

[43] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 5206–5210 (cited on pp. 12, 13, 33).

[44] Z.-Q. Wang, S. Cornell, S. Choi, Y. Lee, B.-Y. Kim, and S. Watanabe, "Tf-gridnet: Integrating full- and sub-band modeling for speech separation," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 31, pp. 3221–3236, 2023 (cited on pp. 13, 117).

[45] S. Choi et al., "An empirical study of training mixture generation strategies on speech separation: Dynamic mixing and augmentation," in *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2022, pp. 1070–1075 (cited on p. 14).

[46] M. Van Segbroeck et al., "Dipco — dinner party corpus," in *Interspeech*, 2020, pp. 434–436 (cited on p. 14).

[47] I. McCowan et al., "The AMI meeting corpus," in *Proceedings of the 5th International Conference on Methods and Techniques in Behavioral Research*, vol. 88, 2005, p. 100 (cited on pp. 14, 114).

[48] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, vol. abs/1412.6980, 2015 (cited on pp. 15, 74, 116).

[49] Y. Liu, M. Delfarah, and D. Wang, "Deep casa for talker-independent monaural speech separation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 6354–6358 (cited on p. 15).

[50] M. Kolbaek, D. Yu, Z.-H. Tan, and J. Jensen, "Multitalker speech separation with utterance-level permutation invariant training of deep recurrent neural networks.," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 10, pp. 1901–1913, 2017 (cited on pp. 15, 69).

[51] G.-P. Yang, S.-L. Wu, Y.-W. Mao, H.-y. Lee, and L.-s. Lee, "Interrupted and cascaded permutation invariant training for speech separation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2020, pp. 6369–6373 (cited on p. 16).

[52] H. Taherian and D. Wang, "Time-domain loss modulation based on overlap ratio for monaural conversational speaker separation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 5744–5748 (cited on p. 18).

[53] X. Chang, N. Kanda, Y. Gaur, X. Wang, Z. Meng, and T. Yoshioka, "Hypothesis stitcher for end-to-end speaker-attributed ASR on long-form multi-talker recordings," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 6763–6767 (cited on p. 18).

[54] N. Dehak, P. J. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-end factor analysis for speaker verification," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 4, pp. 788–798, 2011 (cited on p. 19).

[55] E. Variani, X. Lei, E. McDermott, I. L. Moreno, and J. Gonzalez-Dominguez, "Deep neural networks for small footprint text-dependent speaker verification," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 4052–4056 (cited on p. 19).

[56] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, vol. 5, University of California press, 1967, pp. 281–298 (cited on p. 19).

[57] A. Ng, M. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in Neural information Processing Systems (NeurIPS)*, vol. 14, MIT Press, 2001 (cited on p. 19).

[58] F. Murtagh and P. Contreras, "Algorithms for hierarchical clustering: An overview," *WIREs Data Mining and Knowledge Discovery*, vol. 2, no. 1, pp. 86–97, 2012 (cited on p. 19).

[59] S. Wisdom et al., "What's all the fuss about free universal sound separation data?" In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 186–190 (cited on pp. 21, 72, 78).

[60] C. Févotte, R. Gribonval, and E. Vincent, "Bss_eval toolbox user guide – revision 2.0," 2005 (cited on p. 22).

[61] E. Vincent, R. Gribonval, and C. Févotte, "Performance measurement in blind audio source separation," vol. 14, no. 4, pp. 1462–1469, 2006 (cited on p. 22).

[62] 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)*, 2019, pp. 626–630 (cited on pp. 22, 75).

[63] J. G. Fiscus, N. Radde, J. S. Garofolo, A. Le, J. Ajot, and C. Laprun, "The rich transcription 2005 spring meeting recognition evaluation," in *Machine Learning for Multimodal Interaction*, S. Renals and S. Bengio, Eds., Springer Berlin Heidelberg, 2006, pp. 369–389 (cited on pp. 23, 38).

[64] NIST, *The 2009 (rt-09) rich transcription meeting recognition evaluation plan*, 2009. [Online]. Available: https://web.archive.org/web/20100606092041if%5C_/http://www.itl.nist.gov/iad/mig/tests/rt/2009/docs/rt09-meeting-eval-plan-v2.pdf (cited on pp. 23, 29, 38, 47).

[65] N. Ryant et al., "First DIHARD challenge evaluation plan," 2018 (cited on p. 24).

[66] S. Baghel et al., "The DISPLACE challenge 2023 - diarization of speaker and language in conversational environments," in *Interspeech*, 2023, pp. 3562–3566 (cited on p. 24).

[67] V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Proceedings of the Soviet physics doklady*, 1966 (cited on p. 24).

[68] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem.," *Journal of the ACM*, vol. 21, no. 1, pp. 168–173, 1974 (cited on p. 25).

[69] J. G. Fiscus, J. Ajot, N. Radde, and C. Laprun, "Multiple dimension levenshtein edit distance calculations for evaluating automatic speech recognition systems during simultaneous speech," in *International Conference on Language Resources and Evaluation (LREC)*, Genoa, Italy: European Language Resources Association (ELRA), 2006, p. 6 (cited on p. 26).

[70] A. Backurs and P. Indyk, "Edit distance cannot be computed in strongly subquadratic time (unless SETH is false)," in *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing (STOC)*, New York, NY, USA: Association for Computing Machinery, 2015, pp. 51–58 (cited on p. 27).

[71] G. Myers, "A fast bit-vector algorithm for approximate string matching based on dynamic programming.," in *Annual Symposium on Combinatorial Pattern Matching (CPM)*, 1998 (cited on pp. 27, 52).

[72] E. Ukkonen, "Algorithms for approximate string matching," *Information and Control*, vol. 64, no. 1-3, pp. 100–118, 1985 (cited on p. 27).

[73] H. Berghel and D. Roach, "An extension of ukkonen's enhanced dynamic programming ASM algorithm," *ACM Transactions on Information Systems*, vol. 14, no. 1, pp. 94–106, 1996 (cited on p. 27).

[74] X. Huang, A. Acero, and H.-W. Hon, *Spoken language processing: a guide to theory, algorithm, and system development*. Prentice hall PTR, 2001 (cited on p. 29).

[75] M. Ravanelli et al., "SpeechBrain: A general-purpose speech toolkit," 2021. arXiv: 2106.04624 [eess.AS] (cited on p. 29).

[76] S. Young and L. Chase, "Speech recognition evaluation: A review of the U.S. CSR and LVCSR programmes," *Computer Speech & Language*, vol. 12, no. 4, pp. 263–279, 1998 (cited on pp. 29, 39).

[77] D. Raj, L. Lu, Z. Chen, Y. Gaur, and J. Li, "Continuous streaming multi-talker ASR with dual-path transducers," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 7317–7321 (cited on pp. 30, 42, 53).

[78] K. Hu et al., "Word level timestamp generation for automatic speech recognition and translation," in *Interspeech*, 2025, pp. 2565–2569 (cited on p. 33).

[79] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," in *International Conference on Machine Learning (ICML)*, 2022 (cited on pp. 36, 106).

[80] L. Brandschain, D. Graff, C. Cieri, K. Walker, C. Caruso, and A. Neely, "The mixer 6 corpus: Resources for cross-channel and text independent speaker recognition," in *International Conference on Language Resources and Evaluation (LREC)*, 2010 (cited on p. 38).

[81] A. Mansfield, "On the computational complexity of a merge recognition problem," *Discrete Applied Mathematics*, vol. 5, no. 1, pp. 119–122, 1983 (cited on pp. 44, 53, 54).

[82] S. Buss and M. Soltys, "Unshuffling a square is NP-hard," *Journal of Computer and System Sciences*, vol. 80, no. 4, pp. 766–776, 2014 (cited on p. 44).

[83] H. W. Kuhn, "The hungarian method for the assignment problem," in *50 Years of Integer Programming 1958-2008* (cited on pp. 49, 83, 89).

[84] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, 1957 (cited on pp. 49, 83).

[85] J. B. Kruskal, "An overview of sequence comparison: Time warps, string edits, and macromolecules," *SIAM Review*, vol. 25, no. 2, pp. 201–237, 1983 (cited on p. 59).

[86] A. Polok et al., "DiCoW: Diarization-conditioned whisper for target speaker automatic speech recognition," *Computer Speech & Language*, vol. 95, p. 101 841, 2026 (cited on pp. 68, 111).

[87] Y. Luo and N. Mesgarani, "Separating varying numbers of sources with auxiliary autoencoding loss," in *Interspeech*, 2020, pp. 2622–2626 (cited on pp. 71, 72, 78).

[88] S. Wisdom, E. Tzinis, H. Erdogan, R. J. Weiss, K. Wilson, and J. R. Hershey, "Unsupervised speech separation using mixtures of mixtures," in *Advances in Neural information Processing Systems (NeurIPS)*, vol. 33, Curran Associates, Inc., 2020, pp. 3846–3857 (cited on pp. 71, 78).

[89] J. Heitkaemper, D. Jakobeit, C. Boeddeker, L. Drude, and R. Haeb-Umbach, "Demystifying TasNet: A dissecting approach," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 6359–6363 (cited on pp. 71, 74, 78).

[90] H. Kim and J. W. Shin, "On training speech separation models with various numbers of speakers," *IEEE Signal Processing Letters (SPL)*, vol. 30, pp. 1202–1206, 2023 (cited on p. 72).

[91] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning. lecture 6a: Overview of mini-batch gradient descent," Tech. Rep., 2013 (cited on p. 74).

[92] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *International Conference on Machine Learning (ICML)*, Montreal, Quebec, Canada: Association for Computing Machinery, 2009, pp. 41–48 (cited on p. 74).

[93] C. Ma, D. Li, and X. Jia, "Optimal scale-invariant signal-to-noise ratio and curriculum learning for monaural multi-speaker speech separation in noisy environment," in *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2020, pp. 711–715 (cited on p. 75).

[94] E. Harper et al., *NeMo: a toolkit for Conversational AI and Large Language Models*. [Online]. Available: https://github.com/NVIDIA/NeMo (cited on p. 78).

[95] S. Dovrat, E. Nachmani, and L. Wolf, "Many-speakers single channel speech separation with optimal permutation training.," in *Interspeech*, 2021, pp. 3890–3894 (cited on p. 83).

[96] K. Supowit, "Finding a maximum planar subset of a set of nets in a channel," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 6, no. 1, pp. 93–94, 1987 (cited on p. 89).

[97] E. Kubicka and A. J. Schwenk, "An introduction to chromatic sums," New York, NY, USA: Association for Computing Machinery, 1989, pp. 39–45 (cited on p. 89).

[98] S.-C. Chang and R. Shrock, "Weighted graph colorings," *Journal of Statistical Physics*, vol. 138, no. 1, pp. 496–542, 2010 (cited on p. 89).

[99] R. Berke and D. Mitsche, "Colorings at minimum cost," *Sixth Czech-Slovak International Symposium on Combinatorics, Graph Theory, Algorithms and Applications*, vol. 310, no. 3, pp. 561–569, 2010 (cited on p. 89).

[100] J. E. Hopcroft, R. Motwani, and J. D. Ullman, "Automata theory, languages, and computation," *International Edition*, vol. 24, no. 2, 2006 (cited on p. 90).

[101] F. Maffray, "On the coloration of perfect graphs," in *Recent Advances in Algorithms and Combinatorics*, B. A. Reed and C. L. Sales, Eds., New York, NY: Springer, 2003, pp. 65–84 (cited on p. 90).

[102] J. B. Allen and D. A. Berkley, "Image method for efficiently simulating small-room acoustics," *The Journal of the Acoustical Society of America*, vol. 65, no. 4, pp. 943–950, 1979 (cited on p. 97).

[103] T. Cord-Landwehr, C. Boeddeker, C. Zorilă, R. Doddipatla, and R. Haeb-Umbach, "Frame-wise and overlap-robust speaker embeddings for meeting diarization," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023 (cited on p. 106).

[104] A. Nagrani, J. S. Chung, W. Xie, and A. Zisserman, "Voxceleb: Large-scale speaker verification in the wild," *Computer Speech & Language*, vol. 60, Mar. 2020 (cited on p. 106).

[105] D. Snyder, G. Chen, and D. Povey, "Musan: A music, speech, and noise corpus," 2015. arXiv: 1510.08484 `[cs.SD]`. [Online]. Available: https://arxiv.org/abs/1510.08484 (cited on p. 106).

[106] D. Raj, Z. Huang, and S. Khudanpur, "Multi-class spectral clustering with overlaps for speaker diarization," in *IEEE Spoken Language Technology Workshop (SLT)*, 2021 (cited on p. 110).

[107] Z. Huang, M. Delcroix, L. P. Garcia, S. Watanabe, D. Raj, and S. Khudanpur, "Joint speaker diarization and speech recognition based on region proposal networks," *Computer Speech & Language*, vol. 72, no. C, 2022 (cited on p. 110).

[108] S.-T. Niu et al., "DCF-DS: Deep cascade fusion of diarization and separation for speech recognition under realistic single-channel conditions," *IEEE Transactions on Audio, Speech, and Language Processing*, 2025 (cited on pp. 110–112).

[109] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur, "X-vectors: Robust DNN embeddings for speaker recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 5329–5333 (cited on p. 110).

[110] C. Boeddeker, T. Cord-Landwehr, and R. Haeb-Umbach, "Once more diarization: Improving meeting transcription systems through segment-level speaker reassignment," in *Interspeech*, 2024, pp. 1615–1619 (cited on p. 111).

[111] M. Delcroix, K. Zmolikova, T. Ochiai, K. Kinoshita, and T. Nakatani, "Speaker activity driven neural speech extraction," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6099–6103, 2021 (cited on p. 111).

[112] H. Taherian and D. Wang, "Multi-channel conversational speaker separation via neural diarization," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 32, pp. 2467–2476, 2024 (cited on p. 111).

[113] H. Taherian, A. Pandey, D. Wong, B. Xu, and D. Wang, "Leveraging sound localization to improve continuous speaker separation," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 621–625, 2024 (cited on p. 111).

[114] T. Cord-Landwehr, C. Boeddeker, and R. Haeb-Umbach, "Simultaneous diarization and separation of meetings through the integration of statistical mixture models," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Hyderabad, India, 2025 (cited on p. 111).

[115] Q. Hu, T. Sun, X. Chen, X. Rong, and J. Lu, "Optimization of modular multi-speaker distant conversational speech recognition," *Computer Speech & Language*, vol. 95, p. 101 816, 2026 (cited on p. 111).

[116] J. Schmalenstroeer, T. Gburrek, and R. Haeb-Umbach, "Libriwasn: A data set for meeting separation, diarization, and recognition with asynchronous recording devices," pp. 86–90, 2023 (cited on p. 111).

[117] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997 (cited on p. 117).

[118] A. Li, C. Zheng, R. Peng, and X. Li, "On the importance of power compression and phase estimation in monaural speech dereverberation," *JASA Express Letters*, vol. 1, p. 014 802, 2021 (cited on p. 118).

[119] Y. Luo, Z. Chen, N. Mesgarani, and T. Yoshioka, "End-to-end microphone permutation and number invariant multi-channel speech separation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 6394–6398 (cited on p. 118).

[120] M. Farber, "Characterizations of strongly chordal graphs.," *Discrete Mathematics*, vol. 43, no. 2-3, pp. 173–189, 1983 (cited on p. 119).