Bachelor's Thesis Computing Science

Radboud University Nijmegen

Oracle Component Analysis on Pyannote's speaker-diarization-3.1 Pipeline

---

Analysing the effect of speaker overlap on the SOTA model using Oracle components

Author:
Mohamad Naseb Aljalab
s1071472

First supervisor/assessor:
Dr. David van Leeuwen

Second assessor:
Dr. Twan van Laarhoven

June 19, 2024

Abstract

Speaker diarization, the process of determining 'who spoke when' in audio streams, is crucial for various speech processing applications but remains challenged by overlapping speech. This thesis investigates the impact of speaker overlap on the performance of Pyannote's speaker-diarization-3.1 pipeline using an oracle component analysis approach.

This approach makes use of ideal (oracle) components to isolate the effects of different stages in the pipeline. Results indicate that the pipeline still faces considerable difficulty in dealing with speaker overlap as it contributes to roughly half of all the errors while making up only a tenth of the speech time. By labeling segments conservatively, the segmentation module is prone to missing (parts of) overlapping speech, but does not 'hallucinate' any speech consequently. The clustering module faced more difficulty with speaker overlap as two-thirds of the recorded confusion was caused by those areas of overlap. The study concludes that while the Pyannote pipeline effectively handles speaker diarization, further improvements in handling overlapping speech are necessary, particularly in the clustering component.

Contents

Chapter 1

Introduction

1.1 Motivation

Speaker diarization (SD), the task of determining 'who spoke when' given a stream of audio, plays a crucial role in various speech processing applications ranging from transcription of meetings and interviews to speaker identification in multimedia archives. More precisely, an SD system that receives a stream of audio is expected to return the time stamps corresponding to each speaker's utterances. Despite significant advancements in speaker diarization techniques, challenges remain persistent. As technology evolves, there is a growing need for more efficient and accurate speaker diarization methods.

Traditionally, SD systems have followed a pipeline structure comprising four main modules or steps: speech activity detection, speaker change detection, speech turn clustering, and supervised speaker recognition. These cluster-based systems have demonstrated considerable success in various SD challenges. However, inherent limitations exist within this framework. Notably, these systems are inherently ill-equipped to handle speaker overlap [4], and errors introduced in one stage can propagate through subsequent steps.

In response to the shortcomings of traditional methods, the End-to-End Neural Diarization (EEND) paradigm has emerged as a promising alternative. Unlike the modular approach, EEND comprises a unified neural network responsible for all stages, from data intake to SD output. This conceptualization transforms the speaker diarization problem into a multi-label classification task within the EEND framework [8].

Despite its advancements, EEND encounters an array of challenges. Large amounts of data are required to train an EEND system, and running the system tends to be quite computationally expensive. Furthermore, EEND seems to be prone to underestimating the number of speakers (especially when the test set has more speakers than the train set), and it is prone to performing less well on long conversations (due to the inter-

nal self-attention mechanism) [21]. This led to the introduction of best-of-both-worlds (BoBW) implementations [16] that have been shown to perform better than either system alone.

1.2 Contributions

This thesis provides a setup for performing an oracle components analysis on a BoBW SD system along with some experiments and their results. Similar work, dating 2010 [13], has been done on SD systems, but mostly on traditional systems that existed before the introduction of EEND or BoBW approaches. The aim of this work is to understand the extent that speaker overlap plays in an SD system that has already been developed with overlap as a challenge in mind. The oracle experiments help achieve this by discovering how certain parts of the system perform under *perfect* conditions, while also extracting further insight from their behavior.

1.3 Outline

In chapter 2, the diarization error rate is broken down with a brief explanation to give the reader a sense of what the errors depicted in this work are referring to. An overview of the AMI dataset is also given since any work done in this thesis is relevant, especially to this dataset. After that, chapter 3 covers some of the related works showcasing how other oracle analysis work is done and what can be extracted from them. Chapter 4 explains the architecture of traditional SD systems while later showing the need for a different approach by way of discussing the shortcomings. This leads to the fifth chapter which introduces End-to-End Neural Diarization which was proposed in 2019 as well as the improvements made to the approach since then, including the development of a BoBW framework. Before beginning with the analysis that this work revolves around, the Pyannote pipeline is discussed and deconstructed in chapter 6. This not only informs the reader of the inner workings of the pipeline used but also provides researchers replicating this work a reference of the sort of system compatible with this analysis. Next, chapter 7 details the setup for the analysis where the reader can learn how every oracle component was built. A baseline for the experiments performed is then given to function as a reference point. Finally, the results of each experiment are discussed in order of the components.

Chapter 2

Preliminaries

This chapter describes some preliminaries that the reader must be aware of to understand the work done in this thesis. First, the diarization error rate (DER), and the Augmented Multiparty Interaction (AMI) corpus. Results in this work are framed using the DER, and the components that make it up. Without knowing these concepts, the results do not mean much more than some list of percentages. The AMI corpus is the sole dataset used in this work, which means that the corpus must be kept in mind when also thinking about the results. This chapter helps the reader become more aware of what the results that follow signify and where they come from.

2.1 DER

Evaluating the performance of diarization systems is essential to quantify and understand their effectiveness as well as to improve their accuracy. Currently, one of the most widely used metrics is the Diarization Error Rate (DER). DER is a comprehensive metric that measures the mismatch between the system's output and the reference transcription in terms of speaker labels and temporal boundaries. DER is defined as the sum of three types of errors:

1. Missed Speech[1] (MS) represents segments of speech that were present in the reference transcription but were not detected by the system. It is calculated as the duration of speech in the reference that is not covered by any speech segment in the system output. This also includes segments where a single speaker is hypothesized and mapped to the reference, but another speaker in the same segment was not detected. The latter speaker segment is then considered missed speech

2. False Alarm (FA) refers to segments that the system incorrectly marked as speech when there was none in the reference. It is calculated as the

[1]Missed Speech is sometimes also referred to as Missed Detection

duration of speech in the system output that does not overlap with any speech segment in the reference.

3. Speaker Confusion accounts for segments where the speaker label assigned by the system does not match the speaker in the reference. It is calculated as the duration of speech segments where the speaker label in the system output does not match the reference, excluding segments already accounted for by MS and FA

Mathematically, DER can be expressed as:

$$\text{DER} = \frac{\text{Missed Speech} + \text{False Alarm} + \text{Confusion}}{\text{Total Speech Duration}}$$

where each component is measured in terms of duration, typically in seconds, and the total speech duration is the duration of speech from each speaker.

2.2 AMI

The Augmented Multiparty Interaction (AMI) corpus - used in this research - is a multimodal data set with over 100 hours of meeting recordings. The corpus was created as part of a project that aimed to develop meeting browsing technology but was publicly released later on. The meetings took place in four different locations (Idiap, Edinburgh, TNO, and Brno) and were divided into scenario-based and naturally occurring meetings. Four distinct speakers participate in each meeting.[2].

An official fork of the AMI diarization setup, proposed in [18], is provided with the pyannote toolkit. This fork is split into the two sets *AMI* and *AMI-SDM*. The former includes multiple microphones in each recording while the latter is made up of only single distant microphone (SDM) audio setups. This work makes use of the AMI dataset, in which Beamforming [1] was used to obtain a single signal from multiple microphones.

The AMI set contains manual annotations in which all words are considered to be speech but is separated into two versions named 'only_words', and 'word_and_vocalsounds'[3]. The dataset is further divided into three disjoint sets: train, test, and dev. The train set was used by models in pyannote-audio as part of their training data and it includes over 80 annotated hours, of which around 66 are speech. The test set was used to obtain metrics in this research, and it is made up of 9 annotated hours, of which around 7 are speech. The dev set is of a similar composition to the test set, and it can be used for fine-tuning or validating models.

[2]This information is all encoded into the titles of the meetings as explained on https://groups.inf.ed.ac.uk/ami/corpus/meetingids.shtml.

[3]As the 'word_and_vocalsounds' references contain some inconsistencies in what annotators marked as vocal sounds and not, this work is focused on the 'only_words' version. Therefore, all mentions of AMI references in this thesis refer to the 'only_words' version.

Chapter 3

Related Work

The aim of an oracle analysis is generally to evaluate the potential performance of individual components within a system under ideal conditions. Several studies have applied this approach to speaker diarization systems to understand performance limits and identify bottlenecks.

In their paper 'The Blame Game: Performance Analysis of Speaker Diarization System Components', Marijn Huijbregts and Chuck Wooters analyze the performance of a speaker diarization system through a series of oracle experiments, which isolate the impact of individual components [14]. The aim was to understand the performance of each component in a system tested on a dataset of twelve conference meetings used in previous NIST benchmarks. Their findings highlighted that errors in speech activity detection (SAD) component had the most substantial impact on overall system performance. The authors also emphasize that one cannot assume every component in an SD system to be completely independent of the others. It is likely the case that a change from one component will lead to influence on others. Furthermore, the results of their analysis will be partially biased with respect to the evaluation data used. Regardless, the results will give a better understanding of the system at hand as well as an overview of what could be improved.

Another paper by Marijn Huijbregts and David van Leeuwen has conducted an in-depth error analysis of speaker diarization systems using oracle components [13]. Titled 'Speaker Diarization Error Analysis Using Oracle Components', the paper found that errors in (the tested) speaker diarization systems are predominantly due to weaknesses in the SAD component, the presence of overlapping speech, and the clustering component of the system[1]. The authors also highlight that in an oracle component analysis, the task of developing the oracle's to be used is a quite elaborate one.

Since then, many researchers have been making use of oracle components

[1]More specifically, it is the 'robustness of the merging component to cluster impurity' that leads to error.

in development to create better SD systems [20, 23, 11]. Most of the work on oracle component analysis done so far has been focused on cluster-based systems, which can be regarded as the previous SOTA. Thus, this work aims to perform this analysis on the current SOTA system which implements a BoBW approach.

Chapter 4

Clustering-Based Speaker Diarization

Before the introduction of end-to-end neural diarization systems, clustering-based approaches had dominated the field. These methods involve several stages, including speech activity detection (SAD), speaker change detection (SCD), feature extraction, and clustering. This chapter provides an overview of each of the individual components of these systems as well as discusses some shortcomings of the approach.

4.1 Speech Activity Detection (SAD)

SAD is the initial step in the diarization pipeline, distinguishing between speech and non-speech segments. Early SAD methods were based on simple thresholding techniques, but these have been largely replaced by more advanced machine learning approaches, such as Gaussian Mixture Models (GMMs) or deep neural networks (DNNs) [10]. These models are trained to recognize speech patterns more accurately, even in relatively noisy environments.

4.2 Speaker Change Detection (SCD)

SCD is crucial for identifying the exact points where speaker changes (or turns) occur. Traditional approaches, like the Bayesian Information Criterion (BIC) method [5], involve statistical modeling of the audio signal to detect changes in the speaker. More recent methods leverage supervised learning algorithms, including support vector machines (SVMs) [7] and neural networks, which have shown significant improvements in accurately detecting speaker changes.

4.3 Feature Extraction

Feature extraction converts speech segments into numerical representations that encapsulate speaker-specific information. MFCCs were among the first features used in speaker diarization due to their ability to model the human auditory system. However, the emergence of more sophisticated techniques, such as i-vectors and x-vectors, has significantly enhanced the performance of these features. While MFCCs are applied per frame, i- and x-vector are applied per segment. I-vectors provide a compact representation of speaker characteristics, while x-vectors, derived from deep neural networks, offer even higher accuracy and robustness.

4.4 Clustering

The final step in the diarization pipeline is clustering, where feature vectors are grouped to form clusters, each representing a unique speaker. Agglomerative Hierarchical Clustering (AHC) is a widely used approach where the most similar clusters are iteratively merged. The choice of distance metric, such as cosine similarity, plays a critical role in determining the effectiveness of the clustering process. Other clustering methods, such as k-means or spectral clustering, have also been explored, but AHC remains a popular choice due to its simplicity and effectiveness.

4.5 Shortcomings

Despite their success, clustering-based speaker diarization systems have notable limitations. Traditional clustering-based systems struggle with overlapping speech segments, where multiple speakers talk simultaneously. These systems, by design, typically assume that each segment contains only one speaker[1], which is very often incorrect in multi-speaker scenarios. Moreover, the modular nature of clustering-based diarization means that errors in one stage can propagate to subsequent stages. For instance, inaccuracies in SAD or SCD can lead to poor feature extraction and clustering, resulting in higher DER. Additionally, most clustering techniques are unsupervised, making it challenging to optimize the system directly for the diarization task. This often results in suboptimal performance, particularly in diverse and noisy environments.

[1]It is worth noting that a possible workaround to this is reserving certain clusters that would represent multiple speakers instead of a single one

Chapter 5

End-to-End Neural Diarization

5.1 Hitachi's Proposal

In 2019, scientists at Hitachi, Ltd. shared a paper titled *End-to-End Neural Speaker Diarization with Permutation-Free Objectives* [8] that introduced a new approach to speaker diarization. Instead of the usual system consisting of several separate modules, the team proposed using a single neural network. This novel end-to-end neural network (NN) would be able to ingest a stream of audio, and then directly output the diarization results. Furthermore, the speaker diarization problem becomes framed as a multi-label classification problem under this EEND approach. To briefly clarify this formulation, the speaker identities are viewed as labels, and the goal of the SD system is to assign the correct labels to each speech frame. This new framework is motivated by presenting the two major shortcomings of traditional cluster-based systems that have been addressed by EEND:

1. By design, cluster-based systems can only match a segment with at most one speaker, leading to the (false) assumption that two (or more) speakers cannot be speaking at the same time.

2. Since clustering is typically an unsupervised algorithm, the system cannot directly be optimized to minimize the DER.

By training on overlapping speech, it is claimed that the model then becomes better suited to handle overlapping speech in general. Furthermore, since the problem is multi-label classification, the system is not restricted to selecting a single label during inference and can point out different active speakers simultaneously.

5.2 Permutation Invariant Training

The permutation problem arises when an SD system tries to consistently assign output nodes to specific speakers across different time frames. The EEND framework addresses this issue using permutation invariant training (PIT). PIT is designed to handle the permutation problem by considering all possible permutations of speaker labels during training, and does so in the following manner:

- During training, for each time frame, the EEND model generates output probabilities for each potential speaker. Since the speakers' order is unknown and can vary, PIT evaluates all possible permutations of the output nodes (speaker labels).
- The training loss is computed for each permutation of the predicted speaker probabilities and the ground truth labels. PIT then selects the permutation that results in the minimum loss.
- The model is optimized based on this minimum loss, ensuring that it learns to produce consistent outputs regardless of the order in which speakers appear.

5.3 Self-Attention (SA-EEND)

While the EEND system proposed by Hitachi initially used *Bidirectional Long Short-Term Memory* (BLSTM) networks, BLSTM-based EEND could still benefit from certain enhancements in handling the dynamic and overlapping nature of speaker diarization tasks.

In 'End-to-end neural speaker diarization with self-attention', EEND is extended by integrating Self-Attention mechanisms, replacing the traditional BLSTM layers [9]. This enhancement leverages the ability of self-attention to directly condition on all frames within an input sequence. This is as opposed to BLSTMs, which are limited to their immediate previous and next hidden states. This characteristic of self-attention allows for a more comprehensive analysis of the temporal dependencies and relationships across the entire audio sequence, which shows to be particularly advantageous for identifying and distinguishing between speakers in scenarios with frequent overlaps.

5.4 Encoder-Decoder based Attractor Calculation

In addressing the limitations of the SA-EEND method, particularly its inability to dynamically adapt to varying numbers of speakers, another approach called Encoder-Decoder based Attractor Calculation (EDA) has been

proposed [12]. This extension builds on the original SA-EEND by allowing it to manage diarizations for an unknown and theoretically unlimited number of speakers.

The EDA approach introduces a mechanism to generate a flexible number of attractors from a speech embedding sequence, which is then used to estimate speaker activities dynamically. The attractors are specific points in a high-dimensional space used to represent individual speakers. Like the embeddings, attractors serve as references that help the system distinguish between different speakers within an audio recording. This method leverages the ability of the existing SA-EEND system to extract speech embeddings but extends it through an encoder-decoder structure that predicts the presence and characteristics of each speaker in the audio without prior knowledge of speaker count. The EDA approach is as follows:

1. An LSTM-based encoder-decoder architecture calculates a series of attractors from the input speech embeddings. These attractors represent distinct speaker characteristics extracted throughout the audio sequence.

2. The number of attractors (and thus the number of detected speakers) is determined on the fly using a sigmoid activation function that assesses the probability of each attractor's presence. This process allows the model to adapt to the actual number of speakers in the recording[1].

3. The final diarization output is computed by assessing the relationship between the attractors and the full set of embeddings using dot product operations. This facilitates an accurate allocation of speech segments to individual speakers.

5.5 BoBW

As described in 4.5, clustering-based SD systems face several shortcomings, but this is also true for EEND. The challenges faced by EEND systems include:

1. EEND systems struggle to predict the true number of speakers [6]

2. Training EEND systems requires very large amounts of data. This often leads to training on synthetic unrealistic data which further hinders performance [17].

3. As audio lengths grow, performance in EEND systems is progressively throttled (partially due to the self-attention mechanism).

[1] In the traditional (SA-EEND) framework, the number of sigmoid nodes limits the number of speakers. This is because the linear transformation applied to the embeddings outputs a fixed number of sigmoid activations, each corresponding to a potential speaker.

Thus, several publications have been made after the introduction of EEND suggesting an approach in which elements of clustering-based systems are adopted alongside EEND models [16][15]. These two methods have been shown to possibly be complementary to one another following diarization challenges such as in the third DIHARD diarization challenge [22]. This combination has been dubbed a best-of-both-worlds (BoBW) framework as it draws advantages from each approach [6]. Most importantly, a BoBW framework is more successful in handling overlapping speech due to its EEND component but does not struggle with long streams of audio due to the clustering component[2].

Although EEND was extended to handle a variable amount of speakers (see section 5.4), it was later experimentally shown to nonetheless struggle in dealing with meetings containing a realistically large amount of speakers (over 3). Improvements to the BoBW approach have been made more recently [6] (see section 6.1).

[2]The latter claim might not seem as clear at first as one would assume that the EEND model would still have to ingest the long streams of audio as well. However, the clustering component allows the EEND model to ingest smaller chunks of the long stream without suffering the inter-block label permutation problem. Furthermore, the clustering property is better able to predict the true number of (global) speakers. This is elaborated on further in 6.1

Chapter 6

Pyannote

Pyannote, pronounced like the French verb 'pianoter' [2], is an open-source toolkit for speaker diarization created by Hervé Bredin. This toolkit includes pre-trained pipelines, models, data structures, and metrics written in Python, allowing ease of use and portability. This chapter will break down the SOTA pyannote pipeline detailing the individual modules and their integration.

6.1 Pipeline

Pyannote provides some open-source pre-trained pipelines and models that can be found on `https://huggingface.co/pyannote`. At the time of writing, the (published) pyannote SOTA pipeline for speaker diarizations is the speaker-diarization-3.1 pipeline. The three main modules that make this pipeline up are segmentation, embedding, and clustering. This chapter describes each of these modules as well as how results are utilized in subsequent modules.

6.1.1 Segmentation

Speaker segmentation is described as the task of 'partitioning a conversation between one or more speakers into speaker turns'[1] [3]. This is the first phase of the speaker diarization pipeline, and it yields chunks of speech that contain one (or more) speaker each. Traditionally, this module would be separated into different steps. Speech activity detection (SAD) would first be responsible for differentiating between speaker and non-speaker segments, followed by speaker change detection (SCD) where boundaries between speaker segments of different speakers are marked.

[1] A speaker turn is a point in an audio stream where 'a change of speaker occurs'.

Sliding Windows

The segmentation model of pyannote adopts the EEND approach developed by [8], but alters some specifics of the model. Instead of handling long streams of audio at once, the model is trained on and operates in, windows of five seconds. More specifically, these are fixed-size sliding windows that move across the entire audio stream from start to end with a given step size. The window represents a sequence of binary frames that encode whether or not a speaker is active during some moment. To be more specific, the pyannote-core library (responsible for providing the various data structures for diarization) includes two structures: SlidingWindow and SlidingWindowFeature.

The SlidingWindow does not hold the binary frame sequences but only indicates how some sliding window should look like. For example, a SlidingWindow object can indicate that some sliding window moves with a duration of 0.01 seconds and a step of 0.01 seconds. The SlidingWindowFeature carries a SlidingWindow as well as the binary frame sequences that were generated using it. Traditionally, the binary frames are computed using a segmentation threshold that is passed through every speaker's predictions at test time to determine whether or not they were active. A static threshold suffices, but a more sophisticated dynamic threshold method could yield better results in practice.

Since the neural network no longer has to consider the entire audio stream at once, it is also able to handle a much smaller set of speakers. The modification is important not only for memory concerns but Kinoshita et al. note that the EEND neural network faces considerable difficulty when dealing with very long novel sequences of data [16]. With this challenge in mind, Bredin and Laurent define a $K_{\max}$ of four speakers per window [3]. This choice also follows the observation that 99% of every five-second chunk encountered during training contained less than four speakers. However, this sliding window approach introduces the inter-block label permutation problem [16]. This means that although speakers within each five-second window could be differentiated, it is not yet possible to infer the mapping of identities across windows[2].

Power-set Encoding

Speaker diarization is framed as a multi-label classification problem according to the original EEND proposal, but an alternative phrasing, describing it as a single-label power-set classification problem is later introduced [6]. In the power-set encoding, instead of **only** encoding the speakers, the **power-set** of speakers is encoded. For example, suppose that three speakers exist:

[2]Luckily, this pipeline does not end at segmentation, and the later modules will address this issue.

A, B, and C. Then, the multi-label encoding of these speakers would be [{A}, {B}, {C}], while the power-set encoding would be [{∅} {A}, {B}, {C}, {A&B}, {A&C}, {B&C}, {A&B&C}]. This encoding provides the ability to explicitly encode overlap in the segments as its own class, which leads to better handling of overlap in the system. However, this approach could also be more computationally demanding, as the number of encodings grows very large with additional speakers. Specifically, for every new speaker added to an encoding with $N > 1$ existing speakers, the set of classes doubles in size. Luckily, with the pyannote-defined K_{max} limiting the number of speakers in a single window to four, the power-set encoding would be limited to sixteen classes. As sixteen classes can still be more than one would prefer, an alternative K_{max} of three (which is used in practice) could be employed, as it would only result in eight power-set classes! Moreover, the segmentation threshold that was previously used would now get replaced by an *argmax* that would select the class with the highest prediction at test time.

6.1.2 Embedding

Following segmentation, the pipeline moves to an embedding module that converts the segments (or windows) it receives into numerical representations that capture the unique characteristics of each speaker's voice. The model currently used in the SOTA pipeline is the *wespeaker-voxceleb-resnet34-LM* model from the Wespeaker toolkit [24]. The underlying neural network (NN) here is the ResNet34 convolutional NN. The utility of embeddings is as follows: with a **good** set of embeddings, two embeddings should be as similar as possible if they represent the same speaker, and as different as possible if they represent different speakers[3]. This allows us to learn about speaker identities within, but more importantly across, windows.

6.1.3 Clustering

In the final module of the pipeline, the previously generated embeddings can now be clustered to demonstrate different (global) speaker identities. Pyannote employs an Agglomerative Clustering, which is a type of hierarchical clustering. Generally, the clustering begins with each embedding as its own cluster and computes distances between each pair of clusters. By default, a cosine similarity measure is used but other metrics are also supported. Next, the closest pair of clusters, determined by the previous metric, are merged into a single cluster. The linkage method, which determines how metrics should measure distances between clusters, can also be changed. For example, the linkage method could dictate that the metric must be applied on the centroids of the two clusters (average linkage), or the closest points between two clusters (single linkage). After merging, the distances are updated and

[3]This is also true for any embedding of multiple speakers.

the algorithm continues to search for another pair of clusters to merge. This is repeated until termination, which is determined by a given threshold. The threshold represents how far apart clusters must (at least) be for them to be deemed 'not similar'. A minimum cluster size that specifies the minimum number of embeddings that a cluster must contain is also taken into account. This parameter helps in filtering out smaller clusters that may not represent a distinct speaker effectively.

Chapter 7

Oracle Component Analysis

The SD pipeline dissected in section 6.1 was described as a sequence of modules, making it suitable for an oracle component analysis. An oracle is generally someone or something that can generate insight or truths. In speaker diarization, an oracle replaces or simulates part of the system in a *perfect* manner. This is typically done by 'cheating' and using information from the ground truth of a certain audio stream[1].

By coming up with enough oracles, an oracle component analysis could be performed, where certain parts of the system being analyzed are replaced with their oracle counterparts. This is such that some experiments can be executed to showcase the strengths and weaknesses of a given system. More specifically, the performance of an SD system where every module but one has been replaced by an oracle counterpart provides us insight into the specific module that hasn't been replaced.

The phase of coming up with oracle variants for each component can often require the most time in this analysis as the task is quite elaborate [13]. However, after tests have also been developed, they can be automated across the different oracle configurations. This chapter will introduce the experimental setup for the oracle component analysis followed by the actual experiments and their results.

7.1 Setup

To execute this analysis on the pyannote pipeline, an experimental setup must be established. This setup must replicate the original pipeline's behavior when it's under the same circumstances (hyper-parameters, data, etc..) but also allow for internal changes to the pipeline. To accomplish this, the scripts[2] responsible for the pipeline's execution are broken down

[1] From this, it naturally follows that an oracle component analysis requires the presence of ground truth annotations.

[2] Namely pipeline.py and speaker_diarization.py from the pyannote.audio library.

and copied into a single Jupyter notebook. This helps break down the abstractions of pyannote such that certain components of the pipeline can be swapped out for their oracle counterparts.

7.1.1 Oracle Segmentation

The original segmentation function applies the chosen segmentation model on the audio stream, resulting in a SlidingWindowFeature object which holds a numpy array with shape: (num_chunks, num_frames, num_speakers). On the other hand, the oracle counterpart applies a deterministic algorithm to generate oracle segments for the audio stream using its corresponding RTTM file. This also results in a SlidingWindowFeature object similar to the one generated with the original segmentation, except that this one accurately represents the audio with no error[3]. The algorithm takes the following steps:

1. Determine the audio duration, speaker labels, and number of speakers from the input RTTM
2. Initialize SlidingWindow with given step and duration
3. Iterate over audio in equally sized chunks defined by the SlidingWindow
4. For each chunk, discretize[4] the annotation into a SlidingWindowFeature based on the resolution defined by frames and add the specified labels. This represents one segmentation.
5. Stack the segments of all chunks into a single array and return it as a SlidingWindowFeature object.

7.1.2 Oracle Embedding

Although the results of an oracle embedding experiment would be trivial, it is still done as a sanity check to ensure that the regular agglomerative clustering is functioning properly.

Assuming that embeddings in a system are perfect implies that the segments generated prior were also perfect. Moreover, the clustering done subsequently should also have no trouble distinguishing the different speakers since the embeddings being clustered are as distinct as possible.

Generating these embeddings is rather straightforward. Given the true (oracle) segments, a one-hot encoding can be generated in the shape of

[3]Negligable errors, of at most 0.5%, might still occur due to system rounding errors.

[4]The discretize function takes an annotation and converts it into a binary feature matrix.

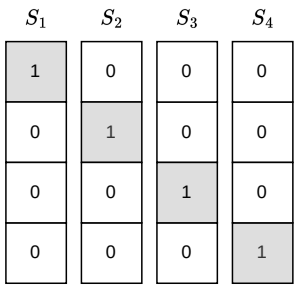

Figure 7.1: One-hot encoding of four speakers

embeddings[5]. The algorithm achieves these embeddings in the following manner:

1. Iterate over the segments counting the occurrences of each speaker in each chunk to create a list of speaker counts
2. Normalize the speaker counts by the number of frames to obtain the average embedding vector for each speaker
3. Use *torch.diag_embed* to transform the normalized counts into diagonal matrices.

Figure 7.1 provides an example of a one-hot encoding matrix for four speakers. This construction ensures that each embedding vector is unique, which allows the clustering to clearly identify each speaker.

7.1.3 Oracle Clustering

The original clustering taking place in the pipeline is specifically agglomerative. This module receives speaker embeddings of the segments created earlier in the pipeline along with some customizable hyper-parameters. In turn, it produces a list of 0-indexed cluster indices. The oracle counterpart for clustering has a slightly different I/O but accomplishes the same task. The method receives previously computed speaker segments, but also (optionally) receives speaker embeddings if centroids need to be returned. The result of computation is hard clusters, soft clusters, and (optionally) centroids. Hard cluster assignments indicate which cluster each speaker belongs to, while soft cluster assignments contain probabilities indicating the likelihood of a speaker belonging to each cluster. The algorithm takes the following steps:

[5]In pyannote, embeddings carry the shape (num_chunks, local_num_speakers, dimension).

Missed Detection	False Alarm	Confusion	DER
9.53%	3.57%	5.71%	18.80%

Table 7.1: DER breakdown for AMI test set

1. Extract the shape and SlidingWindow from the given segments
2. Compute oracle segments for the audio stream and extract the number of clusters needed from their shape
3. Align the number of frames in the input segments[6] and the oracle segments to the minimum of their respective lengths.
4. Initialize hard cluster assignments with -2 (indicating unassigned) and soft cluster assignments with zeros.
5. Iterate over each chunk and its corresponding segments while using a permutation function to align oracle segments with input segments and update all cluster assignments based on the permutation.
6. If embeddings are not provided, return the hard and soft clusters without centroids[7].

7.2 Baseline

Before going over the results of the oracle experiments, baseline results for the AMI test set are given in this section. Table 7.1 breaks down the average DER - computed between the ground truth annotations, and pipeline's output annotations - of the test set.

The DER of 18.80% matches the rate reported by pyannote for this pipeline. Furthermore, the missed detection dominating about half of the errors indicates that the pipeline fails to detect speech roughly 9.53% of the time. This error is caused in the segmentation module but is also likely caused by certain difficulties in the data. On the other hand, 3.57% of the total speech hypothesized was not present in the data, which could also be 'blamed' on the segmentation module. For 5.71% of the speech time, the clustering module assigned speaker labels to the wrong speaker.

Moreover, since speaker overlap is one of the major challenges that speaker diarization faces, and a focus of this research, some figures and experiments included in this thesis aim to give a clearer idea of how well this pipeline can handle its presence. One way of quantifying the amount of

[6]Note that these are regularly computed segments, but oracle segments can also be inputted. In that case, one would be simulating a *near perfect* system.

[7]The rest of the algorithm responsible for computing centroids will not be covered as it is not relevant enough.

Missed Detection	False Alarm	Confusion	DER
6.22%	0.00%	3.32%	9.55%

Table 7.2: DER breakdown for AMI test set on overlap segments only

difficulty that overlap imposes on this SD system would be to obtain a DER measure computed **only** over parts of the output that contain overlap. This can be done by first obtaining the difference in errors (MD, FA, confusion) between the DER computed over the whole speech and the DER computed without overlap. The latter can be done conveniently using a *skip_overlap* parameter. Next, after the difference in error has been summed, they are divided by the total speech time. This translates into percentages of error on segments that only contain speaker overlap.

According to table 7.2, about half of the errors made in the diarization of an audio stream are present in speaker overlap sections. These overlap sections also make up only 11% of the total speech time on average. No false alarm is detected in overlap sections, but about 65% of all missed speech occurs in these sections. As overlapping speech segments can be difficult to label, the segmentation model might behave more conservatively in labeling a segment as speech when facing overlap. This bias would explain an increase in missed detection and the inverse in false alarms. Finally, 58% of the total confusion is made in overlap sections. As this is only 3.32% of the whole speech time, it seems that the clustering in this BoBW system can handle overlap very well. Regardless, the system still faces some difficulty distinguishing between multiple speakers that happen to be speaking simultaneously, which leads to incorrect attribution.

To support the previous table, figure 7.2 plots each test file's DER against the amount of true overlap relative to their total length[8]. A trend line, computed using numpy's polyfit function, is also displayed in red, highlighting the upward trend in DER as the relative amount of overlap grows. According to the figure, audio with less than 12% overlap tends to face a DER of roughly 10-20%, while audio with 20-27% overlap faces a higher DER of roughly 23-28%.

This analysis can be taken a step deeper by breaking the figure down into three other individual figures. Namely, the rates of Missed Detection, False Alarm, and Confusion could be plotted against the amount of relative overlap. Each of these figures also contains their corresponding trend line. The upwards trends in A.3 and A.2, as well as the slightly downwards trend in A.1, can all be explained by the results of table 7.2.

[8]This overlap is true in respect to the ground truth annotations, as opposed to the hypothesized annotations. Picking the latter would indicate how much overlap the system can hypothesize and not how well it deals with existing overlap.

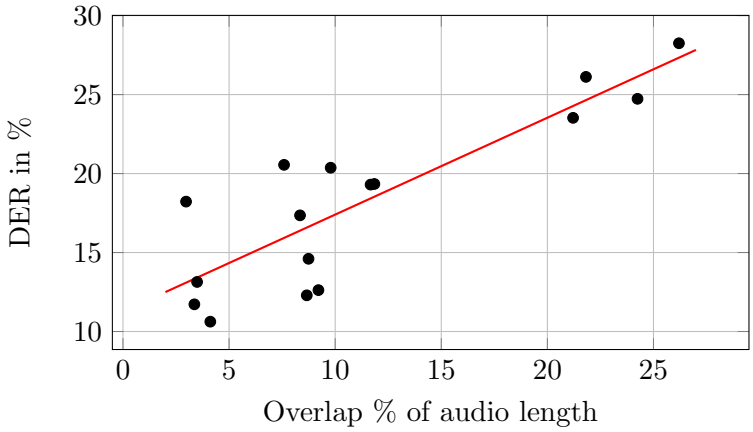

Figure 7.2: DER vs Relative Overlap amount

Missed Detection	False Alarm	Confusion	DER
0.01%	0.41%	8.59%	9.01%

Table 7.3: DER breakdown for AMI test set with Oracle Segmentation

7.3 Oracle Segmentation

This section explores how the pipeline performs when segments are replaced with oracle segments. First, the system is run on the 16 audio files that make up the AMI test set. Table 7.3 shows the results of this inference in terms of DER. Moreover, figure 7.3 plots the DER computed in this configuration against each audio stream's relative overlap amount[9]. In this figure, a strong upward trend line can be seen. The DER noted here translates fully to confusion, which reinforces the idea that clustering segments with overlapping speech is a difficult task.

7.3.1 False Alarm & Missed Detection

Although much lower than the baseline, the false alarm and missed detection remain just above 0. This could suggest that the oracle segments generated are not fully accurate with respect to the ground truth annotations. Recall from section 7.1.1 that every chunk in the oracle segments is discretized according to a specified resolution. The default resolution (≈ 0.01697) is defined by the segmentation model and seems to have a notable influence on the extracted segments.

As the resolution gets closer to 0, so do the false alarm and missed detection rates. The DER in this configuration is then made up entirely

[9]This DER was computed after the resolution increase described in section 7.3.1

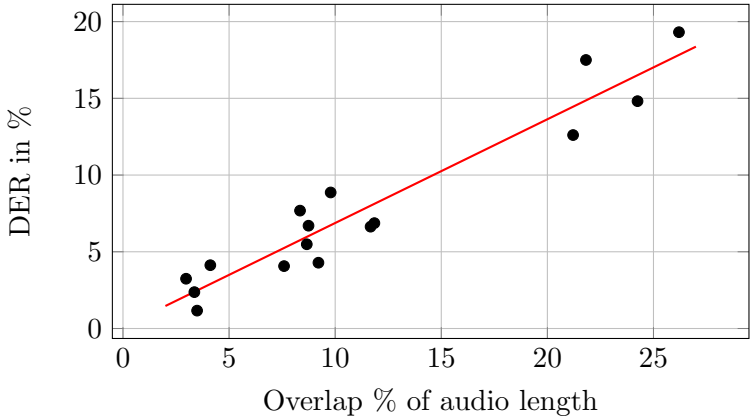

Figure 7.3: Oracle Segmentation DER vs Relative Overlap amount

of confusion, which remains almost identical to the one computed before adjusting the resolution. The remaining DER can be seen as an estimate of how accurately the embeddings (produced from the oracle segments) are clustered later in the pipeline. This amounts to 7.10% DER after optimizing the clustering hyper-parameters.

7.3.2 Confusion

When compared to the baseline confusion, it is worth noting that the confusion in this setup has increased. The already notable confusion rate indicates that the clustering done later in the pipeline is not able to perfectly distinguish between the different speakers.

Since the clustering hyper-parameters were not trained on oracle segments, the clustering is expected to perform worse than usual in this setup, which partially justifies the increase in confusion. When clustering is tuned on the dev set using oracle segments, a new confusion rate of 7.10% is reached[10]. In this fine-tuning, the *min_cluster_size* increased from 12 to 20, while the *threshold* decreased from 0.70 to 0.61.

Moreover, the embeddings in this configuration are still being generated using the Wespeaker model, which also hasn't been trained/tuned on any oracle segments (or even the AMI set in general, as opposed to the segmentation module). This is another factor that further explains the increased confusion (in comparison to the baseline) that is computed even after tuning the cluster parameters.

It is also worth noting that the clusters hypothesize anywhere between 6 and 22 speakers across the AMI test sets (compared to 3 and 9 in the oracle-

[10]This is an improvement from the previous 8.60% but still over the baseline of 5.71%. This is the result of 20 epochs, and confusion would likely decrease further with more tuning epochs.

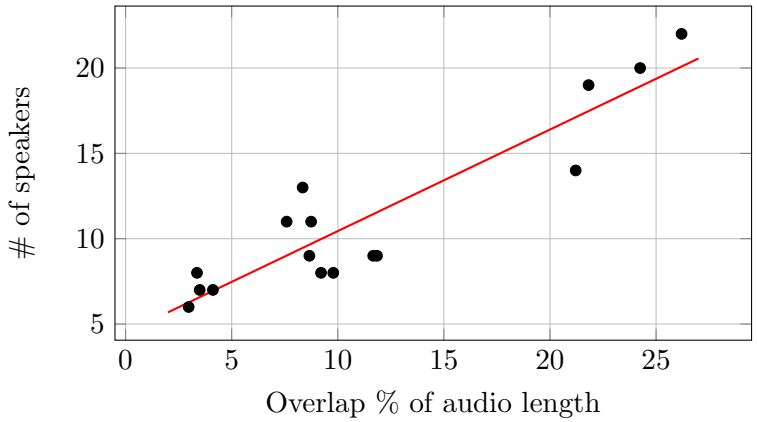

Figure 7.4: Hypothesized # of speakers vs Relative Overlap amount pre-tuning

free system), which is always above the true amount of speakers (4). The amount of different speakers is determined after clustering and derived from the number of detected clusters. These additional speakers are generally not very active but are always viewed as confusion nonetheless.

By increasing the minimum cluster size parameter, less active speaker clusters are ignored and the system thus produces fewer speakers in its hypothesis. Furthermore, by decreasing the clustering threshold, less active speakers can also get merged into the closest speaker if they are within the new threshold. The parameter adjustments specified earlier result in a more conservative range of detected speakers of 3 to 9. The difference made on the hypothesized amount of speakers against the relative amount of overlap before and after tuning the clustering parameters can be seen in figures 7.4 and 7.5 respectively. From these figures, it can be seen that optimizing the clustering parameters makes the system more resistant to overlap. The remainder of the confusion is then a general failure to correctly label some segments.

Closed Clustering

Since the only error occurring in this configuration is due to clustering, a closed clustering experiment was done to see whether is it possible to get better results if the amount of speakers is known beforehand. In the case of the AMI test set, each audio file contains exactly 4 speakers. By specifying the number of speakers before running inference, the clustering module knows exactly how many clusters it should end up with, and can potentially produce a better fit.

However, upon experimentation, it is found that this approach does not necessarily lead to better diarization results. In the case of confusion com-

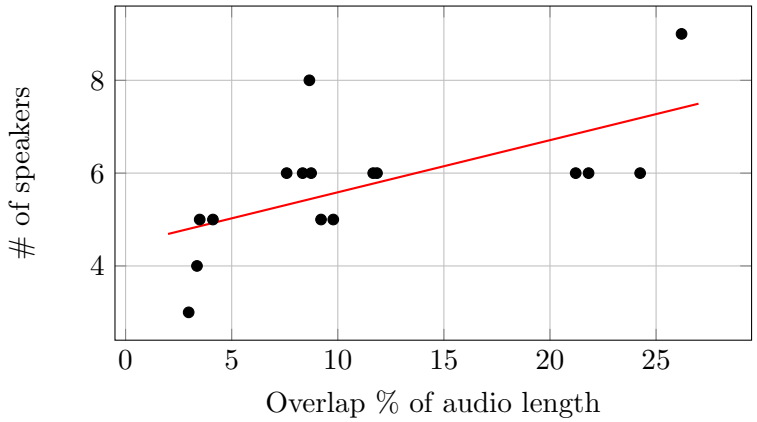

Figure 7.5: Hypothesized # of speakers vs Relative Overlap amount post-tuning

Missed Detection	False Alarm	Confusion	DER
0.00%	0.00%	6.40%	6.40%

Table 7.4: DER breakdown for AMI test set on overlap segments only with Oracle Segmentation

puted with oracle segmentation, an increase of 312% was noted when performing closed clustering. As discussed earlier in the section, this configuration tends to heavily overestimate the amount of speakers present in an audio stream. Under closed clustering, the module attempts to force the clusters to fit the pre-specified cluster amount. It does so by refining the threshold used until a fit is found. By doing so, the 'additional' speakers must eventually get merged into the closest cluster, which might be the wrong one according to the annotation. Although this is a worse result according to the DER, it must be noted that this metric could sometimes be biased in the way it computes errors.

Finally, table 7.4 reveals that sections of speaker overlap contribute 6.40% to the total confusion. That is roughly 75% of the total confusion caused by approx. 11% of speaker time overall, indicating a great struggle in accurately labeling speaker segments containing speaker results.

7.4 Oracle Embedding

Instead of using the regular embeddings module, this configuration uses oracle segments to produce ideal embeddings. As mentioned in section 7.1.2, if the clustering module is working adequately, it should provide a perfect diarization with 0% DER, as:

Missed Detection	False Alarm	Confusion	DER
9.53%	3.57%	2.90%	16.00%

Table 7.5: DER breakdown for AMI test set with Oracle Clustering

Missed Detection	False Alarm	Confusion	DER
6.22%	0.00%	1.32%	7.54%

Table 7.6: DER breakdown for AMI test set on overlap segments only with Oracle Clustering

- The missed detection & false alarm is at 0% as the segments were generated using the ground truth.
- The confusion would be at 0% as clustering fully disjoint and consistent embeddings is a trivial task.

After running the inference on the AMI test set, this is noted to be the case.

7.5 Oracle Clustering

In this section, experiments are done as the clustering module is replaced with its oracle counterpart. Recall that in this configuration, speaker embeddings are not necessary for generating a diarization, but they might be provided for computing centroids. In this setup, centroids are not needed.

The clustering method relies on ground truth segmentations to directly assign clusters as opposed to inferring labels from speaker embeddings. Inference was performed on the test set using this system as shown by the results in table 7.5. Table 7.6 displays the DER breakdown computed only over speaker overlap segments in this configuration. To aid the table in showcasing the struggle against overlap, figure 7.6 plots this configuration's DER against the relative overlap, where an upward trend is still clear.

7.5.1 Confusion

When compared to the baseline confusion, a reduction of roughly 50% can be seen, signifying a noticeable improvement. A reduced confusion rate is to be expected when considering that the clustering component can make use of the true speaker labels in this configuration. For overlap-only regions, confusion faces a reduction of roughly 40%.

7.5.2 False Alarm & Missed Detection

Both measures of false alarm and missed detection remain identical in this configuration in comparison to the baseline. Similar to the baseline, the

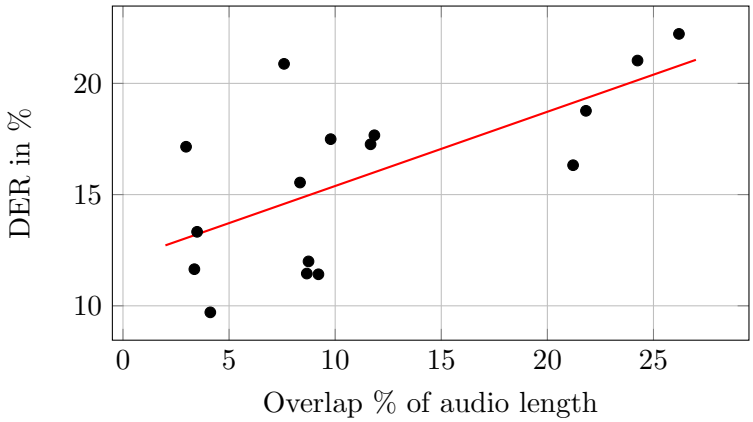

Figure 7.6: Oracle Clustering DER vs Relative Overlap amount

Missed Detection	False Alarm	Confusion	DER
0.00%	0.00%	0.00%	00.00%

Table 7.7: DER breakdown for AMI test set with both Oracle Clustering & Segmentation

segmentations generated here rely on the pyannote segmentation model. By only changing the clustering component, the system can start allocating different speaker labels to certain segments, but cannot alter the speaker segments. This leads to the hypothesis that this configuration would lead to the same FA and MD as the baseline, which can be seen in table 7.5. For the DER computed over the overlap-only sections, the same reasoning holds.

7.6 Fully-Oracle Pipeline

With the oracle segmentations and oracle clustering modules set up, the pipeline can also be executed using both at once. As the oracle clustering ignores embeddings, this constitutes a full *oracle system*. Any hypothesis produced from the ground truth should measure a 0% DER. This provides no further insights besides serving as a sanity check to prove that each oracle component is functioning properly and that the results generated using them are valid.

As shown by 7.7, the system is able to generate a hypothesis that exactly matches the reference!

7.7 Limitations

The oracle components analysis points out the weaknesses and strengths of the pipeline, but it also faces some limitations. Some of the assumptions that are made following the results of an experiment can often be biased by the data used (amongst other variables), and not deliver a complete picture. This would especially be the case in this study as the AMI dataset was amongst the datasets used to train the segmentation module. Ideally, the experiments are replicated with data from different settings, different speaking styles (accents/inflections/etc.), different speaker amounts, etc.

Moreover, the results of every experiment are framed using the DER framework, which might deliver a narrowed view of the system's performance. Although this metric is the most popular in the diarization realm, it tends to leave some oversight. For example, errors made in shorter segments can often be overlooked or 'drowned out' by the presence of longer segments. Errors concerning less active speakers can also get overlooked. This motivated Tao Liu, and Kai Yu to describe the Balanced Error Rate as a more even metric for diarization [19]. Other popular metrics besides the DER also include the Jaccard Error Rate (JER), and the Conversational Diarization Error Rate (CDER).

The embedding module of the pipeline had unfortunately contributed to bits of uncertainty in the experiments as well. For example, in the oracle segmentation setup, clustering is not done directly on the oracle segments, but on the embeddings of those segments. When an oracle embedding is also provided, clustering becomes trivial. Thus, the extent to which embeddings contribute to the DER is not exactly clear, which makes the computed error rate an upper bound for clustering performance rather than an accurate measurement. In addition, since embeddings are not needed when performing oracle clustering, they are only ignored. Therefore, the embedding module is not directly assessed by the oracle components approach. Luckily, the developers of the Wespeaker embeddings module reported the model's performance in their paper that introduces the Wespeaker toolkit. Under an oracle speech activity detection (SAD) setup, they report a DER of 4.2% [24][11].

[11] This DER is not directly comparable to the other error rates computed in this work as it was measured on a different dataset. Namely, the VoxConverse dev set.

Chapter 8

Conclusions

This thesis explored the effects of speaker overlap on the performance of Pyannote's speaker-diarization-3.1 pipeline through an oracle component analysis. The analysis confirmed that overlapping speech still poses significant challenges for speaker diarization systems. Segments with overlapping speech accounted for a disproportionate share of errors, highlighting the need for better handling of such scenarios.

Experiments using oracle segmentation demonstrated a substantial reduction in missed detections and false alarms, emphasizing the importance of accurate segmentation. However, even with perfect segmentation, the clustering component still struggled with overlap, suggesting that clustering improvements are critical for overall performance enhancement.

The use of oracle embeddings resulted in perfect clustering, indicating that the current embedding methods contribute notably to the overall error rate[1]. Oracle clustering reduced the DER by approximately 50%, primarily by reducing speaker confusion. These findings underscore the necessity of better embedding techniques and more sophisticated clustering algorithms.

The research highlighted certain limitations, including the potential biases introduced by using DER as the sole metric as well as AMI as the sole dataset. Future work should focus on generalizing these findings across diverse datasets and exploring additional metrics for a more comprehensive evaluation of diarization systems.

In conclusion, while Pyannote's speaker-diarization-3.1 pipeline demonstrates strong diarization results, significant challenges persist in accurately processing overlapping speech. The oracle component analysis provided valuable insights into the strengths and weaknesses of individual pipeline components, guiding future research directions to enhance the state-of-the-art in speaker diarization.

[1]This does not provide a full picture for embeddings as an oracle embedding in this setup was only possible from oracle segmentations

Bibliography

[1] Xavier Anguera, Chuck Wooters, and Javier Hernando. Acoustic beamforming for speaker diarization of meetings. *Audio, Speech, and Language Processing, IEEE Transactions on*, 15:2011 – 2022, 10 2007.

[2] Hervé Bredin. pyannote-audio/questions/pyannote.question.md at develop · pyannote/pyannote-audio, Mar 2023.

[3] Hervé Bredin and Antoine Laurent. End-to-end speaker segmentation for overlap-aware resegmentation, 2021.

[4] Latané Bullock, Hervé Bredin, and Leibny Paola García-Perera. Overlap-aware diarization: Resegmentation using neural end-to-end overlapped speech detection. *HAL (Le Centre pour la Communication Scientifique Directe)*, 05 2020.

[5] Scott Saobing Chen and Ponani S. Gopalakrishnan. Clustering via the bayesian information criterion with applications in speech recognition. *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98 (Cat. No.98CH36181)*, 2:645–648 vol.2, 1998.

[6] Zhihao Du, Shiliang Zhang, Siqi Zheng, and Zhijie Yan. Speaker overlap-aware neural diarization for multi-party meeting analysis, 2022.

[7] Belkacem Fergani, Manuel Davy, and Amrane Houacine. Speaker diarization using one-class support vector machines. *Speech Communication*, 50(5):355–365, 2008.

[8] Yusuke Fujita, Naoyuki Kanda, Shota Horiguchi, Kenji Nagamatsu, and Shinji Watanabe. End-to-end neural speaker diarization with permutation-free objectives. *Conference of the International Speech Communication Association*, 09 2019.

[9] Yusuke Fujita, Naoyuki Kanda, Shota Horiguchi, Yawen Xue, Kenji Nagamatsu, and Shinji Watanabe. End-to-end neural speaker diarization with self-attention, 2019.

[10] Daniel Garcia-Romero, David Snyder, Gregory Sell, Daniel Povey, and Alan McCree. Speaker diarization using deep neural network embeddings. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4930–4934, 2017.

[11] Shota Horiguchi, Yusuke Fujita, Shinji Watanabe, Yawen Xue, and Paola Garcia. Encoder-decoder based attractors for end-to-end neural diarization. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30:1493–1507, 2022.

[12] Shota Horiguchi, Yusuke Fujita, Shinji Watanabe, Yawen Xue, and Kenji Nagamatsu. End-to-end speaker diarization for an unknown number of speakers with encoder-decoder based attractors, 2020.

[13] Marijn Huijbregts, David Van Leeuwen, and Chuck Wooters. Speaker diarization error analysis using oracle components. *IEEE Transactions on Audio, Speech Language Processing*, 20:393–403, 02 2012.

[14] Marijn Huijbregts and Chuck Wooters. The blame game: performance analysis of speaker diarization system components. pages 1857–1860, 08 2007.

[15] Keisuke Kinoshita, Marc Delcroix, and Naohiro Tawara. Advances in integration of end-to-end neural and clustering-based diarization for real conversational speech, 2021.

[16] Keisuke Kinoshita, Marc Delcroix, and Naohiro Tawara. Integrating end-to-end neural and clustering-based diarization: Getting the best of both worlds, 2021.

[17] Federico Landini, Alicia Lozano-Diez, Mireia Diez, and Lukáš Burget. From simulated mixtures to simulated conversations as training data for end-to-end neural diarization, 2022.

[18] Federico Landini, Ján Profant, Mireia Diez, and Lukas Burget. Bayesian hmm clustering of x-vector sequences (vbx) in speaker diarization: Theory, implementation and analysis on standard tasks. *Computer Speech Language*, 71:101254–101254, Jan 2022.

[19] Tao Liu and Kai Yu. Ber: Balanced error rate for speaker diarization, 2022.

[20] Giovanni Morrone, Samuele Cornell, Desh Raj, Luca Serafini, Enrico Zovato, Alessio Brutti, and Stefano Squartini. Low-latency speech separation guided diarization for telephone conversations, 2022.

[21] Alexis Plaquet and Hervé Bredin. *Powerset multi-class cross entropy loss for neural speaker diarization.* Oct 2023.

[22] Neville Ryant, Prachi Singh, Venkat Krishnamohan, Rajat Varma, Kenneth Church, Christopher Cieri, Jun Du, Sriram Ganapathy, and Mark Liberman. The third dihard diarization challenge, 2021.

[23] Naohiro Tawara, Marc Delcroix, Atsushi Ando, and Atsunori Ogawa. Ntt speaker diarization system for chime-7: Multi-domain, multi-microphone end-to-end and vector clustering diarization. In *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 11281–11285, 2024.

[24] Hongji Wang, Chengdong Liang, Shuai Wang, Zhengyang Chen, Binbin Zhang, Xu Xiang, Yanlei Deng, and Yanmin Qian. Wespeaker: A research and production oriented speaker embedding learning toolkit, 2022.

Appendix A

Appendix

A.1 Oracle Component Analysis Baseline (additional figures)

A.2 Oracle Clustering (additional figures)

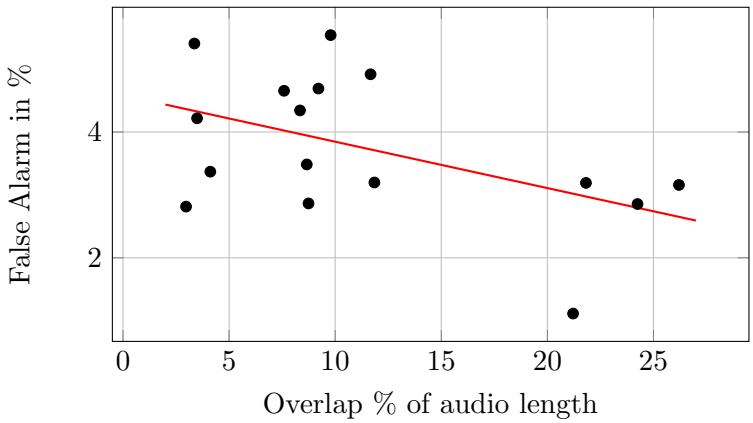

Figure A.1: False Alarm vs Relative Overlap amount

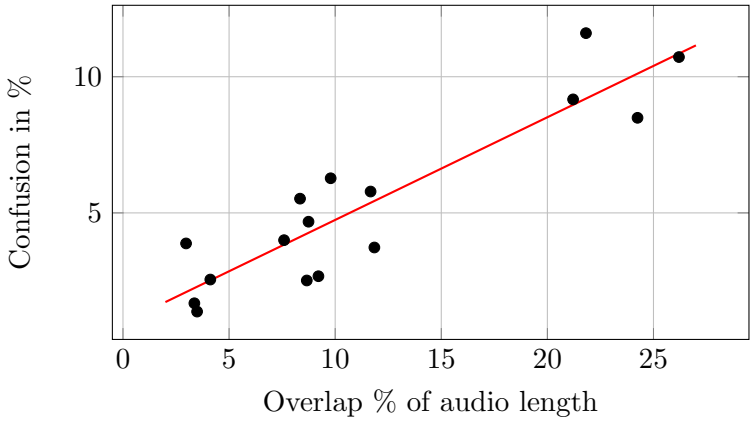

Figure A.2: Confusion vs Relative Overlap amount

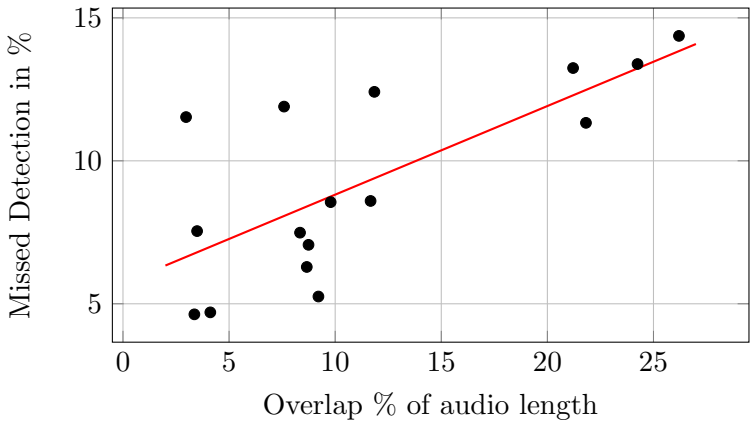

Figure A.3: Missed Detection vs Relative Overlap amount

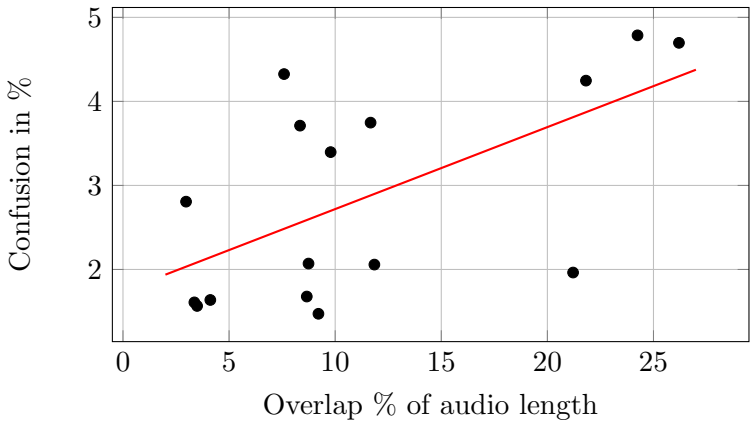

Figure A.4: Oracle Clustering Confusion vs Relative Overlap amount

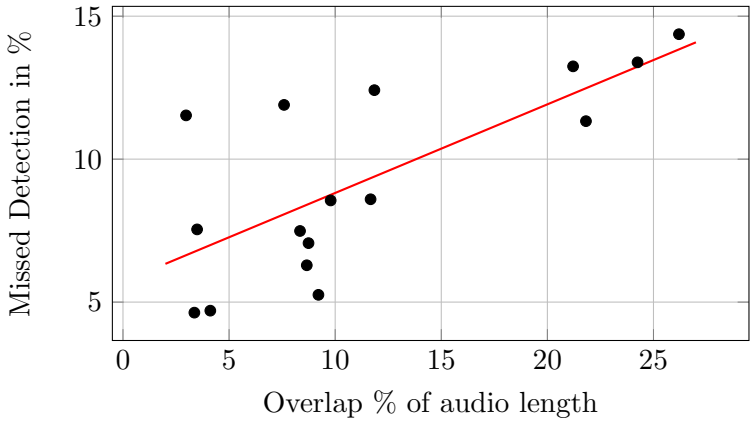

Figure A.5: Oracle Clustering Missed Detection vs Relative Overlap amount

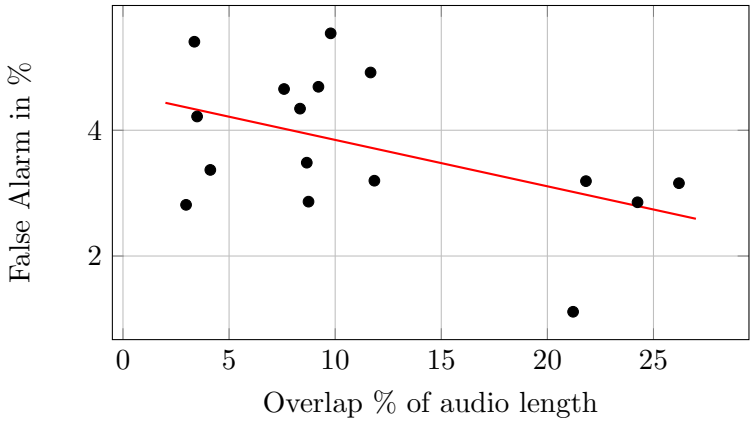

Figure A.6: Oracle Clustering False Alarm vs Relative Overlap amount