UTRECHT UNIVERSITY

Department of Information and Computing Science

Applied Data Science master thesis

Extending the Automatic Speech Recognition Model Whisper with a Speaker Diarization Model

Candidate:

Anouk Christina Mul

1429264

In cooperation with:

The Dutch National Police

Examiners:

Dr. I.R. (Ioana) Karnstedt-Hulpus

V.J. Shahrivari

Daily supervisors:

D.S. Islakoglu

N. Hulzebosch

July 7, 2023

Abstract

The Dutch National Police faces the challenge of efficiently processing and transcribing a significant amount of audio data collected during investigations. To assist detectives in their work, artificial intelligence (AI) models, such as Whisper, an automatic speech recognition (ASR) model, are implemented into user-friendly applications. However, Whisper lacks the ability to distinguish between speakers, limiting its application in scenarios involving multiple speakers and overlapping speech. This thesis explores the performance of speaker diarization pipelines from PyAnnote and NeMo on the VoxConverse and NFI-FRITS datasets. Additionally, experiments are conducted to improve the performance of the pipelines on both datasets by choosing appropriate hyperparameter settings. By incorporating a speaker diarization system alongside Whisper, the aim is to enhance the robustness and comprehensiveness of an existing speech-to-text application. The evaluation reveals promising results, with hyperparameter tuning and domain-specific configurations significantly improving the Diarization Error Rate (DER) for both datasets. PyAnnote benefits from adjusted segmentation and clustering thresholds, as well as changes in the clustering method. NeMo's clustering diarizer outperforms the neural diarizer, and domain-specific configurations enhance performance. In general, NeMo demonstrates superior performance on both datasets in terms of Diarization Error Rate (DER) compared to PyAnnote. However, this improved performance comes at the cost of increased computational requirements in terms of speed and memory usage. By augmenting Whisper with speaker diarization, investigators can efficiently analyze transcribed text ascribed to individual speakers, improving the accuracy and efficiency of audio data analysis. Further research should focus on compiling an enlarged domain-specific dataset with varying numbers of speakers to enable more specific hyperparameter tuning and achieve better performance results. Additionally, optimizing resource usage for the superior NeMo model would enhance its speed and memory efficiency. Overall, this research contributes to advancing speaker diarization methods alongside the Whisper ASR model. These advancements will lead to more effective speech analysis tools for law enforcement and other fields relying on accurate and comprehensive audio processing. The code is available at `https://github.com/anouk1512/MSc_WhisperSpeakerDiarization.git`.

Contents

List of abbreviations

AI	Artificial Intelligence
ASR	Automatic Speech Recognition
CNN	Convolutional Neural Network
DER	Diarization Error Rate
EEND	End-to-End Neural Network
FA	False Alarm
HAC	Hierarchical Agglomerative Clustering
MS	Missed Speech
MSDD	Multi-Scale Diarization Decoder
NFI-FRITS	Netherlands Forensic Institute's Forensically Realistic Intercepted Telephone Speech
RTF	Real-Time Factor
RTTM	Rich Transcription Time Marked
SAD	Speech Activity Detection
SC	Speaker Confusion
SD	Speaker Diarization
TROI	Team Rendement Operationele Informatie
TST	Total Speech Time
VAD	Voice Activity Detection

1. Introduction

1.1 Motivation and context

The Dutch National Police collects a significant amount of audio during investigations. Listening to and transcribing audio messages manually is a highly time-consuming task. As a result, only a partial transcription is usually carried out, leaving potentially crucial information untapped. The police department *Team Rendement Operationele Informatie* (TROI) implements artificial intelligence models to expedite these kind of procedures and assist detectives with their work. TROI's data specialists develop user-friendly applications to help investigators structure and analyze the vast amounts of data they encounter daily.

TROI's latest application incorporates the state-of-the-art speech-to-text algorithm Whisper [1], enabling the automatic conversion of audio into transcribed text. This advanced functionality empowers investigators to efficiently search for specific keywords within audio messages and access those particular segments directly. Consequently, investigators can save significant time and effort by focusing only on the relevant information, rather than having to listen to the entire audio content, which may or may not contain relevant details.

Since the audio files collected by the police contain very sensitive data, their application cannot run in the cloud, but must run locally. Therefore, they have to develop their own system based on available open-source implementations. The new transcription application is based on the open-source model Whisper.

Whisper is an automatic speech recognition (ASR) model developed by OpenAI [1]. Its primary objective is to convert spoken language into written text, making it a valuable tool for various applications such as transcription services, voice assistants, and more. Unlike traditional methods that rely on high-quality transcriptions, Whisper is intentionally trained on lower-quality transcriptions. This allowed access to a larger volume of data compared to fully supervised experiments. As a result, Whisper benefits from enhanced scalability and diversity in its training data [1].

Since Whisper is trained on data from many different domains and on 96 languages besides English, it approaches human-level accuracy and robustness and generalizes well on new data [1]. Additionally, the system provides complete interpunction in its transcriptions, can automatically detect languages and even accurately transcribes if speakers have a heavy accent. Whisper processes audio roughly 1.5x times faster than real-time on a GPU, so it would take around 6 hours to transcribe 9 hours of audio.

Although the observed transcription speed and performance of Whisper on police datasets are promising enough to launch the newly developed application, there remains potential to further improve the current implementation. One limitation of this ASR model is its inability to distinguish between speakers. Consequently, the transcription of a conversation will be outputted as if uttered by one person.

Speaker diarization (SD) is distinguishing "who spoke when" in an audio stream [2]. With the implementation of a speaker diarization system alongside Whisper, the transcriptions can be ascribed to speakers, making analysis within the speech-to-text application more robust and comprehensive. Besides, speaker diarization can deliver interesting meta-data about audio files, such as the amount of speakers, speaking time per speaker and total speaking time per file. Implementing this would allow investigators to do additional queries, for example filtering on recordings with more than X speakers or selecting long monologues from a specific speaker.

The police encounters audio in various languages, with Dutch being the most prevalent. Typically, speaker diarization models are evaluated using English datasets. Another characteristic of the collected data is its low-quality audio. While most datasets contain "staged" speech, the police data consists of "real-life" audio fragments, with overlapping speech, microphones of poor quality and background noise. Consequently, the chosen

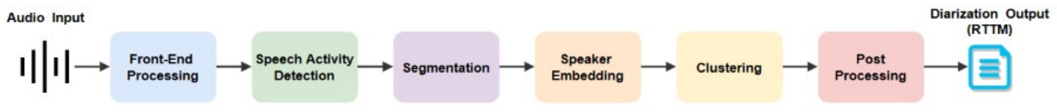


Figure 1.1: Traditional speaker diarization system [3]

```
SPEAKER {file_name} 1 {start_time} {duration} <NA> <NA> {speaker_name} <NA> <NA>
```

Figure 1.2: Structure of the Rich Transcription Time Marked (RTTM) format

speaker diarization model must meet additional criteria, such as demonstrating strong performance on Dutch audio files and effectively handling overlapping speech.

This thesis focuses on finding a suitable speaker diarization model, aiming to improve and advance this aspect of speech analysis alongside Whisper. Therefore, the research question will be the following:

> *"How can the ASR model Whisper be augmented with a speaker diarization method that achieves a low Diarization Error Rate on Dutch audio files containing overlapping speech and multiple speakers, while maintaining reasonable speed in terms of Real-Time Factor?"*

1.2 Background

There are some essential sub-tasks performed in traditional SD systems to achieve the classification of speakers required for speaker diarization. These independent steps are presented in Figure 1.1 and are still relevant for modern diarization methods [3].

The input of a SD system is usually a raw audio file. Front-end processing steps are employed to eliminate any distortions from background noise. Next, speech and non-speech parts are separated using Speech Activity Detection (SAD). The portions of speech are then split into segments and each segment is converted into a speaker embedding, which is a mathematical representation of sound, that captures important characteristics of the audio. Similar segments are clustered together to distinguish between speakers. Every cluster then corresponds to one unique speaker. As postprocessing steps, these clustering results can be optimized in various ways. The output is provided as Rich Transcription Time Marked (RTTM) format. This is a text format with one line per speech turn, conforming to the structure displayed in Figure 1.2.

Raw audio files can be encoded in various formats. Among them, the WAV format is the most commonly used for automatic speech recognition and speaker diarization tasks. In the WAV format, audio files are stored without compression, preserving both the quality and information, which is crucial for accurate analysis of speaker-related tasks [4].

1.2.1 Evaluation metrics

Multiple evaluation metrics are employed to assess the performance of speaker diarization systems. The Diarization Error Rate (DER) is the most commonly used in the evaluation of diarization methods, and also serves as diarization metric for most speaker recognition challenges [5]. The input for the DER per audio file is the reference annotation and the annotation predicted by a model, both delivered in the aforementioned RTTM format. As output, the DER computes the proportion of incorrectly assigned speech time to the total speech time [3][6]. The DER is defined formally as follows:

$$DER(reference, prediction) = \frac{(FA + MS + SC)}{TST} \tag{1.1}$$

The incorrectly assigned speech time is the sum of three different errors:

- False Alarm (FA): duration of non-speech incorrectly labeled as speech
- Missed Speech (MS): duration of speech incorrectly labeled as non-speech
- Speaker Confusion (SC): duration of time in which wrong speaker label is assigned to a speaker

The denominator Total Speech Time (TST) refers to the sum of all speakers' speech duration. The result of the DER is a value between 0 and 1, where 0 indicates perfect speaker diarization. Most implementations of DER support using a collar to account for slight temporal deviations between predicted and reference annotations. This collar can be set to a positive number, denoting the seconds at both the beginning and the end of a speaker turn that are regarded as correctly predicted anyway. A common setting for the collar is 0.25 seconds [3].

The second metric is Real-Time Factor (RTF), which evaluates the speed of audio-processing methods [7]. It measures the ratio between the duration of an audio file and the time it takes for a particular method to process that file. The formula for RTF is given in Equation 1.2, where *d* denotes the duration of an audio file and *f(d)* is the execution time of a specific function on that audio file. In our case, f(d) is the application of a speaker diarization model onto an audio file.

$$RTF = \frac{f(d)}{d} \tag{1.2}$$

The result will be a positive number. If the RTF equals 1, the computation is done 'in real time'. A value higher or lower than 1 indicates slower or faster processing, respectively.

As last metric, the memory usage of a model is monitored, in terms of bytes allocated during execution. Together, these three metrics give a complete comparison of speaker diarization models, aligning with the aspects that are of interest to the police.

1.3 Related work

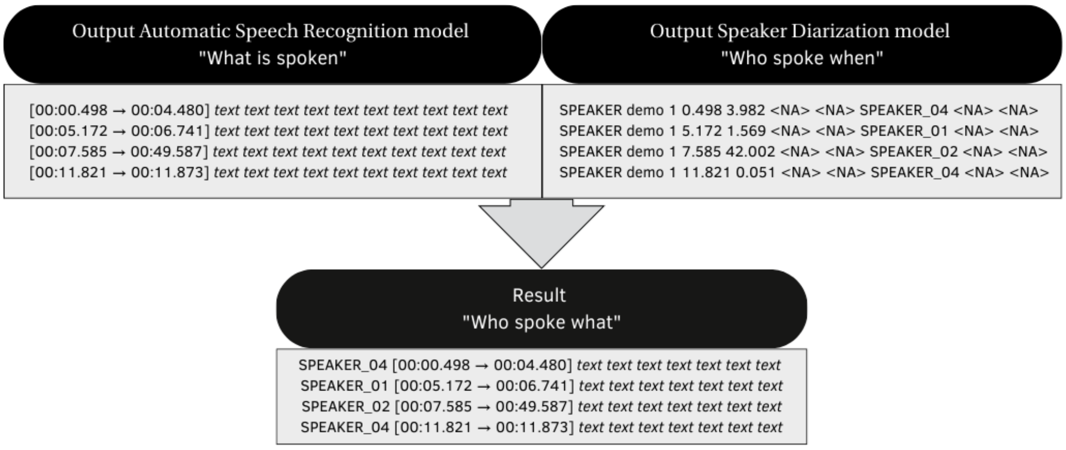
In this section, related literature is reviewed to explore recent improvements in speaker diarization methods. This begins with the rise of deep learning in the last decade, with more robust models that reach higher performance [3]. First, optimizations to one or more of the subtasks, as outlined in Figure 1.1, were introduced to enhance the performance of diarization methods.

In the work of [8], the commonly used i-vector that creates speaker embeddings using factor analysis is replaced by d-vectors, which are speaker representations obtained from a deep neural network. This neural network applies multiple non-linear transformations to convert an utterance into a vector. When d-vectors are extended with a temporal pooling layer, they can deal more accurately with segments of varying lengths and can capture more details. These vectors are called *extended vectors* or *x-vectors* [9]. Others substitute the way in which the clustering of utterances is done. Usually, the clustering module uses some variation of *hierarchical agglomerative clustering* (HAC). This unsupervised clustering module can be replaced by, for example, an *unbounded interleaved-state recurrent neural network* [10]. This is a trainable model that dynamically adapts while learning temporal data from examples, in an online generative process. In [11], affinity propagation clustering employs an iterative process to search for exemplars within the neural speaker embeddings, aiming to find a representative point for each cluster.

Besides individual optimization of modules, it is even more powerful to jointly optimize (neural) building blocks [3]. These type of implementations are called *end-to-end neural diarization* (EEND) methods. In the work of [12], the neural building blocks of the diarization process are jointly optimized through back-propagation. The goal is to optimize the parameters of the entire system to achieve better performance and synergy between the different components. This can done with every type of modules within a pipeline [11].

This research aims to augment an automatic speech recognition (ASR) model with a speaker diarization (SD) model. This can be achieved by combining the outputs of the two models. It is beyond the scope of this section to explain in detail how an ASR model like Whisper works; it is only important how its output looks and how this can be combined with the output of an SD model. This is displayed in Figure 1.3. The combination of the outputs of the two models results in the determination of "who spoke what" in an audio stream.

A hot topic in recent research on ASR and SD systems is the combination of both systems into one complete system. Such an approach can be found in [13][14]. The authors developed an *end-to-end speaker-attributed au-*


Output Automatic Speech Recognition model
"What is spoken"
Output Speaker Diarization model
"Who spoke when"
Result
"Who spoke what"

```
[00:00.498 → 00:04.480] text text text text text text text text text text
[00:05.172 → 00:06.741] text text text text text text text text text text
[00:07.585 → 00:49.587] text text text text text text text text text text
[00:11.821 → 00:11.873] text text text text text text text text text text
```

```
SPEAKER demo 1 0.498 3.982 <NA> <NA> SPEAKER_04 <NA> <NA>
SPEAKER demo 1 5.172 1.569 <NA> <NA> SPEAKER_01 <NA> <NA>
SPEAKER demo 1 7.585 42.002 <NA> <NA> SPEAKER_02 <NA> <NA>
SPEAKER demo 1 11.821 0.051 <NA> <NA> SPEAKER_04 <NA> <NA>
```

```
SPEAKER_04 [00:00.498 → 00:04.480] text text text text text text text
SPEAKER_01 [00:05.172 → 00:06.741] text text text text text text text
SPEAKER_02 [00:07.585 → 00:49.587] text text text text text text text
SPEAKER_04 [00:11.821 → 00:11.873] text text text text text text text
```

Figure 1.3: Combination of ASR system and SD system outputs

tomatic speech recognition system, which recognizes overlapping speech and can work with any number of speakers. Although their reported results are quite promising, combined ASR and SD systems are beyond the scope of this research, since the goal is finding a suitable speaker diarization method for Whisper.

The audio files from the police dataset have some challenging characteristics. Therefore, the speaker diarization method that is selected must be able to deal with a flexible number of speakers which is unknown beforehand, some overlapping speech, recordings of poor quality and various types of audio files. Additionally, considering the application's offline nature, it necessitates an open-source implementation that can run locally.

1.3.1 PyAnnote

Based on the aforementioned criteria for the speaker diarization technique and the recent advances in jointly optimized pipelines, the open-source toolkit that is described in [15] is a promising implementation alongside Whisper. It is called `pyannote.audio` and it is built from neural end-to-end building blocks. The hyperparameters of these blocks are jointly optimized to minimize its DER. Combining the building blocks results in a complete speaker diarization pipeline, which transforms audio from the waveform directly into RTTM format speaker annotations, as displayed in Figure 1.4.

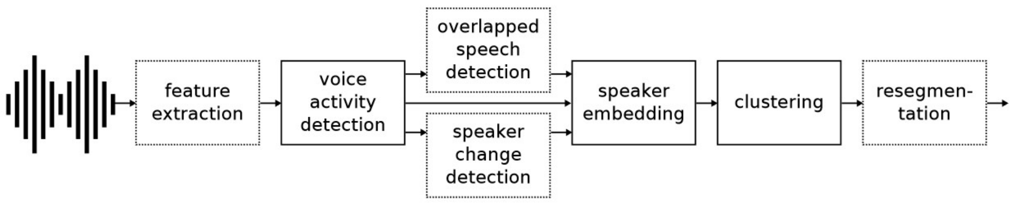


Figure 1.4: PyAnnote speaker diarization pipeline [15]

As feature extraction module, PyAnnote uses the convolutional neural network architecture SincNet [16]. It incorporates sinc functions to capture different frequency components and thus discriminative speaker features from raw audio files. All the detection modules (voice activity detection, overlapped speech detection and speaker change detection) are performed on short fixed-length sub-sequences using overlapping sliding windows. This results in multiple prediction scores, which are averaged into a final prediction score. For all three modules, there is a tunable threshold initiated. At every time step t, the prediction score is compared to the predefined threshold, resulting in a score of 1 when exceeding the threshold, and 0 otherwise. A score of 1 then denotes the detection of voice, speaker change or overlapping speech.

Next, the speaker embeddings are constructed using the aforementioned x-vectors. They serve as input for the clustering phase, which is based on cosine distance metrics, centroid linkage and hierarchical agglomerative clustering. During the re-segmentation step, multiple epochs are executed to enhance the accuracy of speech turn boundaries and speech labels, based on the implementation in [11]. Eventually, after every building block is trained independently, they are combined into a pipeline with optimized hyperparameters. It is trained, tuned and tested on different datasets, containing audio from settings like meetings, broadcast news and 11 other conversational domains. This makes the pipeline robust and generalizable, especially for audio from similar settings [15]. The pipeline is ready-to-use, but can also be adjusted using own data. This approach will be explored in this research. The most recent version of `pyannote.audio` (2.1.1) has some improvements to the original pipeline [17]. These include smaller overlapping windows, separate diarization and embedding modules and even more options to adjust parameters or fine-tune the speaker segmentation model itself. Multiple winning implementations of speaker diarization challenges use an implementation that include `pyannote.audio` utilities [5] [17].

1.3.2 NeMo

NVIDIA also developed a speaker diarization pipeline, which is displayed in Figure 1.5. It is part of the `nemo` toolkit, which provides conversational AI models [18]. The goal of the pipeline is to create an overlap-aware speaker diarization system that can handle a variable number of speakers. The pretrained modules used in the pipeline are briefly explained below.

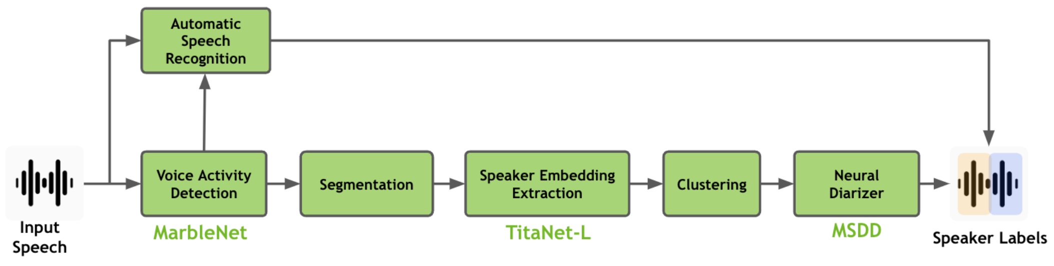


Figure 1.5: NeMo speaker diarization pipeline[1]

The voice activity detection (VAD) module uses MarbleNet, which is a convolutional neural network (CNN) that detects and classifies speech and non-speech parts [19]. After VAD is performed, one can choose to only perform speaker diarization or combine it with automatic speech recognition. The speech segments are subjected to the process of extracting speaker embeddings using TitaNet-L in order to proceed with speaker diarization [20]. The model of TitaNet-L is also based on a CNN, with a pooling layer that is responsible for transforming variable-length speech segments into fixed-length embeddings. The speaker embeddings can be clustered using either a standalone clustering algorithm or a clustering algorithm combined with a neural diarizer, called the *Multi-Scale Diarization Decoder* (MSDD) [21]. Both approaches use various segment lengths simultaneously, instead of

[1]`https://docs.nvidia.com/deeplearning/nemo/user-guide/docs/en/stable/asr/speaker_diarization/intro.html`

one fixed speaker segment length, to extract speaker characteristics. Then, the results from multiple scales are combined, using a weighting per scale. For the clustering diarizer, these weights are fixed, but for the neural diarizer model, the weight assigned to each scale is determined using a CNN. This results in estimated speaker labels per speech segment.

The rest of this thesis is organized as follows. Chapter 2 gives a description of the datasets used for the experiments and the preprocessing steps that are performed. Chapter 3 details the methods of all executed experiments. In Chapter 4, the results of the experiments are reported. Chapter 5 gives an interpretation of the results and their implications. At last, Chapter 6 discusses possible limitations and presents recommendations for future work.

2. Data

2.1 Description of the data

The police collect various audio files during investigations, which vary in quality and language. For research purposes, they work with a dataset set known as the Netherlands Forensic Institute's Forensically Realistic Intercepted Telephone Speech database (NFI-FRITS) [22]. The dataset comprises recorded telephone conversations obtained from actual police investigations. These conversations encompass speech in multiple languages, with a majority of the audio being in Dutch. The dataset also includes recordings in Moroccan Arabic, Berber, and Turkish. However, due to the sensitivity of the data, direct access to this dataset is restricted. Experiments with the dataset could only be executed on-site. Thus, for exploratory experiments, an additional dataset is used.

The NFI-FRITS dataset consists of 4188 audio files of telephone conversations between two speakers. The relative uniformity of these audio files makes the dataset very domain-specific. During the composition of the dataset, the files were anonimized, so for every fragment that contains sensitive information, all audio values were set to zero. Next, they marked and removed the background noise. Nevertheless, since the quality of speech is not optimal at all times, the audio quality can be challenging. Only files with a duration between 30 and 600 seconds were permitted. This results in a dataset that comprises 165 hours of speech, with an average of 142 seconds per file. For the experiments in this thesis, only a subset of the NFI-FRITS is used, as discussed in Section 2.2.

The English open-source dataset VoxConverse is selected to conduct exploratory experiments [23]. This dataset complements the NFI-FRITS dataset since the police collect more types of audio files than telephone conversations between 2 speakers, not only limited to the languages in the NFI-

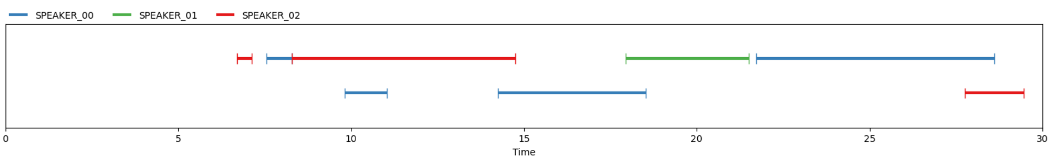


Figure 2.1: Visual representation of overlapping speech and multiple speakers

Table 2.1: Characteristics of NFI-FRITS and VoxConverse test sets; entries with 3 values are reported as min/mean/max; "-" denotes unavailable statistics

	# files	# hours	duration (s)	# spks	% overlap
NFI-FRITS	228	11	57/173/625	2	-
VoxConverse	232	44	26/676/1200	1/6.5/21	0/3.1/29.8

FRITS dataset. VoxConverse is widely used in diarization research, since it is speaker-annotated and approaches real-life conditions of spontaneous speech. The VoxConverse dataset is constructed from YouTube videos from a variety of domains. The videos are selected using keywords like 'panel debate' and 'discussion' to ensure the inclusion of videos where multiple people are talking alternately or at the same time, which is suitable for the speaker diarization task. The recordings contain a large number of speakers, quick exchanges of speech and a lot of background noise. Figure 2.1 illustrates the visual representation of a segment from an audio file exhibiting overlapping speech and the presence of multiple speakers.

2.2 Preparation of the data

The datasets described above could not be used as is for this research; instead, subsets are selected. In the case of the NFI-FRITS, only a subset of Dutch conversations with two speakers was available for research. Table 2.1 displays the other characteristics of this subset. It contains 228 audio files, together around 11 hours of material. The files are 1 to 10 minutes long, and every conversation only has 2 speakers. Information on overlapping speech is not available. As mentioned before, the dataset could only be accessed in a secured environment and therefore cannot be shared with other parties.

A subset from the VoxConverse dataset is utilized due to various constraints encountered during the research. These constraints include limited

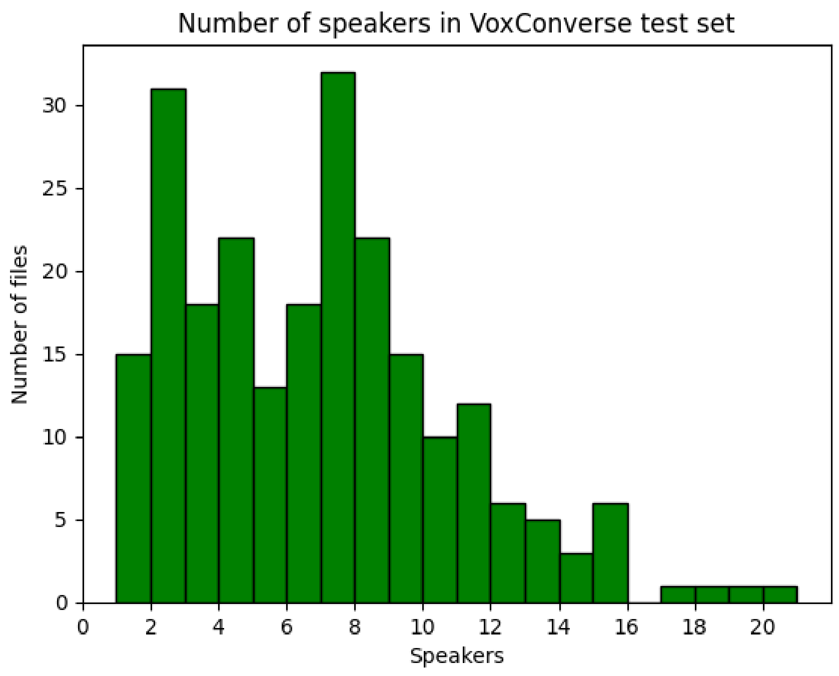
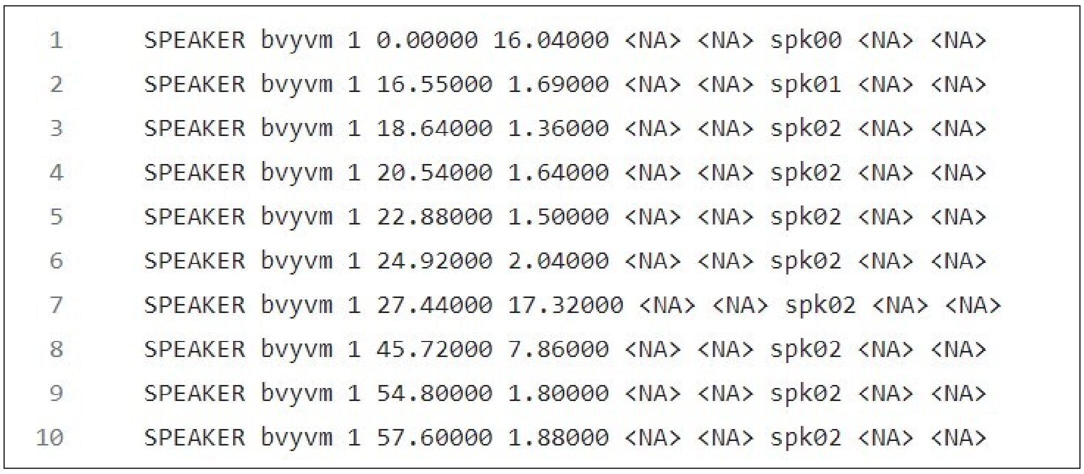


Figure 2.2: Histogram of number of speakers per file in VoxConverse test set

storage capacity, insufficient GPU resources, and time constraints that made it impractical to use the entire dataset. The VoxConverse dataset is already partitioned into randomly drawn subsets, and for the purpose of this research, the test set is selected. It contains 44 hours of speech, distributed amongst 232 audio files, which results in an average length of 11 minutes per audio file. There is a diverse range in the number of speakers per file, which is displayed in the histogram in Figure 2.2. It ranges from 1 speaker to 21 speakers per file. This makes the VoxConverse subset a good addition to the NFI-FRITS subset, which only contains two speakers per file. Furthermore, 3% of the speech in VoxConverse overlaps each other, with some files exhibiting as much as 30% overlap. Both the audio files and the annotations are accessed through GitHub.[1] Eventually, another smaller subset is created from the VoxConverse test set featuring a single speaker. This subset is used for additional experiments, as will be described in Chapter 3.

No other preprocessing steps are performed, since both datasets are provided in suitable formats to facilitate the application of Whisper, PyAnnote

[1]`https://github.com/joonson/voxconverse`



```
SPEAKER bvyvm 1 0.00000 16.04000 <NA> <NA> spk00 <NA> <NA>
SPEAKER bvyvm 1 16.55000 1.69000 <NA> <NA> spk01 <NA> <NA>
SPEAKER bvyvm 1 18.64000 1.36000 <NA> <NA> spk02 <NA> <NA>
SPEAKER bvyvm 1 20.54000 1.64000 <NA> <NA> spk02 <NA> <NA>
SPEAKER bvyvm 1 22.88000 1.50000 <NA> <NA> spk02 <NA> <NA>
SPEAKER bvyvm 1 24.92000 2.04000 <NA> <NA> spk02 <NA> <NA>
SPEAKER bvyvm 1 27.44000 17.32000 <NA> <NA> spk02 <NA> <NA>
SPEAKER bvyvm 1 45.72000 7.86000 <NA> <NA> spk02 <NA> <NA>
SPEAKER bvyvm 1 54.80000 1.80000 <NA> <NA> spk02 <NA> <NA>
SPEAKER bvyvm 1 57.60000 1.88000 <NA> <NA> spk02 <NA> <NA>
```

Figure 2.3: Example of a RTTM file containing annotations for a VoxConverse audio file

and NeMo models. This is the WAV format for the audio files and annotations in RTTM format. In Figure 2.3, an annotation file for one VoxConverse audio file is depicted. Both format types are explained before in Section 1.2. Subsequently, it was simple to check diarization results between the output of an audio file and the corresponding annotations file, since these were stored with identical, unique file names and only a differing file extension. For example, an audio file from the VoxConverse dataset is stored as 'aepyx.wav' and its corresponding annotation as 'aepyx.rttm'.

3. Method

The aim of this research is to investigate speaker diarization methods that can enhance the Whisper automatic speech recognition model. This section will describe the methods used to experiment with various speaker diarization algorithms. The chosen implementations must satisfy specific requirements set by the police in order to produce meaningful results. These requirements include the following: the implementation should be written in Python, designed to run locally, provided as an open-source solution, regularly or recently updated, and accompanied by a pretrained model. The Python libraries `pyannote.audio` and `NeMo` that are discussed in Section 1.3 meet these requirements and are selected for experiments. In the experiments described below, the speaker diarization implementations of both libraries are compared and optimized using the VoxConverse and the NFI-FRITS subsets. Eventually, the best performing models for both datasets are evaluated for all performance metrics.

Each technique's effectiveness is quantified using a collection of distinct metrics. The accuracy is determined primarily using the Diarization Error Rate (DER), as introduced in Section 1.2.1, which is calculated utilizing PyAnnote's corresponding metric.[1] The efficiency of the procedure, on the other hand, is evaluated in terms of the Real-Time Factor (RTF), a measure of implementation speed also expounded upon in the same chapter. The execution time per method is tracked and this is divided by the file duration to extract the RTF. Third, the memory usage per model is recorded using the `tracemalloc` function. The memory is reported in absolute numbers of peak usage in bytes, which refers to the largest amount of memory allocated during execution.

For the experiments described below that concern hyperparameter tun-

[1] `https://pyannote.github.io/pyannote-metrics/reference.html`

ing, only the DER is reported, since preliminary experiments showed that different hyperparameter settings did not affect RTF or memory usage significantly. For the final comparison of the optimized PyAnnote and NeMo versions, the three metrics are used, to give a complete performance evaluation.

3.1 Baseline comparison: PyAnnote and NeMo

In the first experiment, two basic implementations of PyAnnote and NeMo are compared in terms of DER. For PyAnnote, this includes the pretrained pipeline for speaker diarization.[2] For NeMo, this is the clustering diarizer pipeline.[3]

The PyAnnote pretrained pipeline is applied separately to each audio file. This involves looping over every file in the dataset, executing the speaker diarization pipeline, and calculating the DER between the predicted and reference annotation. Since the NeMo implementation automatically calculates the DER with a collar of 0.25 seconds, this is also applied to the scoring of the PyAnnote pipeline.

The NeMo implementation requires the input of a 'manifest'. For both the audio files and the annotation files, a text file is created with the path to every file on a new line. With these text files, a manifest file is created that serves as the input for NeMo.[4] Therefore, it doesn't require looping over every file separately. Inside the model, the DER of the whole subset is calculated. A CSV file is generated for each dataset containing the respective results for all audio files. When comparing the results between PyAnnote and NeMo, the DER values of the PyAnnote files are averaged. Next, the performance of the models on both datasets is evaluated by looking at the DER. These values will serve as baseline measures when improving the im-

[2]`https://huggingface.co/pyannote/speaker-diarization`

[3]`https://github.com/NVIDIA/NeMo/blob/main/tutorials/speaker_tasks/Speaker_Diarization_Inference.ipynb`

[4]`https://github.com/NVIDIA/NeMo/blob/main/scripts/speaker_tasks/pathfiles_to_diarize_manifest.py`

plementations.

The second and third experiment will focus on the improvement of both the PyAnnote and the NeMo implementations, by tuning relevant hyperparameters.

3.2 Optimization of PyAnnote

Besides the pretrained pipeline of PyAnnote, it is also possible to adjust the pipeline to your own data.[5] To create the adjusted pipeline, the data is collected into a database, with a train/development/evaluation split of 80/10/10 applied to it.[6] For the first exploratory experiment, a 30% subset of the VoxConverse dataset is used as training set, and accordingly a development and evaluation set is added. Next, the following steps are performed to fine-tune the pretrained models and optimize PyAnnote's speaker diarization pipeline [17]:

1. Fine-tune the speaker segmentation model
2. Optimize the hyperparameters
    (a) Optimize the segmentation threshold
    (b) Optimize the clustering threshold
3. Measure the performance of the adjusted pipeline

As reported in [17], these hyperparameters have the most influence on the performance of the pipeline and are therefore most important to fine-tune. [17] also reports on the results of the pipeline adjustment by training on the VoxConverse dataset, which gives improved results. Preliminary results showed that training on a subset of VoxConverse did not improve the performance of the speaker diarization pipeline, presumably because of the small amount of training data. Since it is not allowed to perform training on the NFI-FRITS data, due to the sensitivity of the data, the experiments pro-

[5] `https://github.com/pyannote/pyannote-audio/blob/develop/tutorials/adapting_pretrained_pipeline.ipynb`

[6] `https://github.com/pyannote/pyannote-database#configuration-file`

ceed without finetuning models on these datasets. Instead, they are used as validation sets to compare the best hyperparameters.

3.2.1 Hyperparameter tuning

The hyperparameter tuning of the PyAnnote pipeline involves experiments with different values of the segmentation threshold, clustering threshold and clustering method. The segmentation threshold influences the sensitivity of the speaker activity detection. A lower segmentation threshold will be more responsive to quick exchanges and overlapping speech, while a higher threshold will be more robust to slight changes in the audio, like background noise. The clustering threshold determines how similar speech segments or clusters must be to be grouped together. A lower threshold will result in smaller segment groups and, thus, more speakers being identified, while a higher threshold will lead to larger segment groups and, consequently, fewer speakers. Since hierarchical agglomerative clustering combines clusters that are similar, the linkage method serves as metric for similarity between clusters. The "average" method uses the average distance between all the data point pairs, while "centroid" linkage uses the data point per cluster that is most central to all other points to compare with other centroids. Adjusting the linkage method results in different similarity scores.

In case of a large dataset, it is useful to fine-tune the segmentation model, but since the NFI-FRITS dataset only contains 11 hours of speech, experiments will focus on optimizing the hyperparameters. The variation of the clustering method is added because former versions of PyAnnote used 'average' linkage instead of 'centroid' linkage, and this method is runner-up to centroid in the experiments performed by Bredin [17].

The values that are experimented with are based on different values in two different versions of PyAnnote, on the findings from the preliminary experiments with training the pipeline and on the range of numbers that are possible as input. These values are displayed in Table 3.1. This results in the following hyperparameter values in the experiment:

- Segmentation threshold: [0.30, 0.35, 0.40, 0.45, 0.50, 0.55, 0.60]

Table 3.1: Hyperparameters values

	Segmentation threshold	**Clustering threshold**	**Clustering method**
Pretrained pipeline version 2.1.1	0.444	0.715	centroid
Pretrained pipeline version < 2.1.1	0.444	0.582	average
Adjusted pipeline on 30% VC	0.638	0.722	centroid
Possible values	[0, 1]	[0, 2]	centroid, average, complete, single, Ward

- Clustering method: [average, centroid]
- Clustering threshold: [0.50, 0.55, 0.60, 0.65, 0.70, 0.75]

For every combination of values, a CSV file is created, denoting the DER with the adjusted hyperparameters. All the files are compared to evaluate the performance compared to the pretrained pipeline.

3.2.2 Passing a *max_speaker* argument

Additionally, there is the possibility of providing the lower and upper bounds of the number of speakers to the pretrained pipeline. The value of *min_-speakers* is set to 1. Based on the number of speakers reported in Section 2.2 and Figure 2.2, the values for *max_speakers* are adjusted to 1, 2, 5, 10, 15, or 20. When the value of *max_speakers* is reached, the model will stop searching for more speakers. The goal is to check whether it is advantageous to give PyAnnote an indication of the number of speakers and whether that improves the performance compared to the pipeline without the number of speakers declared. In addition to comparing the DER rates of different *max_speakers* values to the normal pipeline, an exploration is conducted on the impact of exceeding the *max_speakers* value on the DER when the number of speakers in an audio file increases.

The efficacy of the adjustments is assessed using a single speaker subset from the VoxConverse dataset. This approach is adopted given the frequent requirement of the police to process files involving only one speaker, like voice memo's. The subset of audio files from VoxConverse featuring a single speaker is used and the results for the various upper bounds on these files are compared to the general results.

3.3 Optimization of NeMo

The third experiment aims to find a better performing implementation of NeMo. The implementation of the diarizer described in the first experiment can both be augmented and adjusted. The following modifications to the diarizer parameters are premised on the hypothesis that these adjustments would have the most significant impact on the performance.

3.3.1 Clustering diarizer: hyperparameter tuning

As possible values for the clustering parameter *max_num_speakers*, the same values as for the PyAnnote optimization are used: [1, 2, 5, 10, 15, 20]. Again, this parameter controls whether the model will keep searching for more speakers or not. Unlike the PyAnnote pipeline argument, where the default value is set to *None*, omitting the *max_num_speakers* argument defaults to a value of 8 for *max_num_speakers*.

The configurations of the diarizer parameters can be adjusted by declaring a different domain type. There are two possible domain types: the default *telephonic*, which is "suitable for telephone recordings involving 2~8 speakers in a session"[7] or *meeting*, which is: "suitable for 3~5 speakers participating in a meeting"[8]. The configurations differ accordingly in voice activity detection and speaker embedding parameters. These parameters are set to different values to extract the right information from the audio files. Both configurations are applied to the datasets.

[7]`https://github.com/NVIDIA/NeMo/blob/main/examples/speaker_tasks/diarization/conf/inference/diar_infer_telephonic.yaml`

[8]`https://github.com/NVIDIA/NeMo/blob/main/examples/speaker_tasks/diarization/conf/inference/diar_infer_meeting.yaml`

3.3.2 Neural diarizer: hyperparameter tuning

The clustering diarizer can be replaced by a neural diarizer[9], since this can lead to more accurate results. This is the MSDD model described in Section 1.3.2, which uses a CNN to combine speaker characteristics into speaker labels. The neural diarizer is able to detect overlapping speech. The sigmoid threshold is the parameter that influences how quickly overlapping speech is detected. The default value is 0.70, and a lower threshold would mean more detection of overlapping speech, while a higher threshold would detect less. Therefore, the values that are experimented with are the following: [0.45, 0.50, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95]. The results of the various sigmoid threshold values are compared per dataset.

3.4 Best models comparison

In the final experiment, the top-performing models are compared to the baseline models in terms of Diarization Error Rate. Then, a complete evaluation is conducted, including the measurement of the Real-Time Factor and memory usage for each model. Eventually, these analyses aim to address the research question and provide meaningful insights on the performance of the models on the specific datasets.

[9]`https://github.com/NVIDIA/NeMo/blob/main/tutorials/speaker_tasks/Speaker_Diarization_Inference.ipynb`

4. Results

This chapter presents the findings of the experiments detailed in Chapter 3. All the experiments are performed on the complete test sets described in Section 2.2, unless explicitly noted. The results on both datasets are evaluated together, since this gives a more complete overview of the performance of the models in different domains. The evaluation metrics that are reported are the average Diarization Error Rate (DER) per dataset for the model baselines and model optimizations, and the average DER, average Real-Time Factor (RTF) and average memory usage for the final model comparisons. In the tables, the lowest values and thus, the best performing configurations, are marked in bold. Furthermore, the default values contain *(=def)* behind its parameter value.

4.1 Results of the baseline comparison

In Figure 4.1, the average DER for PyAnnote and NeMo on both datasets is displayed. The two models perform quite similarly on the VoxConverse dataset, with a DER around 0.120. The DER of the NFI-FRITS dataset, on the other hand, is very different for the two models. The PyAnnote baseline model reaches a DER of almost 2.5 times higher than the NeMo baseline model. The results of this experiment are used as baseline measures in the upcoming sections, in order to evaluate possible optimizations.

Since the PyAnnote model is applied separately to every audio file, it is possible to investigate the potential impact of the number of speakers on the DER score of VoxConverse audio files. To explore this correlation, a scatterplot is shown in Figure 4.2, revealing a positive slope of 0.004 for the regression line.

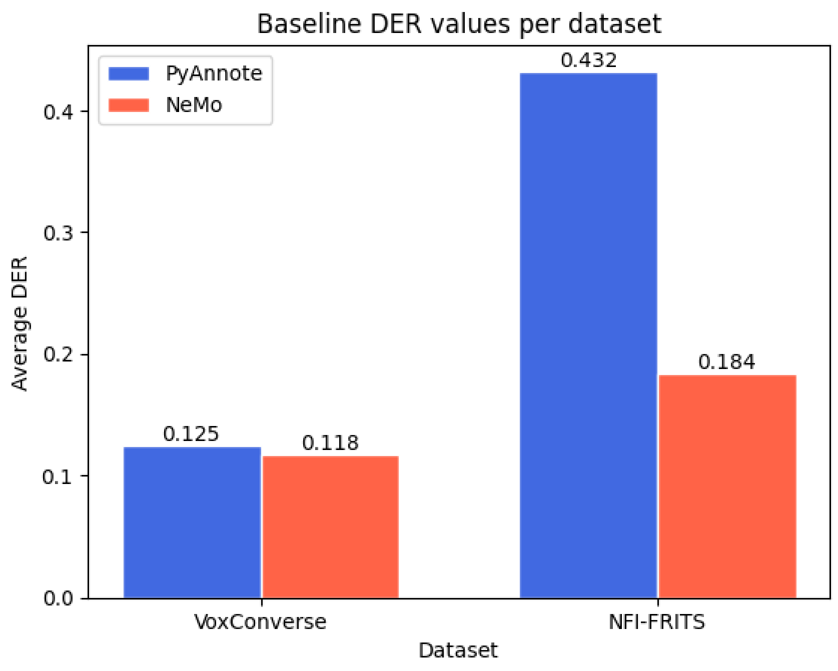


Figure 4.1: DER for the baseline models of PyAnnote and NeMo

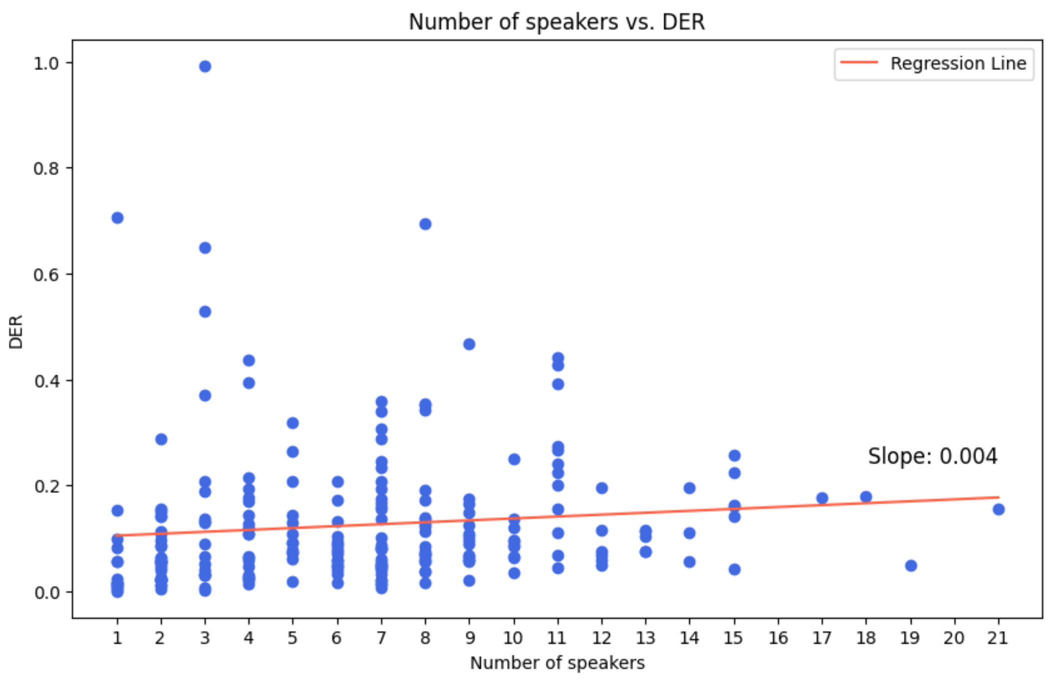


Figure 4.2: Scatterplot of the number of speakers vs. DER per VoxConverse file

4.2 Results of the optimization of PyAnnote

4.2.1 Hyperparameter tuning

The first optimization applied to the PyAnnote pipeline involves hyperparameter tuning within the pipeline itself. The outcomes of the hyperparameter tuning process on the VoxConverse dataset are presented in Table 4.1. The top 3 parameter settings, in terms of DER, are displayed on top, as well as the worst 3 at the bottom. The default setting of the parameters is also displayed as reference. By increasing the segmentation threshold to 0.60 and the clustering threshold to 0.75, and changing the clustering method, the DER improves by 0.014 compared to the default settings. It is also remarkable that the top 3 configurations all use "average" linkage as clustering method, while the default version and the bottom 3 all use "centroid" linkage. The results of all 86 configurations can be found in Appendix A.

Table 4.1: Adjusted hyperparameter settings on VoxConverse. The top and bottom 3 settings in terms of DER are displayed, as well as the hyperparameters from the pretrained pipeline

Ranking	*Method*	*Segmentation threshold*	*Clustering threshold*	*DER*
1	**average**	**0.60**	**0.75**	**0.111**
2	average	0.60	0.70	0.112
3	average	0.50	0.70	0.113
...	...	...	...	...
23 (=def)	centroid	0.444	0.715	0.125
...	...	...	...	...
84	centroid	0.50	0.50	0.434
85	centroid	0.45	0.50	0.434
86	centroid	0.30	0.50	0.443

Table 4.2 displays the same information for the NFI-FRITS dataset. Compared to the results from VoxConverse, the tuning of these hyperparameters achieves more DER improvement, since the difference between the default configuration and the best configuration is 0.09. This is achieved by increasing the segmentation threshold, decreasing the clustering threshold and changing the clustering method. Again, "average" linkage is the clustering method for the top 3 configurations. The total table can be found in

Appendix B.

Table 4.2: Adjusted hyperparameter settings on NFI-FRITS. The top and bottom 3 configurations in terms of DER are displayed, as well as the hyperparameters from the pretrained pipeline

Ranking	*Method*	*Segmentation threshold*	*Clustering threshold*	*DER*
1	**average**	**0.60**	**0.65**	**0.342**
2	average	0.60	0.70	0.345
3	average	0.55	0.70	0.354
...	...	...	...	...
38 (=def)	centroid	0.444	0.715	0.432
...	...	...	...	...
84	average	0.30	0.5	0.524
85	centroid	0.30	0.55	0.531
86	centroid	0.30	0.5	0.533

4.2.2 Passing a *max_speaker* argument

The other approach to improve the performance of PyAnnote, is passing a *min_speakers* and *max_speakers* argument to the pretrained pipeline. With the *min_speakers* set to 1, the values of *max_speakers* are adjusted. The results are presented in Table 4.3. For the VoxConverse dataset, passing a *max_-speakers* argument increases the average DER. The DER for NFI-FRITS is decreasing when adding a *max_speakers* argument higher than 1; the lowest DER is reached when setting *max_speakers* to 2.

Table 4.3: DER for PyAnnote pretrained pipeline with different *max_speakers* values

	DER	
max_speakers	*VoxConverse*	*NFI-FRITS*
None (=def)	**0.125**	0.432
1	0.476	0.471
2	0.361	**0.359**
5	0.238	0.373
10	0.153	0.389
15	0.150	0.393
20	0.153	0.393

Inspecting the influence of setting a *max_speakers* parameter in the pretrained pipeline involves comparing the average DER for files with fewer or the same number of speakers as the parameter value with the average DER for files with more speakers. Table 4.4 shows that the DER lies considerable higher when the number of speakers exceeds the *max_speakers* value.

Table 4.4: DER of files with less speakers than *max_speakers* compared to DER of files with more speakers than *max_speakers* in VoxConverse

	DER for files with num_speakers	
max_speakers	$\leq$ *max_speakers*	$>$ *max_speakers*
1	0.025	0.507
2	0.096	0.427
5	0.131	0.317
10	0.136	0.243
15	0.147	0.279
20	0.152	0.326

As a last step, the influence of setting the *max_speakers* parameter on files with only 1 speaker is evaluated. As Table 4.5 shows, the DER improves by 0.06 when setting the *max_speakers* to 1 instead of *None*. The other values also improve the DER of the files with one speaker compared to not passing a parameter.

Table 4.5: DER for single-speaker VoxConverse files with different *max_speakers* values

max_speakers	*DER*
None (=def)	0.084
1	**0.025**
2	0.052
5	0.069
10	0.069
15	0.069
20	0.069

4.3 Results of the optimization of NeMo

4.3.1 Clustering diarizer: hyperparameter tuning

Multiple adjustments to the NeMo pipeline are implemented to enhance its performance. Table 4.6 reports on the effect of passing different *max_num_-speakers* values to the clustering diarizer. For the VoxConverse dataset, employing a value of 10 or higher yields a lower DER compared to not specifying the *max_num_speakers* argument. The optimal performance, achieving a DER of 0.105, is obtained when setting *max_num_speakers* to 20. Conversely, the performance on NFI-FRITS dataset cannot be enhanced. When setting the argument to 1 or 2, the DER deteriorates, and higher values produce a DER equivalent to that obtained with the default setting of 8.

Table 4.6: DER for NeMo clustering diarizer with different *max_num_speakers* values

max_num_speakers	*VoxConverse*	*NFI-FRITS*
1	0.502	0.469
2	0.340	0.189
5	0.185	**0.184**
8 (=def)	0.118	**0.184**
10	0.114	**0.184**
15	0.107	**0.184**
20	**0.105**	**0.184**

Next, different configuration files are passed to the clustering diarizer. Table 4.7 shows that the "*meeting*" configuration performs better for VoxConverse, reaching a DER of 0.1, while NFI-FRITS's performance remains better when using the default setting "*telephonic*".

Table 4.7: DER for NeMo clustering diarizer with different domain types

Domain type	*VoxConverse*	*NFI-FRITS*
telephonic (=def)	0.118	**0.184**
meeting	**0.100**	0.195

4.3.2 Neural diarizer: hyperparameter tuning

Lastly, the neural diarizer is introduced. The results for the different values of the sigmoid threshold are presented in Table 4.8. For both datasets, an improvement in DER is observed when the sigmoid threshold is increased. For VoxConverse, there is only a slight decrease of 0.003 in DER, and no improvement compared to the default value of 0.70. The DER of the NFI-FRITS dataset, on the other hand, decreases with 0.09 compared to the default setting.

Table 4.8: DER for NeMo neural diarizer with different sigmoid thresholds

Sigmoid threshold	*VoxConverse*	*NFI-FRITS*
0.45	0.138	0.237
0.50	0.137	0.229
0.55	0.136	0.222
0.60	0.136	0.217
0.65	**0.135**	0.213
0.70 (=def)	**0.135**	0.209
0.75	**0.135**	0.205
0.80	**0.135**	0.203
0.85	**0.135**	0.201
0.90	**0.135**	**0.200**
0.95	**0.135**	**0.200**

4.4 Results of the best models comparison

After the different experiments with the optimization of PyAnnote and NeMo, the best models are selected per dataset. The corresponding DER is displayed in Figure 4.3, compared to the baseline DER. Only the NeMo model could not be improved on the NFI-FRITS dataset. The other models could be improved slightly; the performance of PyAnnote on NFI-FRITS improved significantly, with an 0.09 decrease in DER. Still, NeMo outperforms PyAnnote on the NFI-FRITS dataset, as well as on the VoxConverse dataset.

At last, the Real-Time Factor and the memory usage of the models is compared in Table 4.9. The values of the RTF show that PyAnnote's pipeline runs twice as fast as the clustering diarizer of NeMo on both datasets, and

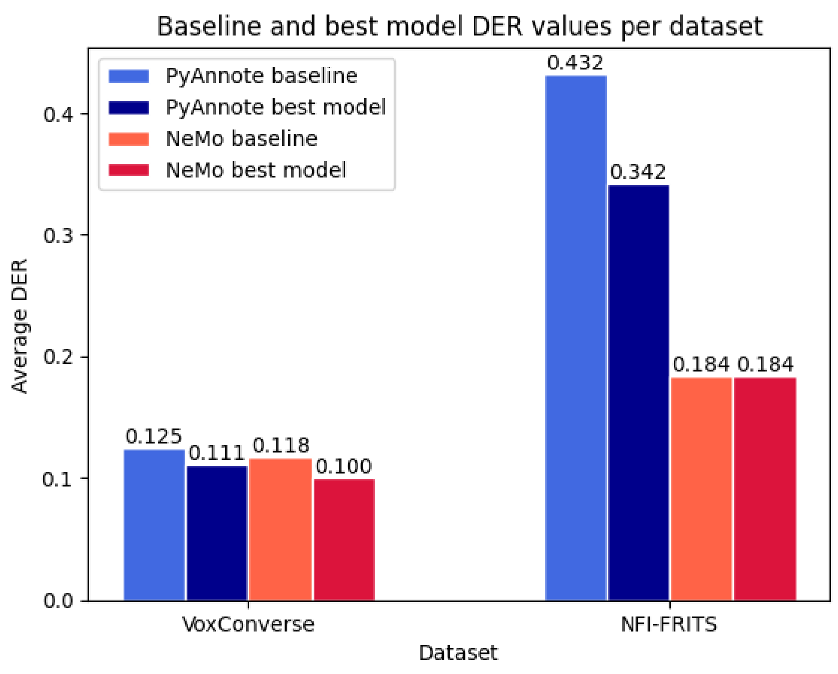


Figure 4.3: Results for the baseline models and best models of PyAnnote and NeMo on both datasets

even five times faster than the neural diarizer. Furthermore, the memory allocation of PyAnnote is significantly lower compared to that of NeMo.

Table 4.9: RTF and memory comparison for the best models of PyAnnote (pre-trained pipeline) and NeMo (clustering or neural diarizer)

Dataset	*Model*	*Method*	*DER*	*RTF*	*Memory peak*
Vox Converse	PyAnnote	Pretrained pipeline	0.111	**0.010**	**116 MB**
	NeMo	Clustering	**0.100**	0.020	798 MB
		Neural	0.135	0.103	852 MB
NFI-FRITS	PyAnnote	Pretrained pipeline	0.342	**0.011**	**20.6 MB**
	NeMo	Clustering	**0.184**	0.023	191 MB
		Neural	0.200	0.137	278 MB

5. Conclusion

In this thesis, the performance of the speaker diarization pipelines from PyAnnote and NeMo on the VoxConverse and NFI-FRITS test sets are explored. This chapter discusses the results presented in Chapter 4.

5.1 Baseline comparison

The evaluation begins with a comparative analysis of the two datasets, since Figure 4.1 shows quite different DER scores per dataset. VoxConverse is a dataset comprising YouTube fragments from the conversational domain, like panel debates and discussions. It can be argued that these audio files must meet a certain level of quality to be published online and offer value to listeners. Furthermore, speakers in these audio files would make an effort to ensure their speech is intelligible. The NFI-FRITS dataset, on the other hand, consists of intercepted telephone calls with speakers that are unaware of being recorded, and consequently, do not pay attention to being audible or articulating their speech. These dataset characteristics potentially contribute to the observed variations in the results, as evidenced by the average DER of approximately 0.120 for VoxConverse and around 0.300 for NFI-FRITS across both models.

Next, the performance of two baseline implementations of the models is inspected. The results from Figure 4.1 indicate that both PyAnnote and NeMo exhibit comparable performance on VoxConverse, while on the NFI-FRITS dataset, NeMo demonstrates significantly superior performance in terms of DER, compared to PyAnnote. This discrepancy in performance can be attributed to the fact that PyAnnote is trained on less noisy data from different domains, which may lead to difficulties in accurately processing low-quality recordings such as the intercepted phone calls from the NFI-FRITS dataset. Conversely, NeMo exhibits better robustness in handling noisy audio conditions.

Figure 4.2 depicts the relationship between number of speakers and DER per VoxConverse audio file. The slope of the regression line indicates that there is no correlation between the number of speakers and the DER per file. Therefore, in the upcoming sections of the discussion, the possibility that a higher number of speakers per audio file leads to a higher DER can be dismissed.

5.2 Optimization of PyAnnote

5.2.1 Hyperparameter tuning

The process of tuning the hyperparameters of the pretrained PyAnnote pipeline demonstrates superior performance compared to the default hyperparameters. For the VoxConverse dataset, it requires the increase of both the segmentation threshold and the clustering threshold, as well as the change of the clustering method, to improve the DER. By raising the segmentation threshold, the system becomes more robust to slight changes. This adjustment proves to be more advantageous for the VoxConverse dataset, which exhibits a relatively low percentage of overlapping speech (around 3%) and contains quite some background noise that shouldn't be detected as speech. These are also characteristics that require a higher clustering threshold, since segments that are detected as speech but are not speech, should not be assigned to a new speaker label. At the same time, it can be argued that the high number of speakers in the VoxConverse dataset requires a lower clustering threshold, to make sure that every speaker is detected. The second and third best hyperparameter setting from the experiment displayed in Table 4.1 do exhibit a lower clustering value compared to the default values.

The best parameter settings for the NFI-FRITS dataset are a higher segmentation threshold, a lower clustering threshold and the "average" clustering method. The segmentation threshold is higher because there is possibly not a lot of overlapping speech in the two-speaker telephone conversations.

On the other hand, the dataset consists of low-quality audio, which would actually require a lower segmentation threshold. Moreover, a lower clustering threshold is suitable for the NFI-FRITS dataset, given that the number of speakers is always 2 and the limited presence of background noise. This ensures the appropriate division of the low-quality audio between the two speakers. Overall, tuning the hyperparameters for NFI-FRITS causes more improvement in DER compared to the VoxConverse dataset. This can be attributed to the specific characteristics of NFI-FRITS, which enables more precise hyperparameter tuning.

For both datasets, the linkage method "average" outperforms the default method "centroid". Apparently, the speech segments in audio files exhibit dissimilarity within speaker clusters. With average linkage, these segments can still be grouped together, while centroid linkage performs better when the speaker clusters are well-separated. Therefore, changing the clustering method improves the DER for both datasets.

5.2.2 Passing a *max_speaker* argument

In the experiments conducted on PyAnnote, the investigation of setting a *max_speaker* parameter has shown promise in improving the DER for a domain-specific dataset like NFI-FRITS. Since the maximum number of speakers influences the maximum number of clusters directly, passing this argument prevents the model from searching for more speakers. Therefore, the best performance for NFI-FRITS with the *max_speaker* argument set to 2 is expected, since this dataset only contains audio files with 2 speakers.

At the same time, the best performance for VoxConverse requires not passing a *max_speaker* argument. This result is surprising, since it is assumed that setting a limit for the number of speakers will be beneficial for all audio files with up to 20 speakers. There is only one audio file that exceeds the limit, containing 21 speakers, which can not account for the difference of 0.028 in DER between the *max_speaker = None* and *max_speaker = 20.*

However, it should be noted that exceeding the *max_speaker* value results in a significant increase in DER, as shown in Table 4.4. By specifying a *max_-*

speaker argument, the model restricts its search for speakers to the provided value. Consequently, every additional speaker in an audio file will contribute to the Speaker Confusion component of the Diarization Error Rate, as explained in Section 1.2.1.

The audio files from VoxConverse with solely one speaker also benefitted from setting a *max_speaker* value. This indicates that, for a low number of speakers, setting a boundary of some sort limits the model in the search for speakers and improves its performance. Splitting audio files on the possible number of speakers and applying the right *max_speaker* parameter accordingly can decrease the DER of the total dataset even further.

5.3 Optimization of NeMo

5.3.1 Clustering diarizer: hyperparameter tuning

In the context of the optimization of the NeMo pipeline, some improvements have been achieved, although they are minimal. The results of NeMo's *max_num_speakers* argument are displayed in Table 4.6 and it achieves quite different results compared to the PyAnnote *max_speaker* argument. Since the default setting is 8 instead of *None*, it is no surprise that a value of 20 reaches the best DER for VoxConverse, considering the diverse range of number of speakers in this dataset. For NFI-FRITS, it was expected that a *max_num_speakers* value of 2 would yield the best results. However, the values from 5 to 20 all surpass the DER of 0.189 generated by *max_num_speakers* = 2. Although the effect of passing an argument is the same for NeMo as for PyAnnote, i.e. restricting the number of clusters, it seems that the value for the maximum number of speakers restricts the model more than for PyAnnote. This explains the slightly better performance of higher *max_num_speakers* values for NFI-FRITS.

Consistent with the expectations, utilizing the configurations from the default domain type "*telephonic*" yields better performance for the intercepted phone conversations in the NFI-FRITS dataset, while the "*meeting*" domain type demonstrates superior results for the VoxConverse dataset. The most significant difference between the two configurations lies in the onset and offset parameters of the voice activity detection module. In the telephonic configuration, both values are set lower compared to the meeting configuration, indicating a more inclusive detection of speech segments. Given the presence of low-quality audio in the NFI-FRITS dataset, these lower onset and offset values are necessary to capture speech correctly. On the other hand, the VoxConverse dataset contains substantial background noise that should not be labeled as speech. Therefore, the meeting configuration with higher onset and offset values proves beneficial in this scenario.

5.3.2 Neural diarizer: hyperparameter tuning

The introduction of the neural diarizer could not improve the DER of both the VoxConverse and the NFI-FRITS dataset compared to the clustering diarizer. Nevertheless, it is worth mentioning that increasing the sigmoid threshold above the default value of 0.7 resulted in a higher DER for the NFI-FRITS dataset. This indicates that with the default value, not all overlapping speech segments were labeled accordingly. Since the audio files are telephone conversations with only 2 speakers, there is not a lot of overlap expected. Therefore, the decrease in DER is minimal.

5.4 Comparison of the best models

In general, NeMo performs better than PyAnnote in terms of DER, but worse when comparing the RTF and the memory usage. This trade-off can be attributed to the multiscale approach that NeMo uses during the clustering process, as explained in Section 1.3.2. Using this approach, there is more information to process and store during execution, which requires more resources. At the same time, the results will be more accurate, resulting in a lower DER. Compared to the clustering diarizer, the neural diarizer is even more inefficient with running time and memory allocation. This can be ascribed to the initiation of the neural network and the additional storing of information. This volume of memory allocation does not apply to PyAnnote, resulting in its remarkably low memory usage. When examining the complete pipelines, it becomes apparent that NeMo's pipeline utilizes larger models in contrast to PyAnnote, resulting in an increased memory usage.

5.5 Answering the research question

This thesis aimed to answer the following research question:

> *"How can the ASR model Whisper be augmented with a speaker diarization method that achieves a low Diarization Error Rate on Dutch audio files containing overlapping speech and multiple speakers, while maintaining reasonable speed in terms of Real-Time Factor?"*

The results of the research presented in the previous sections demonstrate that both PyAnnote and Nemo are suitable options as speaker diarization method alongside Whisper, since their output of RTTM format annotations can easily be merged with the Whisper output. Moreover, the choice between the two methods depends on the specific requirements and priorities. If prioritizing a lower Diarization Error Rate, NeMo proves to be more suitable. On the other hand, for those seeking a faster implementation with lower memory usage, PyAnnote emerges as the optimal choice. NeMo can be considered a preferable choice when hyperparameter tuning cannot or is not desirable (e.g., when the type of data is unknown beforehand), as it demonstrates strong performance out-of-the-box. The optimizations to the pipelines as presented above can be applied simultaneously, with even more specific values, or extended, to improve the performance of the pipelines even further.

6. Discussion

The experiments in this research provide meaningful insights and useful results. Nevertheless, there are limitations associated with the research that need to be addressed. Futhermore, this section provides suggestions for future work.

6.1 Limitations

This research uses two datasets: a subset of the publicly available VoxConverse dataset and a subset of the NFI-FRITS dataset, which has limited access as a result of its sensitive nature. While the VoxConverse dataset serves as a valuable addition to the domain-specific NFI-FRITS dataset, due to the limited number of speakers in the latter, its suitability for representing police data is questionable. This uncertainty arises from the somewhat staged nature of the audio content in VoxConverse, which does not fully capture the characteristics of the data the police deal with. The NFI-FRITS dataset can be considered representative of one subset of audio files encountered by the police, but its duration is relatively short, spanning only 11 hours, and containing only 2 speakers. Additionally, the police encounters languages other than Dutch. Hence, it is questionable whether the results of this research are also applicable to other languages. As a result, the generalizability of the research findings is somewhat limited, and further experiments on larger and more diverse audio corpora are necessary.

Furthermore, an important data limitation relates to ethical considerations, as individuals in the NFI-FRITS dataset are unaware of their inclusion. When expanding the dataset with audio that includes more real-life police audio, careful consideration of the desirability and ethical implications is necessary to protect privacy and obtain consent of all individuals involved. Additionally, it is important to note that the NFI-FRITS dataset is not openly accessible, restricting the reproducibility of the experiments presented in this thesis to a wider audience.

A methodological limitation includes the calculation of the average Diarization Error Rate (DER). When averaging the DER of audio files, every DER gets the same weight, independent of the length of the corresponding audio file. It can be argued that for audio files with the same DER score but different length, longer audio files should be considered more important, since they contain more correctly assigned speech time compared to the shorter audio file. This would result in the use of an weighted average instead of a "normal" average.

At last, it is important to acknowledge that the monitoring of running time and memory usage was conducted concurrently with other experiments. Although the metrics were averaged over multiple experiment runs with varying hyperparameters, they are not completely representative. Moreover, due to resource and time constraints, the experiments could not be repeated multiple times with identical configurations. As a result, slight variations in the DER may arise when replicating the experiments.

6.2 Future work

This thesis aimed to conduct experiments on a limited amount of data. Especially the NFI-FRITS subset, which is very domain-specific and useful as test set for the police, is quite small in terms of hours of audio. For further research, an enlarged domain-specific dataset should be compiled, varying in the number of speakers and languages. This would enable the tuning of the hyperparameters more specifically, yielding better performance results. The choice to optimize specific parameters was driven by the anticipation of achieving the most significant performance enhancement. However, it should be noted that there are additional parameters that have the potential to contribute to performance improvement. In addition to fine-tuning hyperparameters, there is potential benefit in combining multiple optimizations discussed in this research into a single optimization approach.

Considering the superior performance of the NeMo model in terms of DER, it can be valuable to focus on minimizing its resource usage. Exploring strategies to eliminate unnecessary steps or reduce storage requirements could significantly enhance its speed and optimize memory usage.

This thesis only focuses on the speaker diarization pipelines of PyAnnote and NeMo, while there are more promising implementations available. Therefore, future research can explore alternative methods.

A recommendation for the Dutch National Police would be to gain deeper insights into the number of speakers in general and per audio file type. This would facilitate further fine-tuning of the pipelines and the possibility of adapting a combination of models for domain-specific data. Given the prevalence of single-speaker audio files in police data, it is worth investigating whether these files would specifically benefit from separate configurations.

7. Acknowledgements

I am grateful of all the lovely people that I have met during this project. Especially, I would like to thank the following people:

- my supervisors from Utrecht University, Duygu Islakoglu and Vahid Sharivari, for their continuous support. Their weekly check-ins and constructive feedback have significantly contributed to the refinement of my research.
- Nils Hulzebosch, my supervisor from the Dutch National Police, who offered so much help, made us feel at home at the police office and made the last weeks a lot more fun
- my first examiner Dr. I.R. (Ioana) Karnstedt-Hulpus, for taking the time to delve into the field of speech-to-text processing, attend our presentation and read my complete thesis
- Isa Dielen, who I could share all the ups and downs of this project with. Her friendship and support has been incredibly valuable

Besides, I would like to thank all my friends and family, and especially Aafje and Pieter. They flooded me with motivating words when it was much needed.

Bibliography

[1] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, “Robust speech recognition via large-scale weak supervision,” *arXiv preprint arXiv:2212.04356*, 2022.

[2] S. Tranter and D. Reynolds, “An overview of automatic speaker diarization systems,” *IEEE Transactions on Audio, Speech and Language Processing*, vol. 14, no. 5, pp. 1557–1565, Sep. 2006. DOI: `10.1109/tasl.2006.878256`. [Online]. Available: `https://doi.org/10.1109/tasl.2006.878256`.

[3] T. J. Park, N. Kanda, D. Dimitriadis, K. J. Han, S. Watanabe, and S. Narayanan, “A review of speaker diarization: Recent advances with deep learning,” *Computer Speech &amp Language*, vol. 72, p. 101 317, Mar. 2022. DOI: `10.1016/j.csl.2021.101317`. [Online]. Available: `https://doi.org/10.1016/j.csl.2021.101317`.

[4] S. Whibley, M. Day, P. May, and M. Pennock, “Wav format preservation assessment,” *British Library: London, UK*, 2016.

[5] J. Huh, A. Brown, J.-w. Jung, *et al.*, “Voxsrc 2022: The fourth voxceleb speaker recognition challenge,” *arXiv preprint arXiv:2302.10248*, 2023.

[6] J. G. Fiscus, J. Ajot, M. Michel, and J. S. Garofolo, “The rich transcription 2006 spring meeting recognition evaluation,” in *Machine Learning for Multimodal Interaction: Third International Workshop, MLMI 2006, Bethesda, MD, USA, May 1-4, 2006, Revised Selected Papers 3*, Springer, 2006, pp. 309–322.

[7] M. Malik, M. K. Malik, K. Mehmood, and I. Makhdoom, “Automatic speech recognition: A survey,” *Multimedia Tools and Applications*, vol. 80, pp. 9411–9457, 2021.

[8] G. Heigold, I. Moreno, S. Bengio, and N. Shazeer, “End-to-end text-dependent speaker verification,” in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 5115–5119. DOI: `10.1109/ICASSP.2016.7472652`.

[9] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur, “X-vectors: Robust dnn embeddings for speaker recognition,” in *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, IEEE, 2018, pp. 5329–5333.

[10] A. Zhang, Q. Wang, Z. Zhu, J. Paisley, and C. Wang, “Fully supervised speaker diarization,” in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2019, pp. 6301–6305.

[11] R. Yin, H. Bredin, and C. Barras, “Neural speech turn segmentation and affinity propagation for speaker diarization,” in *Annual Conference of the International Speech Communication Association*, 2018.

[12] 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 *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2019, pp. 91–95.

[13] N. Kanda, Y. Gaur, X. Wang, *et al.*, "Joint speaker counting, speech recognition, and speaker identification for overlapped speech of any number of speakers," *arXiv preprint arXiv:2006.10930*, 2020.

[14] N. Kanda, X. Chang, Y. Gaur, *et al.*, "Investigation of end-to-end speaker-attributed asr for continuous multi-talker recordings," in *2021 IEEE Spoken Language Technology Workshop (SLT)*, IEEE, 2021, pp. 809–816.

[15] H. Bredin, R. Yin, J. M. Coria, *et al.*, "Pyannote. audio: Neural building blocks for speaker diarization," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2020, pp. 7124–7128.

[16] M. Ravanelli and Y. Bengio, "Speaker recognition from raw waveform with sincnet," in *2018 IEEE Spoken Language Technology Workshop (SLT)*, IEEE, 2018, pp. 1021–1028.

[17] H. Bredin, "Pyannote. audio 2.1 speaker diarization pipeline: Principle, benchmark, and recipe,"

[18] E. Harper, S. Majumdar, O. Kuchaiev, *et al.*, *NeMo: a toolkit for Conversational AI and Large Language Models*. [Online]. Available: `https://github.com/NVIDIA/NeMo`.

[19] F. Jia, S. Majumdar, and B. Ginsburg, "Marblenet: Deep 1d time-channel separable convolutional neural network for voice activity detection," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2021, pp. 6818–6822.

[20] N. R. Koluguri, T. Park, and B. Ginsburg, "Titanet: Neural model for speaker representation with 1d depth-wise separable convolutions and global context," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 8102–8106. DOI: `10.1109/ICASSP43922.2022.9746806`.

[21] T. J. Park, N. R. Koluguri, J. Balam, and B. Ginsburg, "Multi-scale speaker diarization with dynamic scale weighting," *arXiv preprint arXiv:2203.15974*, 2022.

[22] D. v. Vloed, J. Bouten, and D. A. van Leeuwen, "Nfi-frits: A forensic speaker recognition database and some first experiments," 2014.

[23] J. S. Chung, J. Huh, A. Nagrani, T. Afouras, and A. Zisserman, "Spot the conversation: Speaker diarisation in the wild," *arXiv preprint arXiv:2007.01216*, 2020.

Appendix A

Table 7.1: PyAnnote with adjusted hyperparameters on NFI-FRITS. The 3 configurations with the lowest DER are marked in green, the 3 highest are marked in red and the default values are marked in yellow. A lower DER is better.

Begin of Table

Method	Segmentation threshold	Clustering threshold	DER
average	0.30	0.50	0.524
average	0.30	0.55	0.487
average	0.30	0.60	0.438
average	0.30	0.65	0.417
average	0.30	0.70	0.423
average	0.30	0.75	0.437
average	0.35	0.50	0.499
average	0.35	0.55	0.455
average	0.35	0.60	0.415
average	0.35	0.65	0.394
average	0.35	0.70	0.394
average	0.35	0.75	0.418
average	0.40	0.50	0.485
average	0.40	0.55	0.441
average	0.40	0.60	0.399
average	0.40	0.65	0.375
average	0.40	0.70	0.377
average	0.40	0.75	0.396
average	0.45	0.50	0.471
average	0.45	0.55	0.421
average	0.45	0.60	0.386
average	0.45	0.65	0.369
average	0.45	0.70	0.362
average	0.45	0.75	0.386
average	0.50	0.50	0.461

Continuation of Table 7.1

Method	Segmentation threshold	Clustering threshold	DER
average	0.50	0.55	0.414
average	0.50	0.60	0.372
average	0.50	0.65	0.355
average	0.50	0.70	0.354
average	0.50	0.75	0.382
average	0.55	0.50	0.452
average	0.55	0.55	0.404
average	0.55	0.60	0.365
average	0.55	0.65	0.348
average	0.55	0.70	0.353
average	0.55	0.75	0.377
average	0.60	0.50	0.448
average	0.60	0.55	0.398
average	0.60	0.60	0.357
average	0.60	0.65	0.342
average	0.60	0.70	0.345
average	0.60	0.75	0.375
centroid	0.30	0.50	0.533
centroid	0.30	0.55	0.531
centroid	0.30	0.60	0.509
centroid	0.30	0.65	0.477
centroid	0.30	0.70	0.460
centroid	0.30	0.75	0.492
centroid	0.35	0.50	0.521
centroid	0.35	0.55	0.519
centroid	0.35	0.60	0.498
centroid	0.35	0.65	0.466
centroid	0.35	0.70	0.441
centroid	0.35	0.75	0.476
centroid	0.40	0.50	0.513
centroid	0.40	0.55	0.510
centroid	0.40	0.60	0.489
centroid	0.40	0.65	0.454

Continuation of Table 7.1

Method	Segmentation threshold	Clustering threshold	DER
centroid	0.40	0.70	0.431
centroid	0.40	0.75	0.464
centroid	0.444	0.715	0.432
centroid	0.45	0.50	0.506
centroid	0.45	0.55	0.502
centroid	0.45	0.60	0.481
centroid	0.45	0.65	0.446
centroid	0.45	0.70	0.420
centroid	0.45	0.75	0.455
centroid	0.50	0.50	0.498
centroid	0.50	0.55	0.494
centroid	0.50	0.60	0.475
centroid	0.50	0.65	0.441
centroid	0.50	0.70	0.418
centroid	0.50	0.75	0.442
centroid	0.55	0.50	0.494
centroid	0.55	0.55	0.490
centroid	0.55	0.60	0.465
centroid	0.55	0.65	0.435
centroid	0.55	0.70	0.414
centroid	0.55	0.75	0.440
centroid	0.60	0.50	0.491
centroid	0.60	0.55	0.483
centroid	0.60	0.60	0.458
centroid	0.60	0.65	0.428
centroid	0.60	0.70	0.409
centroid	0.60	0.75	0.438

Appendix B

Table 7.2: PyAnnote with adjusted hyperparameters on VoxConverse. The . 3 configurations with the lowest DER are marked in green, the 3 highest are marked in red and the default values are marked in yellow. A lower DER is better.

Begin of Table

Method	Segmentation threshold	Clustering threshold	DER
average	0.30	0.50	0.249
average	0.30	0.55	0.186
average	0.30	0.60	0.153
average	0.30	0.65	0.137
average	0.30	0.70	0.127
average	0.30	0.75	0.126
average	0.35	0.50	0.246
average	0.35	0.55	0.181
average	0.35	0.60	0.148
average	0.35	0.65	0.132
average	0.35	0.70	0.124
average	0.35	0.75	0.120
average	0.40	0.50	0.239
average	0.40	0.55	0.178
average	0.40	0.60	0.149
average	0.40	0.65	0.129
average	0.40	0.70	0.120
average	0.40	0.75	0.118
average	0.45	0.50	0.239
average	0.45	0.55	0.174
average	0.45	0.60	0.144
average	0.45	0.65	0.125
average	0.45	0.70	0.117
average	0.45	0.75	0.115
average	0.50	0.50	0.232

Continuation of Table 7.2

Method	Segmentation threshold	Clustering threshold	DER
average	0.50	0.55	0.171
average	0.50	0.60	0.142
average	0.50	0.65	0.122
average	0.50	0.70	0.113
average	0.50	0.75	0.114
average	0.55	0.50	0.233
average	0.55	0.55	0.170
average	0.55	0.60	0.140
average	0.55	0.65	0.120
average	0.55	0.70	0.113
average	0.55	0.75	0.113
average	0.60	0.50	0.232
average	0.60	0.55	0.168
average	0.60	0.60	0.139
average	0.60	0.65	0.119
average	0.60	0.70	0.112
average	0.60	0.75	0.111
centroid	0.30	0.50	0.443
centroid	0.30	0.55	0.348
centroid	0.30	0.60	0.228
centroid	0.30	0.65	0.163
centroid	0.30	0.70	0.132
centroid	0.30	0.75	0.129
centroid	0.35	0.50	0.434
centroid	0.35	0.55	0.349
centroid	0.35	0.60	0.222
centroid	0.35	0.65	0.162
centroid	0.35	0.70	0.130
centroid	0.35	0.75	0.127
centroid	0.40	0.50	0.432
centroid	0.40	0.55	0.349
centroid	0.40	0.60	0.220
centroid	0.40	0.65	0.161

Continuation of Table 7.2

Method	Segmentation threshold	Clustering threshold	DER
centroid	0.40	0.70	0.126
centroid	0.40	0.75	0.125
centroid	0.444	0.715	0.125
centroid	0.45	0.50	0.434
centroid	0.45	0.55	0.342
centroid	0.45	0.60	0.215
centroid	0.45	0.65	0.159
centroid	0.45	0.70	0.126
centroid	0.45	0.75	0.123
centroid	0.50	0.50	0.434
centroid	0.50	0.55	0.345
centroid	0.50	0.60	0.213
centroid	0.50	0.65	0.154
centroid	0.50	0.70	0.123
centroid	0.50	0.75	0.117
centroid	0.55	0.50	0.434
centroid	0.55	0.55	0.347
centroid	0.55	0.60	0.215
centroid	0.55	0.65	0.152
centroid	0.55	0.70	0.123
centroid	0.55	0.75	0.115
centroid	0.60	0.50	0.433
centroid	0.60	0.55	0.344
centroid	0.60	0.60	0.211
centroid	0.60	0.65	0.154
centroid	0.60	0.70	0.121
centroid	0.60	0.75	0.117

End of Table