Rheinische
Friedrich-Wilhelms-
Universität Bonn

Institute for Computer Science
Department VI
Autonomous Intelligent Systems

Rheinische
Friedrich-Wilhelms-Universität Bonn

Master Thesis

# Overlap and Speaker-Turn Awareness for Low-Latency Automatic Speech Recognition

*Author:*
Lukas Storck

*First Examiner:*
Prof. Dr. Sven Behnke

*Second Examiner:*
Prof. Dr. Frank Kurth

*Supervisor:*
Bastian Pätzold

Date: April 11, 2025

# Declaration

I hereby declare that I am the sole author of this thesis and that none other than the specified sources and aids have been used. Passages and figures quoted from other works have been marked with appropriate mention of the source.

Bonn, 11th of April 2025

Place, Date

[illegible]

Signature

# Abstract

Many applications, including service robotics, require an audio perception system capable of processing single voice commands or understanding entire conversations. Such a system must be able to transcribe speech and associate speakers with minimal latency in challenging acoustic environments. Existing methods struggle to correctly transcribe overlapping speech from multiple speakers, which is common in natural conversation. We propose an online speech recognition pipeline that uses a diarization-first approach to identify and associate speakers before applying speech recognition. This formulation allows additional processing steps to be applied in between, if necessary, to handle overlapping speech segments by separating the audio sources to improve the quality of the transcription and speaker assignments. The pipeline is evaluated on single and multi-speaker datasets and analyzed in terms of transcription and speaker assignment quality, latency, and required computational resources. Our approach produces accurate diarization and transcription results for single-speaker speech. Overlapping multi-speaker speech is a challenging task, even for offline methods, but we were able to match these results with low-latency online processing, while having low memory requirements for diarization and speaker recognition.

# Contents

# 1. Introduction

Speech serves as a fundamental medium for communication among humans, offering an intuitive and accessible means of sharing information. In the field of human-robot interaction, particularly within service robotics, the ability to perceive and understand speech is crucial. This capability allows robots to understand conversations around it and respond to questions or voice commands. As such, developing robust audio perception systems that can handle real-world scenarios is essential. One example is the *Restaurant* task from the RoboCup@Home competition, wherein a robot acts as a server and must listen to a customer to take their order [1]. Another example would be a household robot that is prompted with speech commands to summarize the contents of a refrigerator and suggest recipes from the items available [2].

A typical challenge faced by audio perception systems is the *Cocktail Party Problem*. This phenomenon occurs in environments where multiple speakers converse simultaneously amidst background noise and reverberation. Humans have the ability to concentrate one speakers voice and blend out the others, thereby separating foreground from background speech, recognizing speaker turns and identifying the correct speaker. [3] The task of isolating and transcribing each speaker's speech, particularly when overlaps occur, presents a challenge for current audio perception systems.

The audio perception system must provide transcriptions of the spoken words. While other representations of speech content are generally possible, common speech recognition systems use text transcriptions as output, which allow easier comparisons between different systems and provide more interpretable results. Additionally, the system must assign speaker labels to the words and recognize previously seen speakers. In the case of overlapping speech, it should provide the words and assigned speaker labels of all overlapping parts, and not just of the more prominent speaker. Finally, the delay between when a word is spoken and when the transcription is provided must be minimal, so as not to interrupt the flow of conversation when providing answers or responding to voice commands. Humans rarely leave a pause longer than one second from the previous utterance, with an average pause length of 275 ms [4]. In other words, the system must provide the speech information with low latency.

The audio pipeline used so far in the NimbRo project [5], developed for the RoboCup@Home competition, employs voice activity detection to select speech segments and speech recognition to transcribe the speech. It assumes only a single speaker and therefore avoids dealing with overlapping speech or speaker information. There are existing methods that solve the tasks of transcription [6, 7], diarization [8, 9, 10], and speaker identification [11, 12]. There are also methods that solve diarization in combination with transcription [13] or focus on low-latency processing [13, 14]. However, most of these systems operate in non-real-time environments or fail to address overlapping speech effectively. The integration of low-latency processing with accurate speaker diarization on overlapping speech remains a gap in current research.

We propose an audio perception system with three primary goals:

1. Achieve similar performance to the NimbRo audio pipeline for single-speaker speech.

2. Add speaker information to the transcription output to distinguish between multiple speakers.

3. Improve transcription quality for overlapping speech from multiple speakers.

The developed method uses a pipeline approach that combines several foundation models for the intermediate steps. The models used are pre-trained to avoid costly training and utilize previously established methods. This has the advantage that models can be replaced as better state-of-the-art models in terms of quality and computational performance become available. It is also possible to evaluate only parts of the pipeline on their own and use intermediate results for other tasks.

Chapters 2 and 3 provide an overview of the theory and related work. Then, chapter 4 details the methodology and design of the proposed pipeline. Chapter 5 evaluates the performance of the pipeline. After that, chapter 6 discusses the results and compares them with existing methods. Finally, chapter 7 summarizes the thesis.

# 2. Theory

This chapter describes the relevant theoretical background for the preparation of this thesis, as well as the various tasks solved throughout the proposed pipeline. First, some basics about audio signals and speech are introduced as well as the tasks that are addressed in the pipeline approach. Then, foundation models used to solve these tasks and sliding windows as a solution for continuous audio data are explained. Finally, the metrics used in this work are described.

## 2.1. Audio Signals

Sound can be represented by digital audio signals when recorded by a microphone. Such a signal is a time series of *samples* representing the amplitude of a sound wave over time. It can be expressed as a function $f : \mathbb{R} \to \mathbb{R}$ which maps time to sound pressure values. The recorded signal is the superposition of all sound sources reaching the microphone, including the desired sources, in our case speech from the primary conversation, and *noise*. Noise includes direct sources such as background speech from people not speaking in the primary conversation, laughter, music that may contain vocal parts, impact or breaking sounds (e.g., a glass breaking on the floor), ventilation or wind, and other outdoor or natural sounds. There is also indirect noise due to acoustic effects of the environment, such as reverberation and echoes from the direct sources.

The first graph in Figure 2.1 depicts the time domain representation of a signal, showing how the amplitude changes over time, which can be used to analyze the average loudness over a segment of the signal and to detect silence. An audio signal can also be represented in the frequency domain, which decomposes the signal into its spectral components, showing the energy of each frequency within the signal, which is classically done with the *Fast Fourier Transform* [15]. Another approach is to use filter banks, which apply a series of bandpass filters to the signal, allowing for a more targeted analysis. For example, *Mel filter banks* [16] are commonly used to approximate human auditory perception. The *Mel scale* was originally created to map frequencies onto a linear scale of perceived equally spaced intervals. Using the Mel scale gives the filter banks more granularity in the low frequency bands,

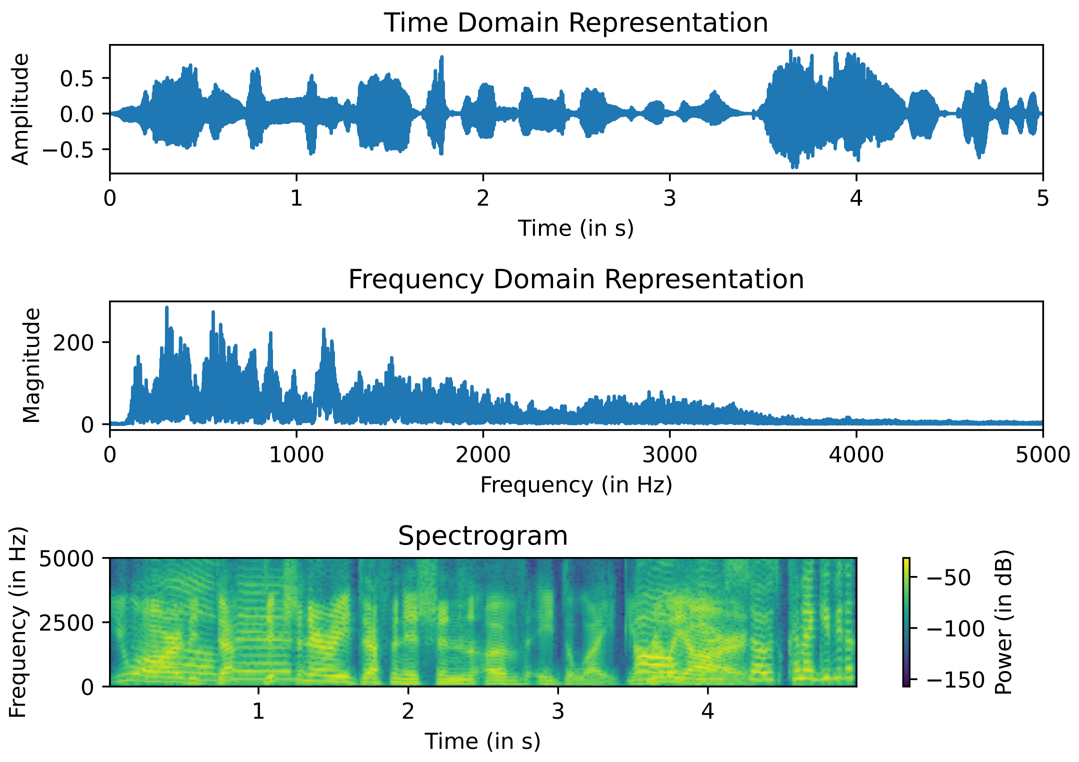

Figure 2.1: A short audio signal is displayed as a waveform in the time domain, as the magnitude of each frequency component in the frequency domain, and as a spectrogram, which combines both time and frequency information.

which also have the most energy for the human voice. The information on the energy for each frequency or frequency band can be used to detect and classify different types of sounds, or even voices, based on their different characteristics. While the second graph in Figure 2.1 shows the frequency information as a sum over the whole signal, the last graph shows a spectrogram, which is a combination of both time and frequency information, showing the energy of frequencies over time.

Differentiating between foreground and background speech can be difficult. Using loudness alone as an indicator, assuming that foreground speakers are closer to the microphone and therefore louder, can fail if the background speakers temporarily speak louder. Additional information can be provided by the acoustic data contained in the audio signal, as the environment affects sound propagation through reverberation or attenuation of frequency bands. Stereo microphones or microphone arrays are designed to capture small differences in the spatial position of sound sources, which can be used to further differentiate foreground and background speech.

A digital audio signal has a *sample rate*, the number of data points per second of audio, which is typically 8 kHz or 16 kHz for audio recordings in speech datasets. Historically, these sample rates were chosen for digital recordings based on the *Nyquist–Shannon sampling theorem* [17] to accommodate the audio bandwidth of telephones (300 – 3.4 kHz). This is sufficient to capture most of the speech information for accurate transcription. [18] While stereo audio is common in media, many speech methods and datasets can only process and provide mono audio data. In this thesis, 16 kHz mono audio data is used for all experiments and resampled and mixed as necessary.

## 2.2. Speech Processing Tasks

Next, the general speech processing tasks of the proposed pipeline are introduced and discussed.

### Speaker Diarization

The task of speaker diarization is to detect when each speaker is speaking. It is one step further than voice activity detection, since the task is to classify audio segments into speech and non-speech, but also to add speaker information. The input for speaker diarization is the audio data as a stream from a microphone or from a file. The output is a list of timestamps indicating the start and end time of speech segments with a label indicating the speaker, also called *speaker annotation*. The resulting speech segments are called *utterances*, which is an uninterrupted sequence of words spoken by one speaker. The utterances are separated by pauses and can therefore be shorter or longer than grammatical sentences, depending on the speaker's speech cadence.

Historically, the speaker diarization task has been divided into several subtasks. First, voice activity detection was used to detect speech segments regardless of the speaker. Next, potential points of speaker changes in the audio were identified. Finally, clustering was used to match speech segments with similar characteristics to recover speaker information. [19]

With *End-to-End Neural Diarization* (EEND) [20], a monolithic machine learning approach was introduced, that uses multi-label classification to solve speaker diarization. The model outputs the estimated speaker activity for a fixed number of concurrent speakers at each time step, which are called *speaker probabilities*. In this approach, the authors were able to directly minimize the diarization error due to the monolithic model and handle overlapping speech, since they did not have to assume single-speaker audio as before. A drawback is that the model provides only local diarization, as there is no method, to track speakers over longer periods of silence. Therefore, this is called *local speaker diarization*, and the diarization result that tracks speakers over the entire duration is called *global speaker diarization*.

In addition to the multi-label representation for the diarization output, there is also a powerset multi-class encoding, as depicted in Figure 2.2. Instead of a single label per speaker, with multiple labels active at the same time to denote overlapping speech, the powerset multi-class encoding uses only one active class at a given time which denote the possible combinations of the multi-label encoding. Contrary to the simplified values shown in the figure, both encodings use continuous probabilities to represent the speaker activity. Results in the powerset multi-class encoding can be converted to the multi-label encoding.

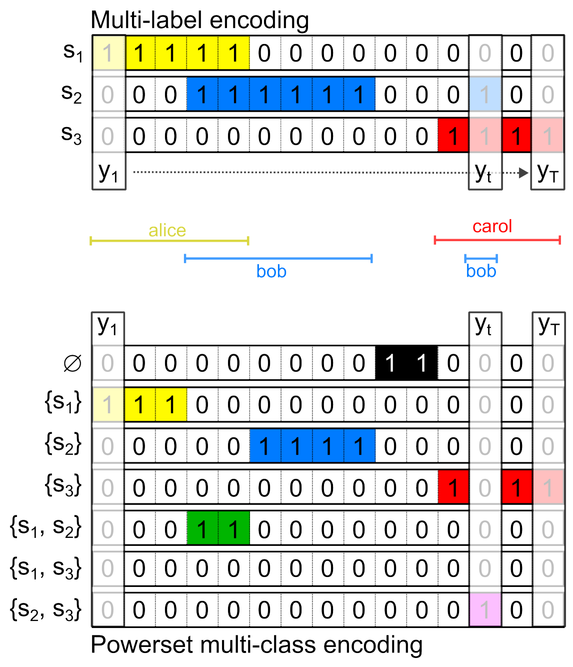

Figure 2.2: Diarization encodings: Multi-label has one class per speaker and can have multiple labels active at the same time. Powerset multi-class has only one active class which signifies either no speaker, one individual speaker, or a combination of two speakers. From [9].

## Speaker Recognition and Clustering

Speaker recognition is the task of determining who is speaking in a selected audio segment. Clustering algorithms are used to compare the identities of selected audio segments to recognize previous or identify new speakers. These identities are computed from the given audio segments containing a speaker's speech. In the past, hand-crafted features such as *Mel-Frequency Cepstral Coefficients* (MFCCs) [16] and *i-vectors* [21] were used to compute speaker embeddings. Modern systems like pyannote.audio [8] use machine learning to compute speaker embeddings from audio segments. These *speaker embeddings* represent the characteristics of a speaker's voice as high-dimensional vectors that can be compared using a distance metric. Two segments from the same speaker have a high similarity in the embedding space, so their embeddings have a small distance. Similarly, segments of different speakers have low similarity and show a high distance. The assignment to clusters is usually done by a distance threshold. The distance metric depends on how the embeddings are designed or what error metric a machine learning model was trained on, but a common metric is cosine similarity.

Diart [14] use an online incremental clustering algorithm to solve speaker diarization. Embeddings of new speech segments are compared to representations of existing speakers. These representations are calculated as the center of all embeddings associated with that speaker, which is called *centroid*. If the distance to a centroid is below a distance threshold, the segment is assigned to that speaker. It is ensured, that two overlapping speaker segments cannot be assigned to the same speaker. If the distance is above the threshold, a new speaker is added. The distance threshold is a hyperparameter that can be adjusted to tune the sensitivity of the algorithm.

DBSTREAM [22], an online adaptation of DBSCAN, is a density-based clustering algorithm for data streams. It aggregates the incoming data embeddings into *micro-clusters* in an online step and uses the density between these micro-clusters to assign *macro-cluster* labels in an offline step. New embeddings are compared to the existing micro-cluster centers, based on their distance and a distance threshold. If covering micro-clustes are found, they are nudged towards the new embedding, otherwise a new micro-cluster is initialized. The movement is controlled to avoid excessive overlap. Weak micro-clusters are periodically pruned. Macro-clusters are formed by connecting micro-clusters that have a high density in overlapping areas.

It is important to note that both algorithms cannot prevent clusters in later timesteps from moving away from embeddings seen in earlier timesteps. In some cases, this may lead to different cluster assignments in hindsight.

## Speech Recognition

Speech recognition is the task of transcribing speech into text, also known as speech-to-text. Just as with the previous tasks, the input is the audio data containing speech. The output is a *transcript*, which is a sequence of words, sometimes with timestamps for each word, depending on the system or dataset. Speech recognition approaches usually target single-speaker or non-overlapping speech.

Early approaches relied on the detection of phonemes, basic sounds of speech, based on pre-stored representations and hand-crafted rules to match these phonemes to syllables and words. These had only a limited vocabulary spanning only the words of digits or the letters of the alphabet and struggeled with pronunciation and speaker differences. [23] Modern speech recognition systems use recent machine learning advances to train on large amounts of speech data. A popular state-of-the-art system is Whisper [6]. By training on very large and diverse speech datasets, they can handle different languages, automatically detect the spoken language, distinguish speech from background noise or silence, and adapt to different accents and speaking styles. They report transcription quality for their system to be close to the level of professional human transcribers.

### Speech Separation

Speech separation is an audio enhancement technique that can isolate speakers for other methods that do not expect multiple speakers. The input is a single audio track containing a mixture of multiple speakers that may have overlapping speech. The outputs are multiple audio tracks, each track containing the audio from a single speaker, with all others removed. In other words, it provides the inverse operation of mixing multiple audio tracks. Typically, these systems are trained to output a fixed number of speakers, which means that the number of speakers in the input mix must match the expected number of output tracks of the system. Otherwise, an output track may contain multiple speakers, or a speaker's speech may be split across multiple tracks. In practice, the number of speakers in an audio segment must be known before the correct speech separation system can be selected and applied.

## 2.3. Machine Learning Methods

Recently published systems for solving the above tasks use machine learning methods. In machine learning, methods are trained on large datasets to learn the solution to a specific task, rather than being directly programmed. The methods used in this context are neural networks, consisting of layers of artificial *neurons* that are connected between the layers. Each neuron processes input values, applies a mathematical transformation, and passes the output to the next layer. These networks learn through *backpropagation*, where errors from predictions are propagated backwards to adjust the model's internal parameters, improving performance over time.

There are many different architectures for neural networks commonly used in speech processing. In speech processing, the first step is to extract features from the audio data. This can also be done using a neural network, like the *Sinc-Net* [24], which processes the time-domain signal directly. The SincNet consists of parameterized *sinc functions* as filters, allowing the network to learn optimal filter parameters during training. It therefore acts similar to filter banks, but with the advantage to adjust to the most interesting acoustic characteristics depending on the task.

*Deep Neural Networks* (DNNs) are feed-forward networks with multiple hidden layers between input and output layers, enabling them to learn hierarchical representations of data. Each additional layer allows the network to model increasingly complex and abstract patterns in the data. While powerful for various speech tasks, standard DNNs lack the ability to model temporal dependencies inherent in speech signals.

*Time Delay Neural Networks* (TDNNs) address this limitation by incorporating time-shifted copies of the input features. TDNNs process multiple frames simultaneously by delaying input features across different time steps, allowing the network to capture temporal patterns and acoustic context. This architecture is particularly effective for detecting acoustic events regardless of their position in time, making them well-suited for speaker-specific vocal characteristics.

Another architecture is the *Long Short-Term Memory* (LSTM) network, which is a type of *recurrent neural network* that captures dependencies over time. Each LSTM cell consists of an input gate, a forget gate, and an output gate, each with learned parameters. These gates control the flow of information, allowing the network to retain relevant information and discard irrelevant data over long sequences. *Bidirectional LSTMs* (Bi-LSTM) are used to capture dependencies in both forward and backward directions, so that the model has access to both past and future audio data at a given time step. A Bi-LSTM cannot be used directly for streaming data because in an online setting only past data is available at any given time, not future data. Only a simple forward LSTM can be used in real time, or the input to a Bi-LSTM must be handled in chunks with no context between chunks. Then additional post-processing is required to mitigate the loss of temporal dependencies between chunks.

*Transformer* architectures have recently revolutionized speech processing with their attention mechanisms. Unlike recurrent networks, Transformers process entire sequences in parallel through self-attention, which weighs the importance of different parts of the input relative to each other. This allows the model to focus on relevant parts of the speech signal regardless of position, capturing long-range dependencies more effectively than recurrent architectures.

*Encoder-Decoder Transformers*, like for example Whisper, combine two Transformer blocks for sequence-to-sequence tasks such as speech recognition. The encoder processes the input speech features, creating contextual representations, while the decoder generates the output text one token at a time. This architecture incorporates cross-attention mechanisms, allowing the decoder to focus on relevant parts of the encoded speech when generating each output token. While models like Whisper achieve state-of-the-art results, they require significant computational resources and face challenges in low-latency, real-time applications due to their non-causal attention mechanisms.

## 2.4. Sliding Windows

The machine learning methods used for speech processing expect blocks of audio data as input and not continuous data streams or the entire data of large files. This has computational reasons, as large files and their processing data may not fit into memory, but also represents a fundamental design choice that enables the non-causal processing capabilities of modern architectures like Bi-LSTMs and Transformers. As a result, the input audio stream must be split into blocks. To avoid information loss at the block boundaries, overlapping windows are used in a *sliding window approach.* For this thesis, the duration of a window is called *window size* and the interval between consecutive windows is called *step size.* The step size must be smaller than the window size to create an overlap between consecutive windows.

## 2.5. Metrics

There are several *quality* metrics for speaker diarization and speech recognition. They are calculated by comparing either the diarization or the transcription of a ground truth reference with the hypothesis output of the evaluated system. Additionally, there are metrics that measure the computational *performance* of these systems or the integrated pipeline as a whole. The following metrics are used during the evaluation of our proposed pipeline in chapter 5.

The *Diarization Error Rate* (DER) [25] is a basic metric used to measure the quality of the diarization result. It is based on the duration of speech segments that were falsely detected, missed, or assigned to the wrong speaker, as shown in Figure 2.3. The error rate is the sum of these three categories divided by the total

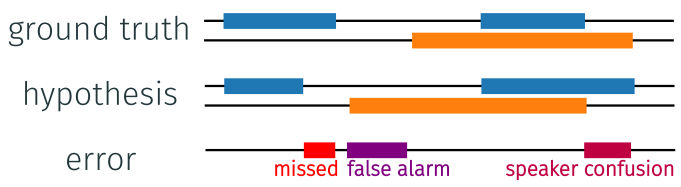

Figure 2.3: A graphical representation of the components of the *Diarization Error Rate* (DER). The ground truth and diarization output are compared based on the duartion of speech segments. The durations of falsely detected, missed speech, and speaker confusion are compared to the total duration of the reference speech.

duration of the reference speech.

$$\text{DER} = \frac{\text{False Alarm} + \text{Missed} + \text{Speaker Confusion}}{\text{Total Reference Speech Duration}} \tag{2.1}$$

The *Word Error Rate* (WER) is a basic metric used to measure the quality of the speech recognition result. It is similar to the Levenshtein distance. Instead of addressing individual characters, it measures the ratio of word substitutions ($S$), deletions ($D$) and insertions ($I$) in the hypothesis to the total number of words in the reference transcription. It is only defined for single-speaker transcriptions or assumes that all words are from the same speaker and does not provide any information about the correct or incorrect assignment of words to speakers.

$$\text{WER} = \frac{S + D + I}{N} \tag{2.2}$$

The *Word Diarization Error Rate* (WDER) [26] is a combination metric of both diarization and speech recognition results. Like DER, it measures the quality of the diarization, but it is not duration based. It is calculated as the ratio of words with incorrectly assigned speakers to the total number of words in the reference transcription. More specifically the numerator is the sum of all substituted $S_{IS}$ or correctly recognized $C_{IS}$ words in the generated hypothesis transcription for which an incorrect speaker label was assigned. The denominator is the total number of substitutions $S$ and correct words $C$. It does not take into account insertions and deletions in the transcription, since they have no speaker assigned in the ground truth and therefore cannot be compared in terms of diarization accuracy. Therefore, WDER cannot be evaluated without also considering the WER.

$$\text{WDER} = \frac{\mathrm{S_{IS}} + \mathrm{C_{IS}}}{\mathrm{S} + \mathrm{C}} \tag{2.3}$$

*Concatenated Minimum-Permutation Word Error Rate* (cpWER) [27] is a combination metric for evaluating the transcription results for multiple speakers. It uses the speaker assignments of the recognized words to calculate the WER for each speaker. The WER of a transcription with multiple speakers is the average of the WERs for each speaker. Since the speaker labels between hypothesis and reference may not match or may be permutated, cpWER is calculated as the minimum WER over all possible speaker permutations ($\pi(s)$).

$$\text{cpWER} = \min_{\pi \in \Pi} \frac{\sum_{s=1}^{S} \text{WER}(\text{reference}_s, \text{hypothesis}_{\pi(s)})}{S} \tag{2.4}$$

To measure the computational performance, the memory usage and processing time can be measured. Memory usage is measured for the general *Random Access Memory* (RAM) or *Video RAM* (VRAM) dedicated to the *Graphics Processing Unit* (GPU). While the RAM holds all the data from the audio input, to intermediate results, like calculated speaker probabilities and embeddings, to the output as speaker annotations and transcriptions, the VRAM holds the loaded machine learning models and their intermediate results during inference.

The processing time is the duration between when the input to the system was provided and when the output is received. By comparing it to the length of the processed audio, we can measure the real-time factor, so how much faster than real-time the system can run. Alternatively, we can also measure the processing time for each step of a system, such as the speaker diarization or speech recognition steps in the proposed pipeline. This provides more insight into the performance of each component and allows us to track the processing time at the utterance or word level. The total latency is the sum of the computational latency and the artificial latency (delay in pipeline design, see section 4.2) of the pipeline. The computational performance metrics are highly dependent on the hardware used. The goal is to develop a method with a real-time factor greater than one on hardware that is feasible in a robotics context.

# 3. Related Work

There has been extensive research in many aspects of audio speech processing in recent years. This chapter describes several advances in single-task methods that form the basis for the processing steps of the proposed pipeline. It also presents other existing methods that combine speaker diarization with speech recognition for both online and offline processing.

## 3.1. Single Task Approaches

The *pyannote.audio* toolkit [28] is an actively developed Python library for tasks related to speaker diarization. It provides tools for training, evaluating and applying speaker diarization models. Over the years there have been several publications around their segmentation model. Like EEND [20], it is an end-to-end neural network that provides speaker activity values.

The pyannote segmentation model [29] was first published in 2021. It was trained on five second audio chunks from longer conversations. First, audio features are extracted using a SincNet for every block with a duration of 16 ms. Then, bidirectional LSTM layers are used to incorporate temporal context. Finally, fully connected layers are used with the last layer having a sigmoid activation function to provide speaker activity values for up to three speakers. These values are between 0 and 1 and form the basis for the multi-label classification. The model was trained using permutation-invariant training. This means that the loss is indifferent to the generated speaker order.

In 2023, an advancement to the segmentation model was published. Instead of multi-label classification with one label per speaker, the new model [9] performs powerset multi-class classification. They achieved this by changing the activation function of the last layer to softmax and adjusting the output size to the number of powerset classes. Free and open-source pre-trained models are provided for both publications.

Based on the work of pyannote, the Diart pipeline was released, which uses the segmentation model to achieve online speaker diarization. Their approach entails generating local speaker diarizations for incoming overlapping segments of audio. As the speaker order of subsequent diarization outputs can be permutated, they extract speaker embeddings for active local speakers and cluster these using constrained incremental controid clustering as decribed in section 2.2. When the correct speaker mapping is recovered, the speaker probabilities of overlapping windows are aggregated as the average for each frame of the window. Then, the speech segments can be identified by thresholding the aggregated speaker probabilities as new segments become available. [14]

In this thesis, three speaker embedding models are evaluated. The first, `pyannote/embedding` from pyannote [28], is based on x-vector [11] and SincNet [24]. In x-vector, filter banks are used to extract basic features, which in this case are replaced by trained SincNet filters. These features are then processed by a TDNN with increasing temporal context for deeper layers. During training, an additional softmax output layer is used for classification, which is excluded in the final model. This aggregates speaker information over the entire audio input into a 512-dimensional embedding.

The speaker embedding model `pyannote/wespeaker-voxceleb-resnet34-LM` is a wrapper for the WeSpeaker [30] model of the same name. The model is based on r-vectors [31] using the ResNet34 [32] topology. Filter banks are used to extract features from the audio input, which are then refined using the CNN layers of the ResNet model. The number of output dimensions for embedding is 256.

Third, the speaker embedding model `pyannote/spkrec-ecapa-voxceleb` of the SpeechBrain toolkit [33] is based on the ECAPA-TDNN architecture [12]. It improves the x-vector approach by adding self-attention, which can weight different time steps independently for each feature channel to find the most relevant features. In addition, it uses Res2Net blocks [34] to increase temporal context in network layers, and it uses multi-layer feature aggregation to capture both low-level (e.g., pitch, energy) and high-level (e.g., articulation) speech features in the embedding. The number of output dimensions for this embedding model is 192.

In the area of audio enhancement, MossFormer 2 [35] was proposed and published as part of the SpeechBrain toolkit [33]. MossFormer 2 is a speech separation model that uses a masking network to generate the individual speaker outputs and an encode-decoder structure for feature extraction and audio signal reconstruction. The core of MossFormer 2 is a hybrid structure of the previous MossFormer module and a recurrent module. The MossFormer module uses self-attention to capture dependencies within short time segments to capture long-range, coarse-scale dependencies. But it does not caputre recurrent dependencies, which is complemented by the recurrent module to capture recurrent patterns.

## 3.2. Integrated Diarization and Speech Recognition

There are many audio processing systems available that advertise speech recognition and speaker diarization, but in many cases the code and models are proprietary and the companies do not publish their research or detailed documentation. Therefore, only publicly available and reproducible work will be discussed.

WhisperX [7] improves on the original Whisper speech recognition model [6]. The revised system described in the paper first analyzes the audio input with voice activity detection to identify speech segments. Among other things, this allows them to avoid truncation during speech segments when the input has to be split due to length constraints, and it is used to truncate silence segments before performing the actual speech recognition. Not included in the paper and only added to the WhisperX repository [36] after publication is a combination with the pyannote speaker diarization pipeline. After generating the normal Whisper transcription, they apply speaker labels as a post-processing step. First, they compute a complete speaker annotation using the diarization system. Then, for each transcription segment, they compare the intersection of the segment with the speaker utterances in the annotation using the generated segment timestamps. The speaker with the largest intersection is selected and applied as a label to all words for the transcription segment. In this way, they add speaker information to the transcribed words. While this approach is similar to the proposed pipeline in that it uses preprocessing to skip silent segments before applying speech recognition, it assumes non-overlapping speech segments. Therefore, any errors from the speech recognition due to overlapping speech will also propagate to the post-processed speaker information and the final output. A major difference is that it is an offline system.

In 2024, an online audio processing system that combines speech and speaker recognition was proposed [13]. They iteratively generate timestamped segments of transcriptions by using the Whisper speech recognition [6]. The text of these segments is immediately reported to the user, but can still change and be updated. When a certain number of segments is reached, they compute a speaker embedding for the full duration of the oldest segment. They assume a single speaker for the segment and assign a speaker label based on a clustering of the embedding. Only then will the speaker information be reported and the segment will not be changed by subsequent processing. As before, this system requires non-overlapping speech segments to provide accurate results. While this system provides online processing, it increases latency by waiting for multiple transcription segments before outputting the final speaker information. Both methods use the timestamps provided in the Whisper transcription to look up or generate speaker information.

# 4. Method

This chapter explains the proposed pipeline for streaming speaker diarization and speech recognition. Other approaches in the literature, as mentioned in section 3.2, use a transcription-first approach. *Transcription-first* means that they first use speech recognition to generate a transcription of the audio, and only in a later step add additional speaker information to the transcription. The speech recognition system used (Whisper) is only trained to output words grouped by segments and an estimated probability for a segment to contain speech [6], but no speaker information. It is not trained to disambiguate speakers or overlapping speech, but rather expects speech from a single speaker. Both systems mentioned in section 3.2 assume that speakers are disambiguated and expect their speech to be accurately transcribed into an interleaved pattern that can be subsequently associated. In the system by Lyu et al. [13] a single speaker for each extracted transcription segment is assumed and WhisperX [7] use a heuristic to select the most likely speaker. While this may work on non-overlapping speech, overlapping speakers are likely to be suppressed or confused by Whisper's training on noisy data, resulting in a distorted transcription. This is also error-prone in terms of speaker assignments, as embeddings on multi-speaker speech may not be clustered correctly, and for online systems there is less information about future speech available, leading to biased results from a heuristic.

Instead, we employ a *diarization-first* approach that first detects speaker activity and then performs speech recognition with the knowledge of utterances and their speakers. This has the advantage that the audio can be pre-processed based on the speaker information before speech recognition is applied, and the segments processed by the speech recognition system are unlikely to begin or end in the middle of a word or utterance. This last aspect can also be achieved via voice activity detection and removing segments of silence as a pre-processing step [7].

We propose a diarization-first approach that processes segments from an incoming audio stream by first extracting utterances using local speaker diarization, then recovering isolated speaker audio, performing speaker recognition, and finally transcribing the utterances. It provides low-latency transcriptions of words with their associated speaker. The following sections describe each step in detail.

## 4.1. Overview

In Figure 4.1 the general structure of the pipeline is shown. The input can come from a microphone or can be a simulated audio stream from a dataset file. The raw audio data is received in the time domain as samples with a fixed sample rate, which are buffered and made available to the pipeline steps as needed. Next, local speaker diarization is used to extract speaker activity for the most recently received audio segment. This step returns speaker utterances that have not been processed before, and if there is no new utterance, the remaining steps are skipped.

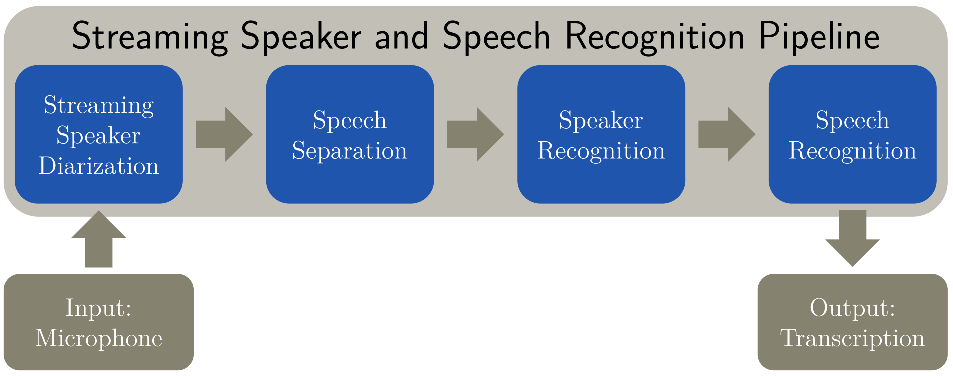

Figure 4.1: The general pipeline structure showing high-level processing steps from the data input as a continuous audio stream to per-speaker transcriptions.

This translation from streaming data to utterance-based time segments allows processing methods in subsequent steps that do not have to adapt to the streaming nature of the input data. However, the methods used in subsequent steps still cannot benefit from information not yet seen at that time step, unlike when these methods are used in systems with offline processing. For example, clustering can only be based on previously seen speaker embeddings and not on all embeddings or any embeddings of future speech, which increases the difficulty of the task for online processing.

For each detected utterance, it is known whether it overlaps with an utterance of another speaker based on the local speaker diarization. In case of overlapping speech, the speaker embedding cannot be extracted directly. In this case, speech separation is used to recover only the audio of the speaker to which the detected utterance belongs. Then, the speaker embedding can be extracted from the recovered audio, which is finally used for speaker recognition to map the local speaker to a global speaker. Finally, the utterance is transcribed using speech recognition.

## 4.2. Streaming Speaker Diarization

The input for *Streaming Speaker Diarization* is the incoming audio stream. The goal is to extract speaker information and provide speaker utterances with timestamps and labels for their respective speaker. A similar approach to *Diart* [14] is used with a changed solution for speaker activity aggregation. We pre-process the incoming audio stream by applying a constant gain limiter to provide a loud but undistorted signal to accommodate quiet or highly dynamic signals. This pre-processed signal is then used for all subsequent processing steps that require audio data.

### 4.2.1. Speaker Probabilities

At the core of the speaker diarization step are the speaker probabilities provided by the pre-trained segmentation model of pyannote.audio [9]. Since the segmentation model requires audio segments as input instead of an audio stream, fixed-length sliding windows are computed on the audio stream as shown in figure 4.2. These windows are defined by the parameters `step size` and `window length`, which must be configured to allow overlap between windows. Overlapping windows are important to maintain speaker detection quality at the edges of the windows and to avoid discontinuities to the next set of speaker probabilities. When the audio buffer has progressed enough to fill the next sliding window, the segmentation model is used to predict the local speaker probabilities on this latest window of the audio stream.

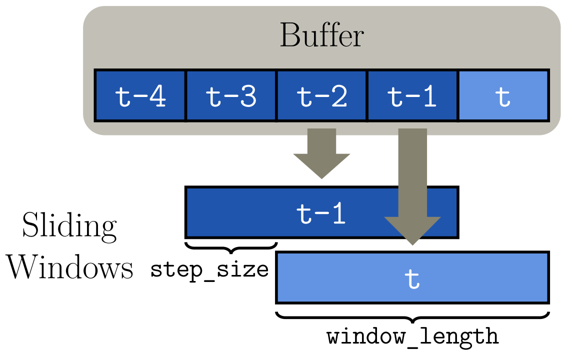

Figure 4.2: Incoming buffered audio segments are concatenated to form segments of sliding windows with overlap. The parameters `step size` and `window length` are used to configure the sliding windows.

The version of the segmentation model used is the latest iteration that uses the powerset multi-class encoding (see section 2.2) as output values for a maximum of three simultaneous speakers. In total there are seven classes, one for the probability of no active speaker, one for each single speaker, and one for all three combinations of two speakers. There is no class for the combination of all three speakers, as this was considered unlikely to be observed based on benchmarks of datasets with diverse domains [9].

### 4.2.2. Speaker Permutation

For successive sets of speaker probabilities, their order does not correspond and must be associated due to the permutation invariant training of the segmentation model. In Figure 4.3, the order of the speakers, indicated by their colors, has changed. The approach used here differs from the Diart implementation in that the speaker identities are used to solve the permutation. However, due to the overlap between windows, there is another piece of shared information, the speaker probabilities themselves, that can be used to reconstruct the correct order. The reason for not using speaker identities is that speaker embeddings computed on very short audio segments of less than or close to a second are difficult to cluster.

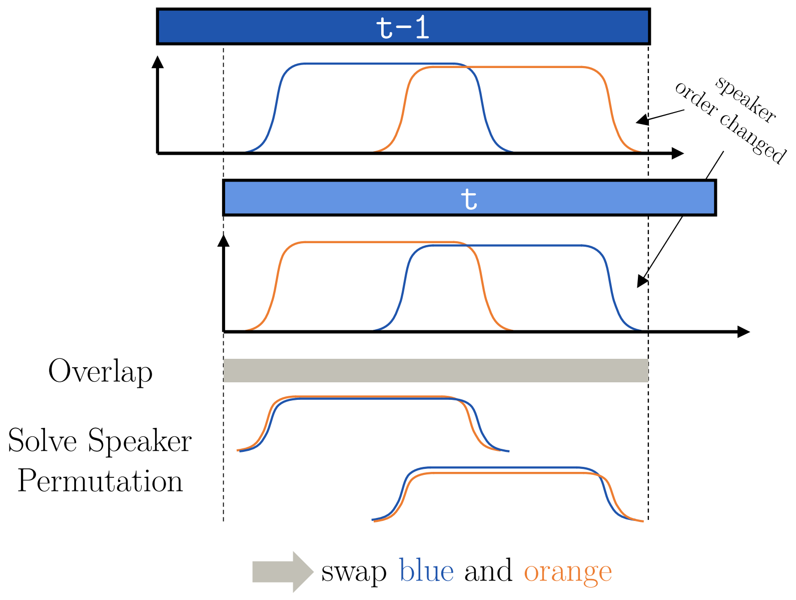

Figure 4.3: The speaker order of successive speaker probabilities can be permutated. In this case, the old *blue* speaker is now recognized as the *orange* speaker and vice versa. The segments are matched by the similarity of their speaker probabilities in the overlapping region.

To solve the speaker permutation problem, the shared speaker probabilities between two overlapping windows are compared. For each possible permutation, the cross-correlation between the common window of the current and previous segment's speaker probabilities is computed. Then, the permutation with the lowest cost of cross-correlation dissimilarities across all speakers is selected. The algorithm used, the *Hungarian algorithm* [37], is the same one used to ensure permutation invariance of the training loss during the training of the segmentation model. It is possible to use both the multi-label representation and the powerset multi-class encoding directly as a basis for cross-correlation. The powerset multi-class encoding provides more information with the explicit probabilities for no speaker present (`class 0`). In the case of powerset multi-class encoding, not all permutations between classes need to be checked, since the number of speakers involved in a class must remain the same. So only the single-speaker classes can be permutated and respectively the combination classes of two speakers, so only those permutations are checked and compared. When the most likely permutation is found, the speaker probabilities are permuted accordingly and saved with the values of previous windows.

### 4.2.3. Probability Aggregation

Depending on the chosen parameters for `step size` and `window length`, there are a number of overlapping sliding windows. Before computing the final diarization results, these values have to be aggregated into one set of speaker probabilities. The values are averaged across their overlapping regions. As speaker probabilities at the edges of windows are inaccurate and to ensure smooth transitions at the edges, the speaker probabilities are weighted with a window function that reduces the weight at the edges of the window. The *Tukey window* [38] is used, which uses a cosine fade-in at the beginning and a cosine fade-out at the end of the window. The remaining part in the middle has a constant value of 1 and is therefore fully weighted. The fade-in and fade-out each occupy 5% of the window length.

Figure 4.4 shows three consecutive outputs of speaker probabilities from the segmentation model. If we always used the most recent data, here the speaker probability data at time step `t=3` from 82.5 to 83.5 seconds, the latency would be minimal, but since there is no overlap with the previous data at this time step, no aggregation is possible. To allow aggregation of overlapping segments and thus improve the quality of speaker activity estimation, an artificial latency is introduced in the form of the `lag` parameter. A delay is added that pushes back the *active time step.* All present overlapping sets of speaker probabilities are aggregated for the duration of the active time step. At this point, an optional confidence decision is made, which is discussed in section 4.2.5. Based on the powerset representation of the speaker probabilities, a confidence score is calculated, and if the confidence is too low, the diarization is deferred until the confidence

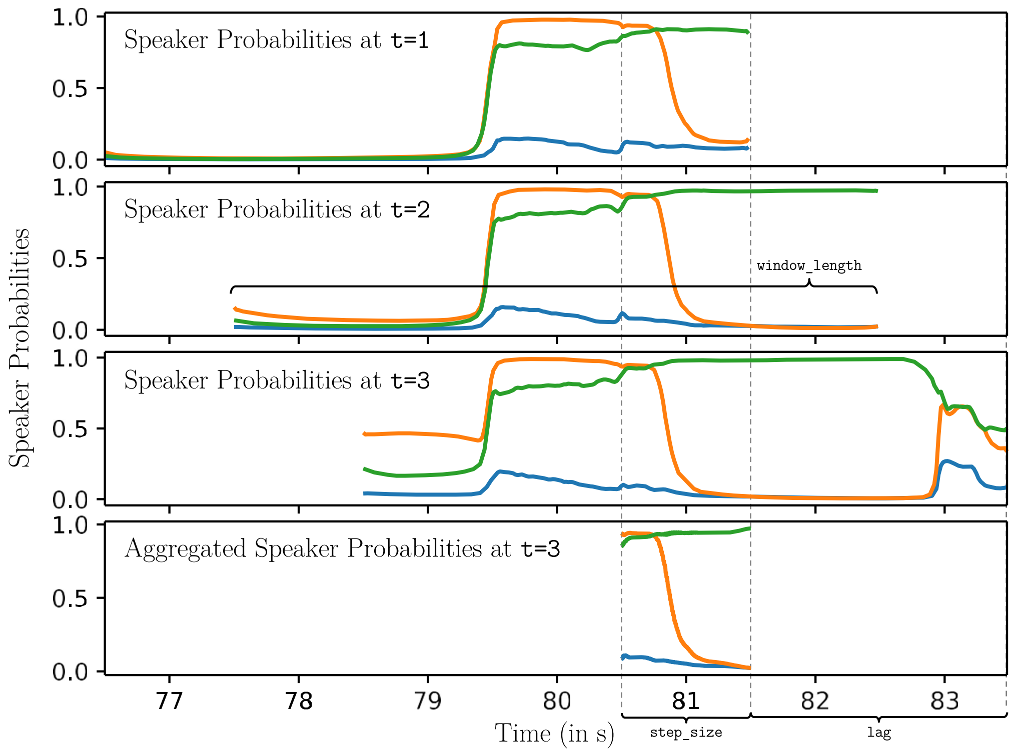

Figure 4.4: Three sets of speaker probabilities are combined for the currently active time step to form aggregated speaker probabilities. The active time step has a duration of `step size` and is delayed by `lag` behind the latest audio data. For readability reasons, the probabilities are shown in their multi-label representation.

increases. After that, the aggregated speaker probabilities are made available for the diarization classification into speech segments.

### 4.2.4. Diarization

To classify speaker activity into segments of speech and segments of silence, a threshold is applied to each local speaker. If the estimated speaker probability is higher than the threshold, the local speaker is considered active. These active speech segments are accumulated and added to previously detected speech segments of the same speaker to form utterances across the edges of the active time step. For each time step, only the aggregated speaker probabilities during the active time step are evaluated, and therefore only this region needs to be aggregated, as shown in the last row of Figure 4.4. This avoids conflicting diarization results for past time steps. For example, a modification or correction could result in a small pause, so that a large utterance is split into two smaller ones, which are now different from the previously processed original large utterance. Thus, additional complexity would be required to later handle these duplicate utterances or their resulting transcriptions.

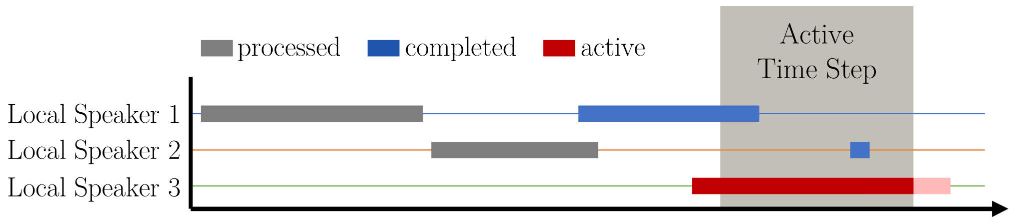

Figure 4.5: Utterances of the three local speakers, colored according to their status at the current time step in either already processed, newly completed, or currently active utterances. The faded segment is not yet processed, but should indicate that the utterance will not be completed until the next time step.

Based on the boundaries of the active time step, all detected utterances can be categorized as either *already processed*, *newly completed*, or *currently active*, as shown in Figure 4.5. Active utterances are utterances that overlap with the active time step but have not yet been completed, meaning that their speaker probability was higher than the threshold at the end of the active time step. They are continued in the next time step. Completed utterances are utterances that are completed during the active time step, which means that their speaker probability has dropped below the threshold which ended the utterance. Most importantly, these utterances are new and have not been processed before. All other detected utterances have been processed in previous time steps. Together, these utterances form the local speaker annotation. The new, completed utterances are selected for the next processing step. Note that they are no longer stream-based, but since their start and end are known, they can be processed with offline methods.

### 4.2.5. Confidence Decision

The speaker probabilities are just an estimate of the actual speaker activity with continuous values between 0 and 1. In the case of the powerset multi-class representation with an ideal predictor, only one class would have a probability of 1 and the others would have a probability of 0. For example, during silence, class 0 should have a probability of 1, or during overlapping speech, only one of the combination classes for two speakers should have a probability of 1, while all other classes should have a probability of 0 each. Instead, ambiguous results can be observed, where multiple classes have partial activation. This effect is particularly noticeable in very low latency scenarios during brief overlapping speech, laughter, or interjections. It results in a lower probability for the correct class, which can lead to incorrect diarizations.

Originally this was solved in pyannote [9] by always choosing the class with the highest value with the `argmax` function. This discards the remaining information encapsulated in the other classes. This information can be used to estimate the confidence of the computed speaker probability values.

$$\text{Confidence Score} = p_{\text{class\_0}} + \max\left(p_{\text{class\_1}}, p_{\text{class\_2}}, \ldots, p_{\text{class\_6}}\right) \tag{4.1}$$

The confidence score is described in 4.1. The maximum over all classes would penalize all probabilities for classes other than the class with the highest probability. Thus, if a second class increases in probability, the confidence score will decrease as the system becomes less certain about the correct class. This would also regularly lower the confidence score at the beginning and end of utterances, since there is a small interval where the probabilities switch from the speaker class to the no-speech class. Instead, `class 0`, which indicates no speech, is added directly and excluded from the maximum, so that only the classes indicating speech are considered. This way, the confidence score remains high when there is a speaker change with a small pause. Since the sum of all classes at a given time is 1, the confidence score can only reach a maximum of 1.

The confidence score alone does not resolve speaker probability ambiguity. To resolve this, the latency of the pipeline is dynamically increased to allow recalculation of the speaker probabilities. After receiving the aggregated speaker probabilities, the confidence score is calculated. If the confidence score is less than a minimum confidence threshold, the final diarization step is deferred. The diarization is delayed until either the confidence score is higher than the threshold again or until a maximum latency is reached. After that, the speaker probabilities are recalculated over the now larger window, which improves the quality. Then diarization is performed over the extended active time step and processing continues as usual.

Figure 4.6 shows the multi-label representation of the initial speaker probabilities, the recalculated probabilities, and the confidence score. Initially, with the low latency setting, several speakers were detected as partially active for the second utterance, and the actual speaker had a lower estimated probability, which would have resulted in a missed speech segment. The diarization was deferred because the confidence score dropped below the threshold. After recalculation, the speaker probabilities were less ambiguous and the correct speaker diarization was obtained.

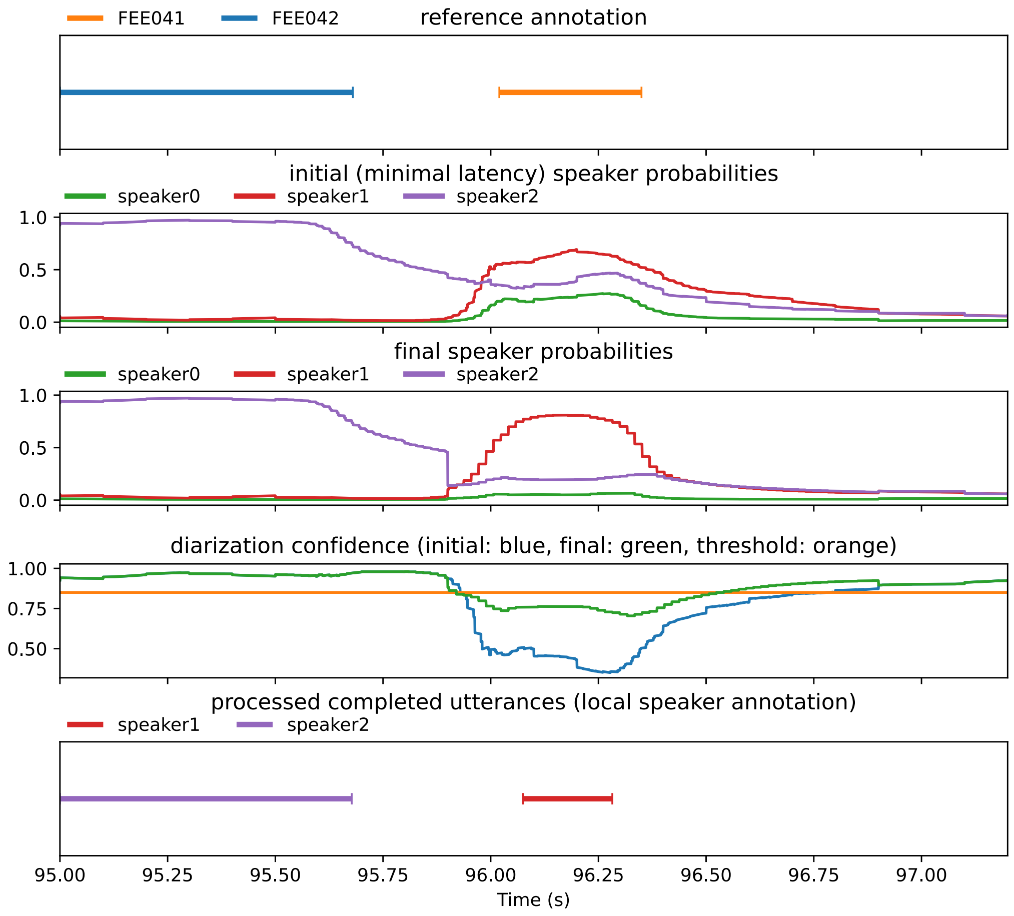

Figure 4.6: An excerpt of the local speaker diarization and confidence values from the file *ES2011a* of the AMI Corpus dataset [39]. Initially ambiguous speaker probabilities are recalculated due to a low confidence score, resulting in temporarily higher latency but better diarization quality.

## 4.3. Speech Separation

The output of the streaming speaker diarization step is a local speaker annotation. More precisely, the newly completed utterances for a given time step are selected for further processing. There are not necessarily new completed utterances for each time step. For example, with a `step size` of 0.1 seconds and an average utterance length of 3 seconds, there are at least 29 time steps without a completed utterance, assuming that there is no overlap between the utterances. In such cases, the processing for these time steps is completed, since the further processing steps depend on completed utterances.

In the case of a completed utterance, the local speaker annotation is queried to determine any overlapping speech with the target speaker for the duration of the new utterance. If no overlapping speech is detected, the utterance can be processed directly and no speech separation is required. Even if only a small part of the utterance overlaps with the target speaker, speech separation is skipped because there is a risk of introducing artifacts into the target speaker's audio if the second speaker is not recognized as a separate speaker. Otherwise, speech separation is used to extract only the target speaker's audio.

The speech separation model MossFormer 2 [35] is used to process the audio segment of the utterance. The model is trained to separate two speakers, and therefore outputs two audio signals, which matches the maximum number of expected simultaneous local speakers as described in section 4.2.1. It would be possible to use different speech separation models for different numbers of speakers in the audio mix. This could avoid potential artifacts for cases where there are more than two speakers in the audio mixture, but it would require loading a second speech separation model into the GPU memory.

Similar to the output of the segmentation model, it is not known whether the target speaker is isolated to the first or the second audio output of the speech separation result. Again, it would be possible to solve this assignment using speaker identities of computed embeddings, but as before, speech segments can be short, leading to imprecise embeddings. The chosen solution is to compute speaker probabilities for both outputs and compare the speaker activity with the previous local speaker annotation. The audio signal is then selected based on similarity in the same way as the permutation solving in section 4.3.

Even if there are exactly two speakers in the audio mix with sufficient speech duration, the separation model may still add artifacts in the form of attenuated speech from the second speaker, static, or high-pitched squeaks. To counteract this, only the overlapping parts of the audio are taken from the separated audio, while the single-speaker segments are kept from the original audio.

During development, it was observed that the separation output did not match the original audio in amplitude. This is likely due to the fact that the model is trained on a dataset with a specific loudness. Therefore, in order to match the original audio as closely as possible, the loudness of the audio is adjusted using the same limiter configuration as at the beginning of the pipeline. This is done immediately after the separation and before the speaker probability estimation, as these can also be affected by loudness. The end result is an utterance with constant loudness that contains only the isolated audio of the target speaker.

## 4.4. Speaker Recognition

Up to this point, the pipeline only has local speaker information for the given utterance, originally provided by the segmentation model. This is useful for discriminating between speakers speaking at the same time or in close temporal proximity, but is not suitable for tracking speaker information over long periods of time, especially with silence in between. Therefore, speaker recognition is used on the extracted utterances to obtain global speaker information and to match local speaker utterances to recurring global speakers.

The speaker recognition step consists of two components. First, a speaker embedding is extracted from the isolated audio. For this embedding models such as X-Vectors [11] or ECAPA-TDNN [12] are used. The speaker embedding is then used in an online clustering algorithm. The clustering algorithm will either assign a label of a matching cluster or create a new cluster for a new speaker.

There are some requirements for the used clustering algorithms. Since the embedding models are usually trained on cosine distance, the clustering algorithms should be able to handle cosine distance. Then, due to the streaming nature of the pipeline, only embeddings of previous utterances are known, so the algorithm must be able to handle sparse data. Also, the total number of speakers is not known, so any algorithm that uses a fixed number of clusters is not suitable. The clustering algorithms and embedding models used are described in section 2.2 and section 3.1 respectively.

## 4.5. Speech Recognition

The last step in the pipeline is the speech recognition step. The words of an utterance are transcribed using a speech recognition model such as Whisper [6]. Since overlapping speech and speaker assignment are handled in the previous steps, the single-speaker assumption of Whisper is retained. The output is a per-utterance transcription, including the speaker label, which is provided to the user once the transcription process is complete. Each word in the transcription also retains the estimated timestamps provided by Whisper.

In most cases, the identified utterances contain speech, but sometimes very short interjections or laughter are identified as utterances during the diarization step that cannot be associated with any words during speech recognition. The Whisper model has a tendency to add hallucinated words during periods of silence or when it cannot distinguish words. The authors include a `no speech` probability with each segment of the transcription output. This is used with a threshold to filter out segments with a high probability of hallucinated words.

Due to the architecture of the pipeline, it would be possible to skip the speech recognition step altogether and output the separated audio with its speaker information directly for tasks that do not require word transcriptions. This is not discussed further in this paper because the standard metrics used for similar methods are based on the transcribed words. This makes comparison with similar approaches difficult, as new metrics would be needed. However, one such use case would be a multimodal large language model such as GPT-4 [40]. Instead of transcribing the audio to text first, the model can process the audio directly in the end-to-end model, adding additional information not covered in a pure text-based transcription, such as tone, emphasis, or non-word based sounds like laughter [41].

In general, the pipeline is easily modifiable, so it can be changed to handle simpler tasks. For example, if only single-speaker speech is expected, the speaker recognition module could be skipped and only a single speaker assigned in the output. However, for the evaluation in chapter 5, all modules are active for all scenarios.

## 4.6. Implementation

The proposed pipeline has a modular architecture. The idea is that each step can be replaced by a different implementation as long as the inputs and outputs are compatible. This is especially advantageous for changing the different foundation models, the clustering algorithm, or the processing logic of the pipeline steps themselves without having to rebuild the rest of the pipeline. Data is stored centrally in buffers for the audio stream, speaker probabilities, confidence scores, local and global annotations, and transcriptions. This way, data is not unnecessarily duplicated, can be accessed by different processing steps as needed, and can be globally purged when no longer needed.

The pipeline is implemented in Python and executed in a Docker environment to be as independend as possible from the host environment. It is designed as a standalone application, but can be integrated with other software by periodically providing the audio stream as input. The output data or any intermediate data can be retrieved via hooks at various points in the pipeline. These hooks are also used to gather the data for the quality and performance metrics in the evaluation.

The pipeline was built on an Ubuntu 20.04 host with NVIDIA's CUDA 12.2 installed. The base Docker image is `nvidia/cuda:12.2.2-cudnn8-runtime-ubuntu 22.04`, which comes with Python 3.10. The main Python libraries used are PyTorch 2.5.1 for running the models, pyannote.audio 3.3.2 [28] for diarization-related helper functions and providing pre-trained models, and speechbrain 1.0.2 [33], also for pre-trained models.

The used speech separation model used is `pyannote/segmentation-3.0` provided by pyannote, which also provided the speaker embedding models `pyannote/embedding` and `pyannote/wespeaker-voxceleb-resnet34-LM`. Another speaker embedding model is `speechbrain/spkrec-ecapa-voxceleb` which is provided by SpeechBrain. The speech separation model used is MossFormer 2 [35] provided by SpeechBrain and the speech recognition model is Whisper [6] `large-v3` with the implementation of faster-whisper 1.0.3 [42]. The pre-trained models are available at either HuggingFace [43] or ModelScope [44].

For clustering algorithms, the implementation for online incremental clustering is based on Diart [14] and the implementation for DBSTREAM [22] is provided by the Python library river [45]. Pyannote provides an implementation for the DER metric. The WER, cpWER and WDER metrics are provided in the GitHub repository of DiarizationLM [46]. DiarizationLM applies input normalization for the reference and hypothesis transcriptions by converting text to lower case and removing punctuation.

# 5. Evaluation

This chapter evaluates the proposed pipeline described in chapter 4. First, the datasets used for the evaluation are described. Then the evaluation setup is described, including the selection of parameters for the pipeline. Finally, the results of the evaluation are presented.

## 5.1. Datasets

There are many speech recognition and speaker diarization datasets that have been created over the years for different use cases. The evaluation of the proposed pipeline requires datasets that provide ground truth for both speaker annotations and transcriptions. While both independently provide partial information, the pipeline can only be fully tested with both pieces of information available on the same dataset. At least one dataset must provide speech with overlap between different speakers, as in conversations, to test realistic multi-speaker scenarios. Therefore, two datasets, *LibriSpeech* [47] and *AMI Corpus* [39], were selected, and additional datasets are sampled from LibriSpeech to gain further insight into specific scenarios. Table 5.1 and Table 5.2 provide statistics about the selected datasets.

LibriSpeech [47] is a dataset of 1000 hours of read audiobooks from the public domain project *LibriVox* [48]. The audiobooks are always read by a single person, making this a single-speaker dataset. The audiobooks are divided into short segments of a few utterances per dataset file, with an average of 7 seconds per file and a maximum of 30 seconds. The domain is natural, easily understandable, non-spontaneous speech with very little background noise. The audio books read are all in English and may contain difficult sentence structure or names. The dataset contains training, development and test splits, of which only *dev-clean* and *test-clean* are used. Since most of the audio is contained in the training split, the selected splits each contain 5.4 hours of audio.

Since the transcriptions provided by LibriSpeech only include the text and speaker labels, a second source is used that provides timestamps for utterances and words. Jemine [49] provides precomputed alignments for the words in the

| Dataset Name | #Files | Total File Len. in h | Avg. File Len. in s | Avg. Segment Len. in s | #Spk. | Avg. #Speakers per File |
|---|---|---|---|---|---|---|
| LibriSpeech$_{dev}$ | 2703 | 5.4 | 7.18 | 1.85 | 40 | 1 |
| LibriSpeech$_{test}$ | 2620 | 5.4 | 7.42 | 1.80 | 40 | 1 |
| AMICorpus$_{dev}$ | 18 | 9.7 | 1933 | 3.64 | 21 | 4 |
| AMICorpus$_{test}$ | 16 | 9.1 | 2039 | 4.10 | 16 | 3.94 |
| Monologue$_{dev}$ | 437 | 4.8 | 39.68 | 3.64 | 40 | 1 |
| Monologue$_{test}$ | 441 | 4.9 | 39.77 | 3.65 | 40 | 1 |
| Conversation$_{dev}$ | 356 | 4.0 | 40.48 | 3.72 | 40 | 4 |
| Conversation$_{test}$ | 329 | 3.8 | 41.12 | 3.78 | 40 | 4 |
| Conversation*$_{dev}$ | 247 | 3.4 | 50.15 | 5.37 | 40 | 4 |
| Conversation*$_{test}$ | 229 | 3.4 | 52.76 | 5.63 | 40 | 4 |

Table 5.1: First part of the overview of development and test sets of the datasets used for the parameter selection and evaluation. The asterisk (*) for the last two rows indicates that the dataset was sampled with overlapping segments.

| Dataset Name | Avg. Words per min. | Speech in % | Silence in % | Overlap vs. Total in % | Overlap vs. Speech in % |
|---|---|---|---|---|---|
| LibriSpeech$_{dev}$ | 168.29 | 83 | 17 | 0 | 0 |
| LibriSpeech$_{test}$ | 162.17 | 84 | 16 | 0 | 0 |
| AMICorpus$_{dev}$ | 192.76 | 78 | 22 | 11 | 14 |
| AMICorpus$_{test}$ | 192.74 | 80 | 20 | 12 | 15 |
| Monologue$_{dev}$ | 180.27 | 92 | 8 | 0 | 0 |
| Monologue$_{test}$ | 172.30 | 92 | 8 | 0 | 0 |
| Conversation$_{dev}$ | 180.82 | 92 | 8 | 0 | 0 |
| Conversation$_{test}$ | 171.97 | 92 | 8 | 0 | 0 |
| Conversation*$_{dev}$ | 204.17 | 98 | 2 | 9 | 9 |
| Conversation*$_{test}$ | 191.74 | 98 | 2 | 8 | 9 |

Table 5.2: Second part of the overview of development and test sets of the datasets used for the parameter selection and evaluation. The asterisk (*) for the last two rows indicates that the dataset was sampled with overlapping segments.

LibriSpeech dataset, which can be used to provide annotation ground truth for evaluation. They achieve this by using the Montreal Forced Aligner [50], which uses learned pronunciation dictionaries to map words to phonemes in the audio. They then recover the timestamps of the detected phonemes and, in turn, the words.

The *AMI Corpus* [39] is a multi-speaker dataset containing 100 hours of meeting conversations. The conversations average about half an hour in length, but can be over an hour in some cases. There are several sets of recordings with different microphones. The specific audio used is a mix of the headset microphones worn by the participants. There are four speakers per conversion, but for one file in the test set there are only three speakers. The conversations are in English and include discussions about work projects with roles for each speaker and naturally occurring meetings with varying topics. While there may be longer segments with only one speaker speaking, these contain natural interjections, but there are also segments of discussion with natural overlapping speech in rapid succession. Training, development, and test splits are provided, with the development and test sets totaling 9.7 and 9.1 hours of audio, respectively. The ground truth for each conversion consists of transcriptions with speaker labels and word-level timestamps.

For both datasets, the training data is not used during the evaluation to avoid biased results, as some of the foundation models used were trained on these datasets. While the development sets were used during pipeline development and parameter selection, only the test sets are used for evaluation.

The datasets labeled as *Monologue* and *Conversation* are sampled from utterances of the LibriSpeech dataset. Each dataset file in Monologue contains 10 utterances of the same speaker, concatenated with small pauses of 0.3 seconds between utterances. Since the LibriSpeech dataset files contain at most a few utterances, this dataset can provide insight for longer single-speaker scenarios. For Conversation, 4 speakers per file were used with round-robin scheduling to ensure a speaker change after each utterance. For *Conversation**, again 4 speakers were used, but with a pause length of -0.5 seconds, meaning that there is no silence between utterances and they overlap. The Conversation datasets provide insight into multi-speaker scenarios without speech overlap and scenarios with overlap that are not as long as the AMI Corpus files. Note that Conversation* does not include interjections, only interruptions and implicit speaker changes.

The actual pause length and overlap in the generated data do not exactly match the values of 0.3 seconds and -0.5 seconds because there is some inaccuracy in the alignment of the LibriSpeech dataset. These values were chosen to feel like a natural pause length between speakers and to have a slight overlap, as if speaker A is interrupted by speaker B, based on randomly chosen samples. Each dataset split contains only sampled utterances from its respective split in LibriSpeech.

For reasons of consistency between sampled utterances, pauses shorter than the absolute value of the selected pause length in the original dataset file are ignored and these neighboring utterances are merged. This results in longer utterances on paper for the sampled datasets compared to the original LibriSpeech dataset and also for the Conversation dataset with overlap compared to the other sampled datasets. Another observation is that the sampled datasets contain less silence around the utterances than LibriSpeech and the AMI Corpus. This is mainly due to untrimmed silence at the beginning and end of the original dataset files and speaker pauses between utterances that occur in natural conversation and are not considered in the sampling. This is also partly an effect of merging adjacent utterances with short pauses during sampling.

## 5.2. Evaluation Setup

The pipeline is evaluated alongside three other methods that serve as comparisons for the different goals defined in the introduction. While the other methods are partly offline methods or handle data loading themselves, the pipeline can only process data streams. Therefore, these streams are simulated by loading audio files and providing the data to the pipeline in chunks. Each chunk has the length of the pipeline's `step size`, but any other length could be used, as the pipeline's audio buffer would simply buffer the data until there is enough data to process in one step. A silence padding is added during the simulation to simulate the idle state of a real input device receiving audio. This way, the pipeline receives the first chunk of audio containing speech in the same way as later steps, rather than starting with a large chunk at the beginning.

The various pipeline processing steps have parameters that need to be configured for optimal performance. This section first calculates the pipeline parameters and then describes the various other methods. All parameter selection calculations and the various evaluations are performed on an *Intel Core i9-9900K* CPU and a *NVIDIA GeForce RTX 4090* GPU with 24 GB of VRAM. All methods use the same Docker environment described in section 4.6.

### Parameter Selection: Speaker Recognition

In section 4.6, three speaker embedding models and two clustering algorithms are listed. To determine which model and algorithm is best suited for the pipeline, the clustering performance is evaluated for the different embedding models and algorithms. Based on the ground truth annotation of the development set of the

AMI Corpus, speaker embeddings are extracted for each utterance present. As in the pipeline, the embeddings are clustered online with the respective clustering algorithm and the speaker segments are mapped to global speakers. In addition, a second set of utterances and embeddings is obtained by detecting silences in utterances originally labelled as speech and removing the silences to create shorter utterances. To detect silences, the voice activity detection of *Silero VAD* [51] is used. These two sets of embeddings are marked as long and short utterances in Figure 5.1.

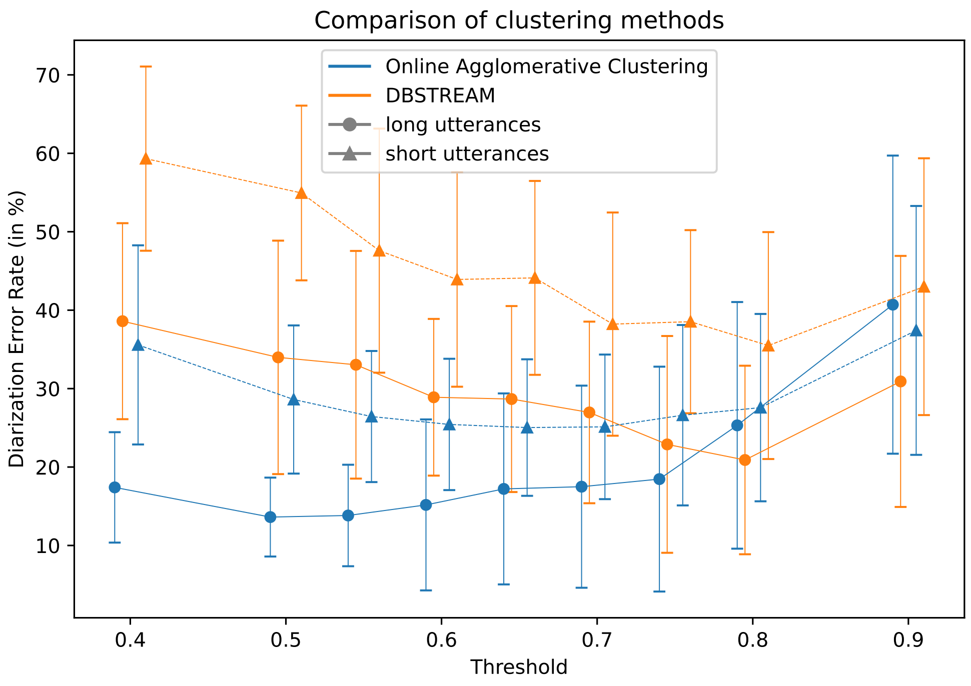

Figure 5.1: A comparison of clustering methods with different clustering threshold settings. The values shown are the mean DER and the standard deviation of the resulting local speaker annotation with `pyannote/wespeaker-voxceleb-resnet34-LM` used as the speaker embedding model. The x-axis offset is only a visual separation for better readability, and the thresholds evaluated are the same for all methods.

Figure 5.1 shows the mean DER for different clustering thresholds. The error bars indicate the standard deviation around the mean. The data are shown for the combinations of both clustering algorithms with the `pyannote/wespeaker-voxceleb-resnet34-LM` speaker embedding model. The other models produce 5 to 20 percentage points worse results than the selected model for both clustering algorithms, but otherwise show similar characteristics. Additional data for the other speaker embedding models can be found in appendix A.

Comparing the respective best results for shorter and original speech segments, the error for the longer speech segments is lower for all embeddings and algorithms. Online Agglomerative Clustering shows a minimum around a clustering threshold of 0.5 to 0.55 for the original segments and 0.6 to 0.7 for the shorter segments. For DBSTREAM, the minimum error is higher at 0.8.

DBSTREAM is consistently outperformed by the former algorithm at its best clustering thresholds, both in terms of mean and variance. While Online Agglomerative Clustering has the clustering threshold as its only parameter, the DBSTREAM algorithm has more parameters, but the influence of the other parameters is insignificant compared to the clustering threshold. Therefore, Online Agglomerative Clustering is selected with a clustering threshold of 0.5.

### Parameter Selection: Streaming Speaker Diarization

The streaming speaker diarization includes the following parameters: `limiter gain` and `limiter threshold` of the audio pre-processing, as well as `step size`, `window length`, `lag`, the `diarization threshold` and the `confidence threshold`. They all contribute to the local speaker diarization. To find good values for these parameters, the classification accuracy into speech and non-speech segments is compared to the ground truth for different parameter settings. We make the assumption that pipeline parameters that show good speaker classification for the first speaker will also lead to good classification for an overlapping second or third speaker. Although the local speaker diarization contains individual information for speech and non-speech segments of each local speaker, the speech activity over all speakers is used because the diarization error is more volatile due to the activity of changing and concurrent speakers, leading to more ambiguous results.

Another constraint is that the parameters must be chosen so that the computation of the pipeline steps is fast enough to run in real time. This is discussed in more detail as part of the evaluation results in section 5.3. The parameters that have a large impact on the runtime are `window length` and `step size`. They determine how often and how much data is processed. While a high confidence threshold can also result in more reprocessing, this parameter is configured to only affect low confidence spikes that are few and short in duration, resulting in a low impact on runtime.

To evaluate different settings of these parameters, the streaming speaker diarization pipeline step is run on a subset of the AMI Corpus. Then, the speaker probabilities are grouped into speech and non-speech categories by their ground truth labels and classified by the diarization threshold. Grid search is used to find good parameters by evaluating the classification accuracy for each combination of parameter values and iteratively refining the search space. Since there are many parameters that are difficult to optimize on their own, many combinations must be tested, which is time-consuming for large datasets. This subset contains 30 randomly selected 60-second audio segments from the development set, kept fixed for all parameter settings. This is a balance between enough test data to get meaningful results and a practical runtime for parameter selection.

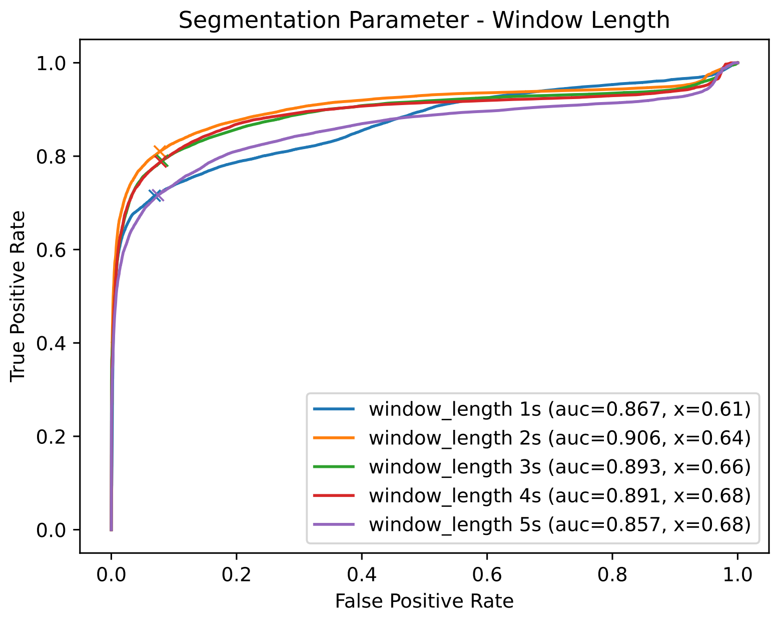

Figure 5.2: Receiver Operating Characteristic (ROC) for different window lengths showing optimal values for the diarization threshold parameter. Both too long and too short windows produce suboptimal results. The respectively optimal threshold value `x` is marked on each curve.

Figure 5.2 shows the receiver operating characteristic (ROC) curves for different window lengths and a fixed step size of 0.3 seconds. They plot the ratio of correctly classified speech segments (sensitivity) to the ratio of incorrectly classified non-speech segments (specificity) for the full range of diarization thresholds. For each curve, the threshold that maximizes the sum of sensitivity and specificity is highlighted. The result for window lengths of one to three seconds is largely homogeneous, while window lengths below and above this are less accurate over a wide range of thresholds, as indicated by the reduced area under the curve (AUC). While too short windows do not provide enough audio information for the segmentation model to accurately distinguish speech from non-speech, too long windows are more prone to speaker permutation errors.

Note the sharp increase in sensitivity for the upper right corner, which means that almost regardless of the threshold and other parameters chosen, there are about five percent of misclassified speech segments. This means that about five percent of the audio labeled as speech in the dataset will be classified as non-speech by the segmentation model no matter what. This is most likely due to errors in labeling in the dataset or different definitions for margins around speech and speech pauses.

The other parameters are more stable, yielding optimal results over a wider range of values. There are several reasons for this. First and foremost, some parameters only mitigate extreme events, such as unusually quiet utterances or segments with low diarization confidence, which are rare and therefore not equally represented in the dataset. Since these are underrepresented, the parameter evaluation is less sensitive to them. In these cases, the parameter evaluation mainly shows that the chosen parameters are also good for the majority of the data that does not contain the extreme events. Second, the chosen metric does not directly evaluate the quality of the overlapping speakers, but only the overall speech activity. This means that the effect of some parameters on the quality of speaker probabilities in overlapping cases is not measured. Specifically for the lag parameter, the increased duration means that the processed speaker probabilities are selected from the center of a speech segment rather than the edges, leading to more pronounced speech activity and fewer speaker permutation errors.

The `limiter gain` is set to 20 dB with the `limiter threshold` set to -1 dB, amplifying quiet speech segments and bringing all audio inputs to a similar level. The `step size` is set to 0.3 seconds, with a `window length` of 2 seconds based on the results from Figure 5.2. The duration of `lag` is also set to 0.3 seconds to allow a full time step of overlap between aggregated segments, but no more to keep the latency low. The `confidence threshold` is set to 0.7, which is lower than the confidence level normally observed, so that only strong cases with low confidence are recalculated. The `diarization threshold` is set to 0.64, which is the optimal threshold based on the results of Figure 5.2.

## Parameter Selection: Speech Recognition

Lastly, for the speech recognition step there is the size of the Whisper model and the `no speech probability` parameter to consider. While smaller models provide faster inference leading to shorter latency, they provide less accurate transcriptions. The chosen model size is `large-v3`.

Whisper provides an estimated probability for each processed segment indicating whether it contains speech or not. This is used to mitigate hallucination in

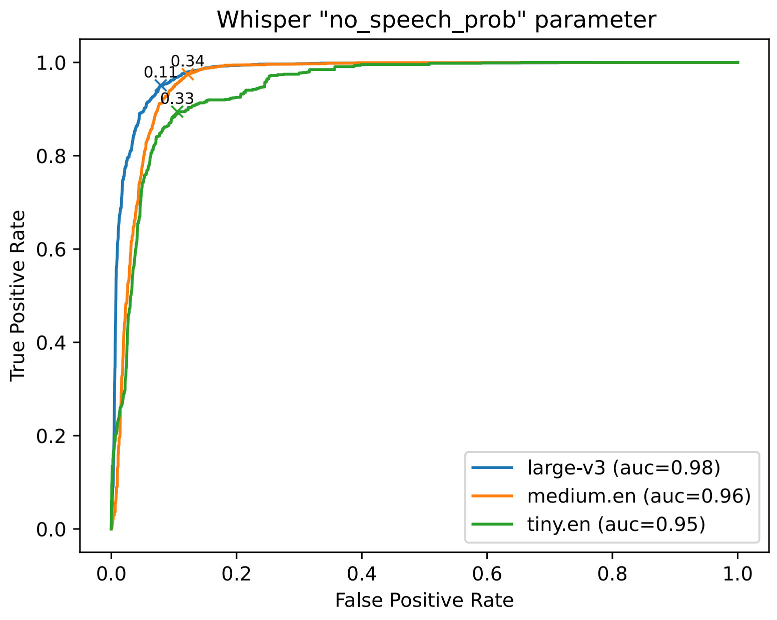

Figure 5.3: ROC curves for different values of the `no_speech_prob` parameter of Whisper models. Only select models are shown for readability, as the larger models all show high accuracy in this speech detection test. Best results are achieved with `large-v3` at a threshold of 0.11.

the model. To determine a good value for this parameter, speech recognition is performed on speech and non-speech segments of the development set of the AMI Corpus. Similar to Figure 5.2, the estimated speech probabilities of the Whisper transcription are then grouped by ground truth labels and classified with the `no speech probability` as a threshold. In Figure 5.3 the ROC curves for different values of the `no speech probability` parameter are shown. This does only show information about the accuracy of detecting any speech and not the quality or accuracy of the transcriptions. Larger Whisper models and models that are not English-only have a better accuracy for speech detection, with `large-v3` having an optimal threshold at 0.11 for the `no speech probability` parameter.

### Baseline Methods

To compare the results of the proposed pipeline with existing methods, the existing methods are also implemented and evaluated on the various datasets. These three methods are used as baseline methods. However, they do not necessarily provide the same functionality as the proposed pipeline, such as missing speaker information or low latency processing. A one-to-one comparison may not be possible, but only for certain aspects.

The first is the previous method used in [5], which consists of online voice activity detection and then utterance-based speech recognition using Whisper. The method implemented for this experiment first performs voice activity detection on the entire dataset file. The voice activity detection system used is Silero VAD [51], which provides real-time timestamps for voice activity. After the utterances are detected, speech recognition is performed. This method does not provide any information about the speaker.

Additionally, the transcription first methods mentioned in section 3.2 are implemented. So the second method is modeled after WhisperX as described in its GitHub repository [36], which includes the added speaker diarization. For a given dataset file, a single-speaker transcription is created using Whisper, and the speaker diarization is created using pyannote's diarization pipeline. Then the timestamps of the transcribed utterances in the Whisper transcription are used to find the best matching speech segments in the diarization output. The words in the utterance are then assigned to the speaker label given by pyannote. This method is denoted as *Whisper+Lookup* during the evaluation.

Finally, the method by Lyu et al. [13] is implemented. Similar to the previous method, utterances are extracted using the Whisper transcriptions, but then embeddings are computed based on the time segments of the utterances. These utterances are then clustered one at a time using the same clustering algorithm used in the proposed pipeline. The resulting label after each individual clustering output is assigned to the words in the given utterance. This method is denoted as *Whisper+Clustering* during the evaluation.

While the second method is also originally computed offline, the first and third methods are originally online methods, but are now implemented as offline methods. However, the first and third methods are implemented in such a way that they do not take advantage of information that would not be available at a given time step. In this way, the diarization and transcription output of the systems does not change, only the output latency would be different, as the results are provided at the very end of the computation.

## 5.3. Results

The pipeline is evaluated in terms of processing time and computational resources required, as well as diarization and transcription quality. The metrics used are defined in the section 2.5.

### 5.3.1. Computational Performance Analysis

The goal of the computational performance analysis is to show whether the proposed pipeline meets the real-time and low-latency requirements of the robotics context. In addition, the computational resources consumed by the pipeline are measured. The processing time for the pipeline and its substeps is measured during the computation for the file *EN2002a* of the AMI Corpus test set. The file contains 36 minutes of audio with speech overlap present, which is sufficient to measure the processing time for all steps of the pipeline. The processing time is measured by comparing timestamps before and after a pipeline step using Python's `time.perf_counter_ns()`, excluding the overhead cost of the measurement itself. RAM and VRAM usage by the Python process is also measured during this experiment.

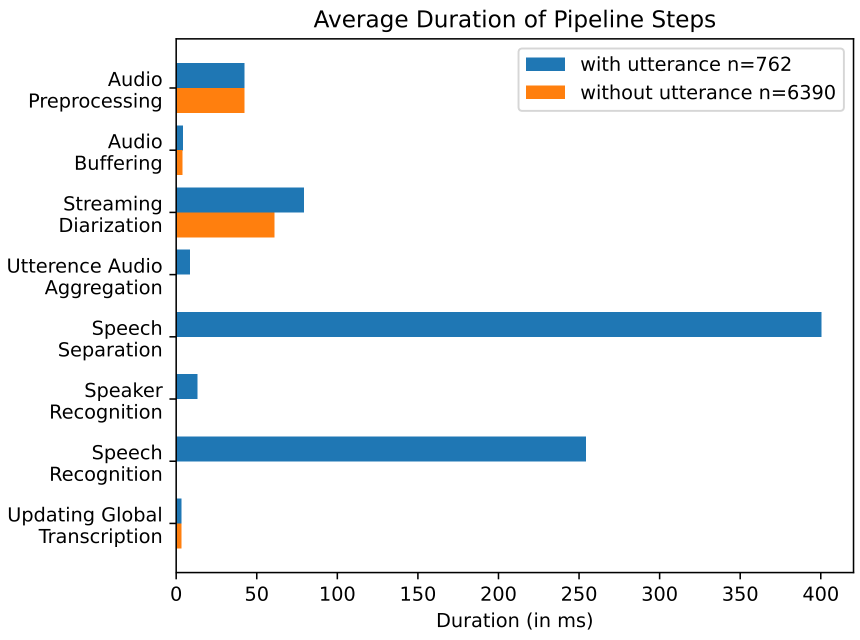

Figure 5.4: The average processing time for the pipeline steps is displayed. The results are grouped according to whether an utterance was completed during the computation of a time step. For steps with a completed utterance, the inference time of the speech separation and recognition models dominates the computation.

Figure 5.4 and Figure 5.5 show the average processing time for each pipeline step or substep of the streaming speaker diarization step. The data is grouped according to whether the result of the diarization step at a given time step contains new completed utterances or not. The aggregation, separation and speaker recognition steps show a duration of zero seconds for time steps without new completed utterances, because these steps are only executed when there is a newly completed utterance. Pre-processing and buffering the audio input and updating the global transcription do not differ in the processing time difference between time steps with and without new completed utterances. For steps with an utterance, the processing time is dominated by the speech separation and speech recognition steps due to the inference time of the larger models.

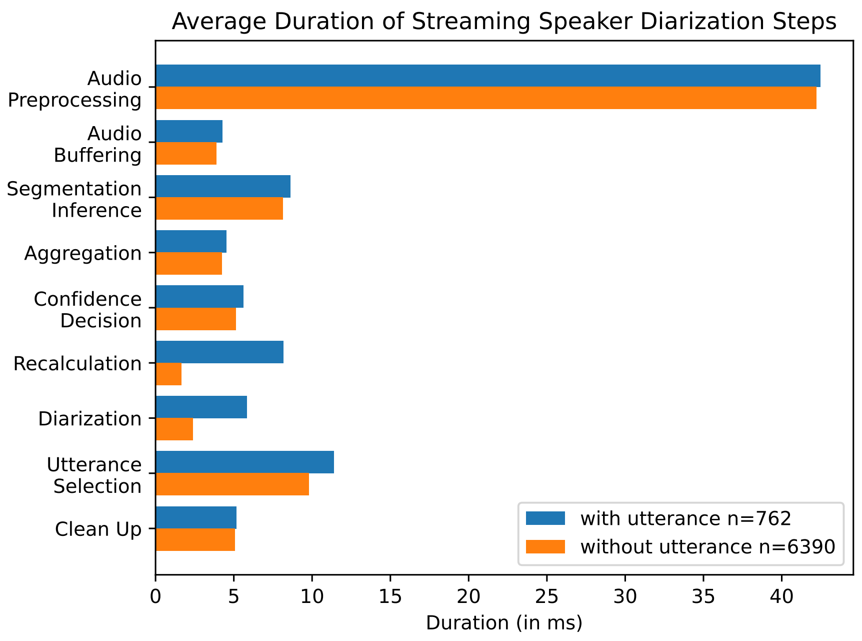

Figure 5.5: The average processing time for pipeline steps during the streaming speech diarization is displayed. The results are again grouped according to whether an utterance was completed during the computation of a time step.

For streaming speaker diarization, the time steps with new completed utterances require more processing time. Since speaker changes are the main cause of low diarization confidence, there can often be a recalculation of speaker probabilities that leads to an utterance ending, so the processing time is increased by the recalculation. Figure 5.5 shows that the following two substeps also have higher processing times with new completed utterances, due to processing a longer segment than the normal step size. Note that the utterance selection for time steps without a completed utterance is not zero, because there is still processing needed to detect the active utterances.

The streaming speaker diarization step is dominated by the audio pre-processing. The pre-processing is currently implemented in Python, which is slow because each frame depends on the previous frame, but could be improved with a faster implementation in C. A matching implementation of the pre-processing in C is measured to be over 50 times faster and could be called from the pipeline via Python bindings.

The average pipeline processing time for time steps with no new completed utterances is 110 ms and 806 ms for time steps with new completed utterances. The former is significantly lower than the chosen step size of 300 ms. Furthermore, the total average processing time is 184 ms. This means that, on average, the pipeline takes less processing time than the duration of the input audio, thus meeting the real-time requirement. For the pipeline latency, the 806 ms is the more relevant value, because the delay is calculated between the time of audio input and when the output data is generated, which is only the case for time steps with a new completed utterance. The duration of 806 ms is larger than the step size, which means that it creates a backlog for the next time steps. In most cases, this backlog is irrelevant because the next utterance does not occur immediately and the backlog can be recovered after a few seconds due to the faster than real-time processing for time steps without new completed utterances.

While the processing time for time steps without new completed utterances has low variance over all measured time steps, the processing time with a completed utterance highly depends on the length of the processed utterance. In the analyzed file, the step with the longest processing time had a duration of close to six seconds, but these extreme values are only reached by rare outliers.

During the latency evaluation, an unexplained error was observed that causes a sudden increase in processing time for all steps of the pipeline after processing a part of the dataset file. This error occurs after about 12.5 seconds, leaving the first part unaffected. Since the increased processing time has a significant impact on the backlog, only the unaffected and explainable first part is used for the latency evaluation. The increase in processing time affects all processing steps equally and therefore does not change the main evaluation results. Solving this problem would lead to different results for the parameter selection, as lower `step size` and `lag` could be chosen, but these parameters have only a small impact on the quality. The diarization and transcription results are not affected by the increased processing time, only the latency.

The left graph in Figure 5.6 shows the simulated backlog that would normally accumulate if audio were delivered and processed in real time instead of as quickly as possible. As mentioned earlier, in most cases there is no backlog, or it can be recovered in a short time. While there are many spikes of backlog, about 74% of the time steps have no backlog, and another 5% of the time steps can be recovered immediately during the next step.

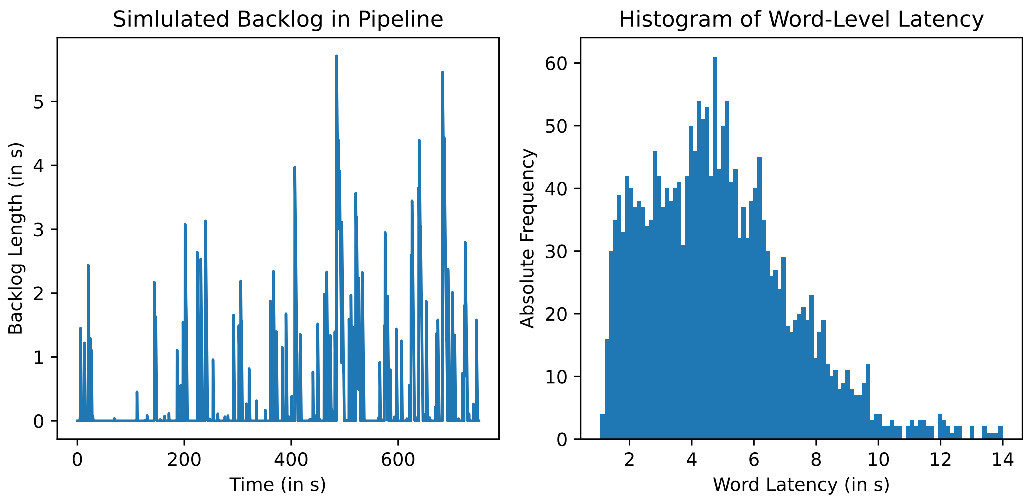

Figure 5.6: Left: Length of the pipeline backlog during the processing of *EN2002a* of the AMI corpus test set.
Right: Word-level latencies of transcribed words. The latency is influenced by the backlog from a previous time step, the length of the detected utterances, as well as the artificial and computational latency of the pipeline.

Word-level latency measures the time between when a word is spoken and when the transcription and speaker information is available in the pipeline output. The worst example of pipeline latency would be the first word of a long utterance. The pipeline would wait until the utterance is complete, which is almost the entire duration of the utterance plus the artificial latency (`step_size + lag = 0.6` seconds) of streaming speaker diarization. Finally, the computational latency is added as the time it takes for the output to be provided by the pipeline. In addition, the backlog from a previous time step also contributes to the increased latency. This total latency is measured as the difference between the timestamp of the end of a detected word and when it is reported by the pipeline with the simulated backlog from before.

The right graph in Figure 5.6 shows much longer latencies than just computational latency, which is often the only factor reported in other work. Our latency measure more accurately represents the amount of time the audio information was available and could have been processed, rather than a reaction time after an utterance was completed. The majority of word-level latencies are between 1 and 6 seconds. Most of the detected utterances have a duration of less than 4 seconds, to which an additional ~1.4 seconds must be added for artificial and computational latency. Assuming that words are generally evenly distributed in an utterance, this results in words with latency values ranging from ~1.4 to ~1.4 plus the utterance length, which in the majority of cases explains the word-level latency shown. In rare cases, increased latency occurs when several long processing steps occur in close succession, but most of the longer latencies are due to long utterances that are not further broken down into smaller utterances.

| Name | Size |
|---|---|
| `pyannote/segmentation-3.0` | 5.91 MB |
| `pyannote/wespeaker-voxceleb-resnet34-LM` | 26.6 MB |
| MossFormer2 | 223.48 MB |
| `Systran/faster-whisper-large-v3` | 3.09 GB |
| runtime environment | ∼650 MB |
| data during inference | ∼1 GB |

Table 5.3: description

Table 5.3 shows the VRAM usage of the pipeline. In total, about 4 GB of VRAM is occupied by the loaded models and their runtime environments, and an additional 1 GB of VRAM is required for the inference data. This does not include the memory required for inference of the speech separation model, MossFormer2, which may require more memory than the GPU can provide for long utterances. The input length for speech separation inference had to be limited to less than 7.63 seconds to avoid out-of-memory errors in the GPU. In these cases, the unseparated audio was used for further pipeline steps as if the target speaker had been extracted. Although the speech separation step can occupy all of the VRAM, this is only for the short duration of the inference itself. Thus, most of the time, at most 5 GB of VRAM is needed for the pipeline. Outside of the GPU, about 2.2 GB of RAM is needed.

### 5.3.2. Diarization and Transcription Quality Analysis

To evaluate the diarization and transcription quality of the pipeline, the metrics diarization error rate (DER), word error rate (WER), word diarization error rate (WDER) and concatenated minimum-permutation word error rate (cpWER) are computed on each file of the respective test split of a dataset. After each file, all buffers are cleared and the pipeline is reset to not carry over information from the previous file. The error rates are reported as averages over all files. Since the files are of different lengths, they must be weighted accordingly. There are two options, weighting by the total audio or the total speech length of a file. The error difference between both options is negligible, for example the WER of the pipeline on the LibriSpeech test set is 3.88% when weighted by total audio length and 3.78% when weighted by total speech length. In order not to overestimate the diarization and transcription quality of the methods, the values reported below are always weighted by the length of the audio, as this showed the worse error rates in most cases of all methods.

#### Single-Speaker Scenario

The quality of diarization and transcription is evaluated in three steps, which are aligned with the research goals stated in section 1. For the first and third steps, the pipeline is compared to alternative methods described in section 5.2, which serve as a baseline even though they may not be comparable in all aspects. The first goal was to match the transcription quality of a previous method for single-speaker speech. The method used as a baseline is a combination of Silero VAD and Whisper, which does not provide speaker information and therefore solves only a subset of the tasks that the proposed pipeline attempts to solve.

To get an accurate comparison, the pipeline could be modified to perform only voice activity detection instead of full diarization, or the clustering threshold could be raised so that all speech is assigned to the same speaker cluster. However, this would result in two slightly different pipeline architectures or two different sets of selected pipeline parameters, skewing the comparison. It would omit potential errors in assigning words of speech from one speaker to two detected speakers in the form of speaker confusion. Therefore, the experiment is treated as a multi-speaker experiment with only one true speaker in the ground truth, allowing the proposed pipeline to make speaker confusion errors. The output of the baseline method is labeled as coming from a single speaker, which means that the baseline cannot make speaker assignment errors on the single-speaker dataset.

Figure 5.7 shows the evaluation results for the proposed pipeline and the baseline method on the LibriSpeech and Monologue datasets. The values shown are the mean for the different metrics, and the error bars are the standard deviation. For all metrics, the results of the proposed pipeline are very similar to the baseline method. The diarization error rate for the proposed pipeline is higher than the baseline, partly due to differences in voice detection sensitivity. The voice activity segments produced by Silero VAD are closer to the expected segments in the ground truth, while the proposed pipeline includes more margin around utterances due to more gradual changes in speaker probabilities, which can lead to merged utterances. Although this affects word-level latencies, it does not necessarily degrade transcription quality, but is shown as an additional diarization error in this duration-based metric. However, there are occasional cases of short utterances where the segmentation threshold is too high to capture the utterance, meaning

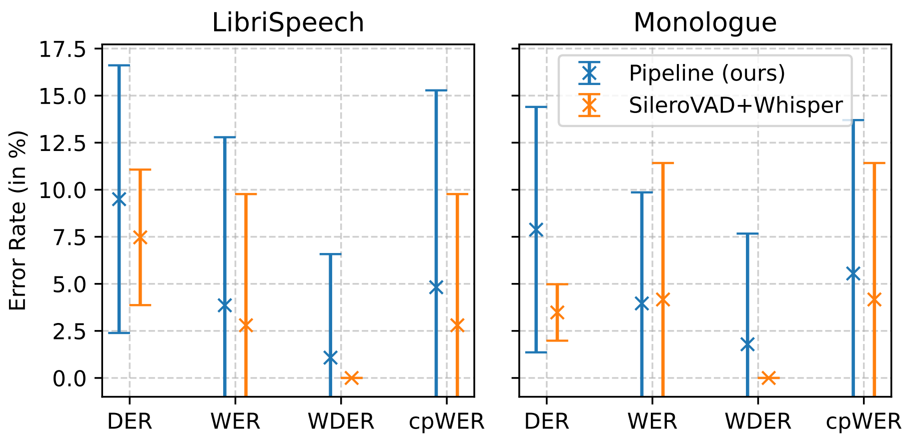

Figure 5.7: Evaluation results for single-speaker datasets are shown for the proposed pipeline and a baseline method. The metrics show similar values between both methods.

that the words of these utterances are missed. Similarly, some of the diarization error is caused by speaker confusion, where some of the sampled utterances were different enough to be detected as two separate speakers. These differences are due to variations in speech expression, where the characteristics of the voice change.

The word error rate for both methods is about 3 to 4%, which is limited by the transcription model used, Whisper, which has a WER of 3.33% on the LibriSpeech test set. Common errors include misspellings of names, differences between British and American spelling, contractions, abbreviations and rarely used vocabulary, which are not covered by the text normalization. The average word diarization error rate for the proposed pipeline is low, with about 1% of words incorrectly assigned to an additional speaker. As mentioned above, the transcribed words for the baseline are all assigned to one speaker, so the WDER is zero and the results for the cpWER are not different from the WER. The cpWER for the proposed pipeline is slightly worse than the WER, at about 5%, because some of the correctly recognized words were assigned to the wrong speaker.

### Multi-Speaker Scenario without Overlapping Speech

The next research goal was to provide speaker information for the generated transcriptions. To test this, the proposed pipeline is evaluated on the Conversation dataset without speaker overlap. Figure 5.8 shows these results in the same way as the previous figure. Comparing the results of the proposed pipeline on the monologue results with the multi-speaker results, the WER increased by 0.75 percentage points to 4.71%, remaining relatively similar for both methods. Meanwhile, the

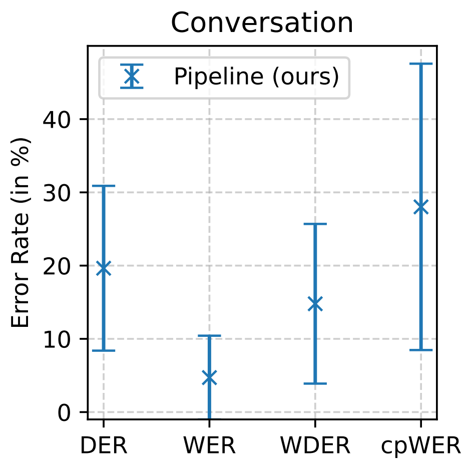

Figure 5.8: Evaluation results are shown for the proposed pipeline on the generated multi-speaker no-speech-overlap dataset Conversation. While maintaining a relatively low WER, speaker assignment errors are increased, resulting in an increased error rate in the multi-speaker WER metric.

WDER increased from 1.79% to almost 15%, showing that a significant proportion of words are misclassified. This can also be seen in the confusion component of the DER, which accounts for 75% of the total DER, doubling the DER compared to the monologue dataset, which is now about 20%. Since the cpWER is a combination of diarization and transcription error, this increased diarization error is also reflected in the cpWER, which has risen to about 28%.

The increased speaker confusion error can be caused by local speaker diarization or speaker recognition. A new error for the multi-speaker scenario is that some utterances are not split at a speaker change, resulting in a large utterance that can only be assigned to one speaker. This merged utterance can either be assigned to one of the speakers of the contained utterances or to a new speaker that is different from any of the speakers. In the latter case, the entire utterance contributes to the error, but in the former case, only the misassigned part contributes to the error. The origin of the error is the local speaker diarization from the segmentation model, which did not detect the speaker change and continued to assign the same local speaker. A workaround would be to increase the segmentation threshold to get finer grained, shorter utterances and let the speaker recognition handle the speaker change. Alternatively, the segmentation output for the local speaker diarization needs to be improved to better capture these speaker changes or individual speaker probabilities.

Most of the speaker confusion errors are due to erroneous assignments during speaker recognition. Two common errors are either assigning an utterance to the same speaker as the immediately preceding utterance, or assigning an utterance to a completely new speaker, even though the speaker has already been established by a previous utterance. The cause for this is that the clustering threshold is set too low or too high, depending on the case, but the underlying reasons are speaker embeddings that do not capture speech characteristics well enough to distinguish between speakers or to be clustered correctly. In addition, the former case can have a lasting effect on subsequent utterances, as the speaker cluster centroid is updated based on both speakers, making it more likely to be the assignment winner for utterances of both speakers. While this is more likely to happen on the first utterance of each speaker, it can also happen after both speakers have been established. In both cases, no specific speaker or speaker combination could be identified that consistently causes the error, meaning that it is not due to the general characteristics of a speaker's voice, but to the variations in expression during the utterance.

Lastly, the reason for the increased standard deviations is that the diarization and transcription outputs for some files contain no errors at all, and some files contain multiple errors, some of which are caused by previous errors in speaker recognition. While the overall WER remains low, the increased difficulty of multiple speakers causes errors in speaker assignments due to imperfect speaker recognition. In some cases, this results in degraded multi-speaker error rates.

### Multi-Speaker Scenario with Overlapping Speech

For the third goal of improving speech recognition on overlapping speech, the proposed pipeline and two baseline methods are evaluated on the multi-speaker datasets with overlapping speech Conversation with Overlap and AMI Corpus. The first baseline method is Whisper combined with speaker lookup from a pyannote diarization and the second baseline method is Whisper with speech segment clustering. Note that a direct comparison is difficult due to the lack of similar methods, as explained in section 5.2

Figure 5.9 shows that overall, no method provides good multi-speaker transcription results, as all methods have cpWER scores averaging between 30 and 60%. Whisper with speaker lookup has the lowest errors for most metrics on both datasets, which is to be expected, since it also has access to more information than the proposed pipeline and the other baseline method. The standard deviations are lower for results on the AMI Corpus dataset because each file is much longer, so sections of different levels of difficulty in the files compensate for each other. Also, since speaker recognition errors can affect subsequent diarization efforts, the methods are less likely to produce perfect results on longer files where these effects

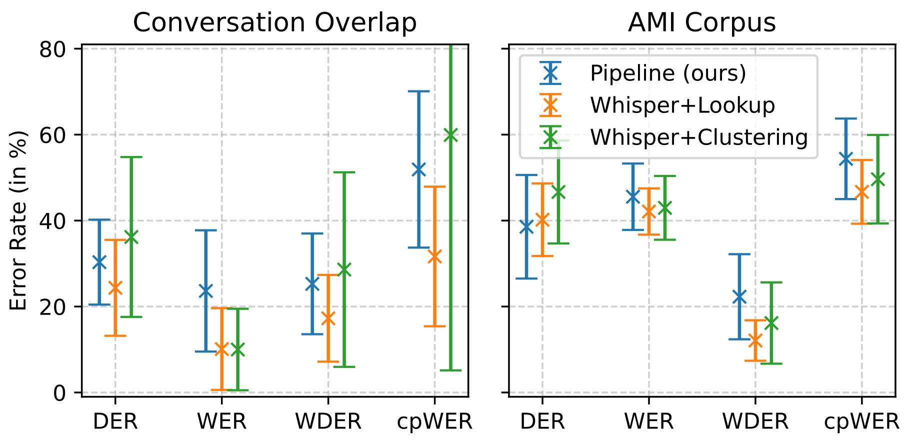

Figure 5.9: Evaluation results for multi-speaker datasets with overlapping speech are shown for the proposed pipeline and two baseline methods. The additional complexity of overlapping speech leads to an increased error compared to the non-overlapping datasets. No method provides good multi-speaker transcriptions with error rates between 30 to 60%.

are more likely to occur.

Comparing the normal Conversation dataset with the Conversation dataset with overlap, the proposed pipeline shows a general increase in error rates across all metrics of 10 to 20 percentage points. For the Conversation dataset with overlap, 76% of the diarization error comes from the speaker confusion component, while for the AMI Corpus dataset, the missed detection and speaker confusion components are equally high. A higher proportion of missed recognition also means that more words are missed in the transcription, which can be observed in the significantly higher WER.

While many of the previously discussed error cases remain, overlapping speech and the speech separation step add additional complexity and potential sources of error. In contrast to the previous datasets, the conversations in the AMI corpus contain, on the one hand, longer sections of utterances by one speaker, which often contain short interjections, and, on the other hand, sections with many speaker changes in short succession. Most of the short interjections like ”um” or ”hmm” are already missed by the diarization step, which may be due to missing labels in the training data used to train the segmentation model. In addition, these interjections can be easily misspelled, such as ”okay” instead of ”’kay”, leading to increased speech recognition errors.

In the case of overlapping speech, the proposed pipeline attempts to create isolated audio by separating the speech of different speakers. This is not always successful, which can lead to duplicate transcriptions if the wrong speaker is identified as the target, or missing transcriptions if only parts of the utterance are separated or are noisy due to artifacts from the separation model. Unseparated speech also increases the likelihood of speaker confusion errors, not only for that utterance, but also for subsequent utterances.

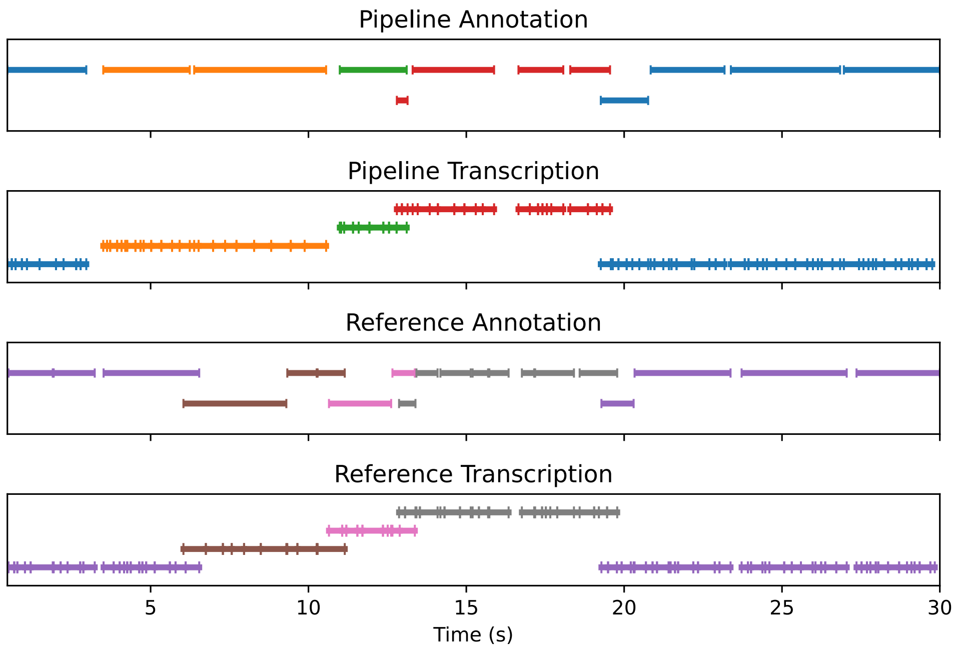

Figure 5.10: An example of the proposed pipeline successfully transcribing overlapping speech from the Conversation with Overlap dataset file *222*. The words of the overlapping utterance are still correctly recognized and assigned to the correct speaker.
**BLUE**: And as soon as I've had my coffee and oatmeal
**ORANGE**: ~~I'll~~ (I will) take him to the room of the great knife and patch him
**ORANGE**: Why should one not explore everything and study everything
**GREEN**: What I say is altogether on your own account
**RED**: In short, he becomes a prominent figure in London society
**RED**: And if he is not careful, somebody will say so
**BLUE**: When first they entered the throne room, they tried to be as haughty and scornful as ever
Note that the the second utterance contains an error due to a contraction and the third utterance was assigned to the wrong speaker.

The introduced speech separation step cannot reliably handle the additional complexity of overlapping utterances. Nevertheless, there are examples where speech separation and the other pipeline steps work well. Figure 5.10 shows an excerpt from file *222* of the Conversation with Overlap dataset. There are two errors in this excerpt, one utterance is assigned to the wrong speaker and ”I will” was transcribed as ”I’ll”. Note that since the utterances in the dataset are sampled, they have no semantic coherence between each other. Nevertheless, the overlapping words between the green and red speaker and the overlapping words between the red and blue speaker are correctly separated, assigned to the correct speaker, and correctly recognized despite the overlap. While the baseline methods are able to provide the transcriptions of the overlapping words in this example, both make mistakes in assigning them to the correct speakers. In the case of the red-to-blue overlap, their direct Whisper transcription added the words ”when first” to the end of the red utterance, resulting in the assignment error, but there are also cases where some of the overlapping words are simply omitted.

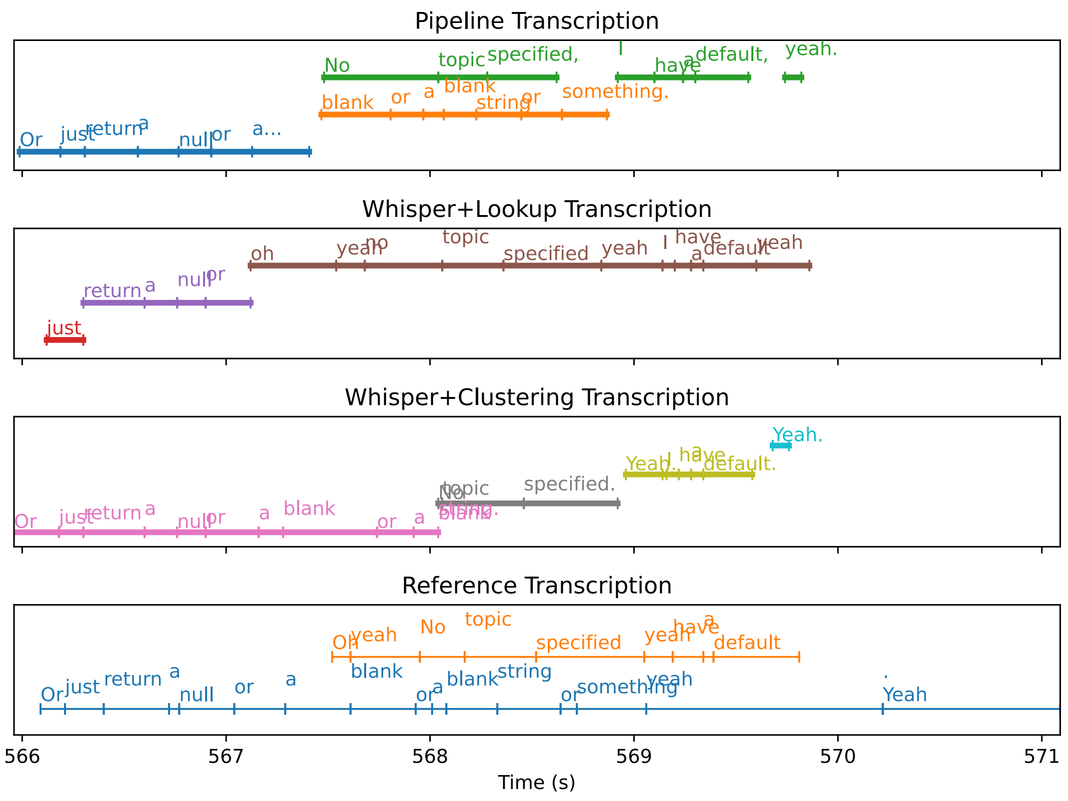

Figure 5.11: An example of transcriptions of overlapping utterances from the file *EN2002c* of the AMI Corpus dataset. Although not error-free, it shows that the pipeline successfully produces transcriptions for two fully overlapping utterances. Meanwhile, the baseline methods fail because they drop words during the overlap.

As a second example, Figure 5.11 shows the transcriptions for all methods of an excerpt from the AMI Corpus dataset file *EN2002c*. The orange utterance in the reference is fully overlapping with the blue utterance. The proposed pipeline recoveres most words, only omitting some interjections and failing to continue with the correct speaker during the overlap. Apart from speaker assignemnt errors, both baseline methods only produce sequential words The first baseline method is entirely dropping the second halve of the first utterance and the second baseline method drops some words and assigns zero length durations to words at their detected speaker change.

Omitting words in case of overlapping speech is a common issue with the direct transcription from Whipser. In cases of less extreme overlap, it is common to squeeze the timestamps of the words at the end and beginning of the overlapping utterances to create seemingly non-overlapping transcriptions. Whisper groups words in segments, which are similar to utterances, but often do not exactly match the correct moment for the speaker change, which leads to the speaker assignment error in the first example. The Whisper model is intended to transcribe speech independend of speakers and is not incentivized during training to accustom separation of words by their speaker. Additionally, the training data may not include many examples of overlapping speech or interjections, as these may not be desired in trnascriptions for general use cases. Therefore, the Whisper transcription on its own or as the basis for further processing steps is due to its design and training not able to fully to solve the challange of speech recognition of overlapping speech.

Overall, the proposed pipeline is able to provide transcriptions with speaker information for single and multi-speaker scenarios, although it loses some accuracy compared to approaches that focus only on single-speaker speech. The diarization step sometimes fails to capture short utterances such as interjections. In addition, speaker recognition can be unreliable when a speaker's voice changes and struggles with overlapping speech. The latency does not yet reach the human level of less than one second, partly due to increased processing time for longer utterances and partly due to selected models and parameters of the pipeline. Two performance issues have been identified that, if optimized, would allow for pipeline parameters with shorter artificial latencies. Similarly, selecting shorter utterances would not only result in shorter processing times for the speech separation and recognition steps, but would also improve the overall word-level latency.

# 6. Discussion

After evaluating the performance of the proposed pipeline, this chapter discusses the results. First, our results are compared to those of existing methods and put into the context of robotics tasks. Then, shortcomings and future work are discussed.

For the first method, since the diarization component for WhisperX was added after the publication of the paper and no additional data was published, a direct comparison is not possible. For the second method by Lyu et al. [13], WER and WDER results as well as latency measurements are available. They also provide WER and WDER results for the first method, which they also use as a baseline.

They test the methods on an unpublished dataset with two and three speakers alternating. They do not report on the severity of overlapping speech, or whether their data contains overlapping speech at all. They only state that Whisper, which is used in their method, "can accurately segment the speech of each speaker, even when they are speaking simultaneously" [13], which is contradictory to our results in section 5.3.2, which show words assigned to incorrect segments or omitted altogether. The closest data from our evaluation is the Conversation dataset, where four speakers speak without intentional overlap.

The reported WER results are between 16% and 26% for all combinations of two and three speakers and both methods [13], which is much higher than what we observed during our evaluation on the Conversation dataset, which was between 3% and 5%. The authors state that they did not focus on WER, and argue that the high WER is due to the fact that they used a smaller Whisper model to achieve lower latency and only used minimal text normalization in the evaluation.

For the WDER, they report results between 3% and 12% for the two and three speaker scenarios, with their method outperforming the baseline for the two speaker scenario as a significant improvement. On the other hand, when evaluating the WDER over the cumulative word count, they show that the initial error rate is as high as 25%, but settles down to 14% after a while, which is higher than their previously reported error. Our proposed pipeline has a WDER of about 15% on the four-speaker Conversation dataset with at least 23% WDER on our baseline implementations, which is closer to the second result mentioned. The most likely reason for these differences in evaluation results is the different data used for evaluation, as there are also large differences observed between the Conversation and AMI Corpus datasets.

For latency, they report the average length of their audio buffer before the audio belonging to the oldest transcribed segment in the buffer is processed to be 5.11 seconds, and the computational latency to be 0.13 seconds. As noted in section 3.2, they perform speech recognition on the audio buffer at each time step, which may contain multiple segments of Whisper transcriptions. Only when the number of segments exceeds a threshold of three or more, the oldest segment is processed to identify the speaker. Assuming that these segments are roughly equal in length, the processed segment is on average 1.7 seconds long and has an average artificial latency of 3.4 seconds in the remaining unprocessed buffer.

Compared to the proposed pipeline, this means that their method produces shorter utterances but higher artificial latency. Although the proposed pipeline has a much higher computational latency, averaging 0.8 seconds for transcribed utterances, the resulting utterance-level latency of 1.4 seconds is still lower due to the lower artificial latency of 0.6 seconds. The minimum word-level latency of the proposed pipeline is lower, at least for short utterances, but the average word-level latency is higher due to the overall longer utterances.

While their method is constrained by the already optimized computational time of Whisper inference and the fact that they already use a smaller and therefore faster model than we do, the proposed pipeline can reduce latency by selecting pipeline parameters that produce shorter utterances through increasing the diarization resolution. This may have a negative impact on the ability to detect low-confidence utterances and on the quality of speaker embeddings, since too short utterances may not contain enough information to represent the speaker. On the other hand, it would drastically reduce word-level latency by skipping time that is otherwise mostly spent waiting for the utterance to be processed. Shorter utterances also have the advantage that some models require less memory for inference, and inference time can be reduced.

## 6.1. Limitations

In the evaluation, the proposed pipeline was tested in a simulated mode and three simplifications were made. First, the audio was provided directly to the pipeline, which in a real scenario must first be captured by a microphone, then buffered, and then provided to the pipeline. Similarly, the latency evaluation only considers the processing of the pipeline itself, but for interaction with a human, the processing of a downstream task such as inference on a large language model and text-to-speech would also contribute to the response time of the system. While the latency of recording and buffering the audio can be implemented with negligible amounts of latency, additional processing time margins for pre- and post-processing must be considered for real-world applications.

Second, noise suppression was not included in the proposed pipeline and its evaluation. Noise suppression is a common pre-processing step in robotics and could be added before or with the loudness normalization. The datasets used are largely noise-free, but in real-world scenarios, noise from the robot and the environment is likely to be present.

Third, there were no other programs running in the background for other robotic tasks during the evaluation. This means that the resources listed in section 5.2 were fully allocated to the proposed pipeline. In a real scenario, computational resources need to be shared with other foundation models on a regular basis. For CPU utilization, this is not a problem because the pipeline is largely sequential and can be run on a single core. A limiting factor is the GPU memory used by the speech separation model, so that only even shorter overlapping utterances can be processed. Distributed computing would be one solution, but this may have an impact on latencies within the system.

## 6.2. Future Work

For seamless human interaction, the response time, the latency of the pipeline and other systems to output the response, must be less than one second or, at best, close to the average pause length of 275 ms [4] of natural conversations. This will require further optimizations, including efficient implementation of parts of the pipeline such as audio pre-processing to adjust loudness. Another solution might be to use smaller and faster models, sacrificing quality for reduced latency. This is a difficult tradeoff. For example, there are speech separation models that run faster or use less memory, but instead of truly separating speech, they only attenuate the different speakers, which means that speech recognition may pick up some of the attenuated words.

Part of the latency in the pipeline comes from the step size, which can be reduced, but at some point the processing time for each step is greater than the step size, resulting in a permanent backlog. Backlog can be reduced or avoided by deferring non-immediately needed processing, such as data cleanup, to later steps, using parallelization to perform streaming speaker diarization independently of speech separation and recognition, or dynamically increasing the step size. Offline methods usually take advantage of batching to reduce processing time, which is difficult to achieve in an online system because there is usually only one time step of data available, but in the case of backlog, multiple time steps could be batched at once.

As the evaluation has shown, the errors in one step of the pipeline often lead to errors in the downstream tasks. Therefore, it is important to reduce the individual errors for each step, but also to make the pipeline robust to errors. The baseline method Whisper with speaker lookup reduces the dependency by performing diarization and speech separation independently. However, it still depends on precise timestamps for both steps, which introduces errors. A common approach in machine learning is to build multi-task models. Not all steps can be combined in an end-to-end model, for example speaker recognition requires some memory component of all previously seen speaker characteristics to track the global dependencies between utterances. PixIT [52] is another model based on the advances of pyannote, which combines the segmentation model and speech separation to produce both outputs simultaneously, but at the time of development the speech separation showed the problems of too weakly attenuated words mentioned above.

In general, the pipeline can be further improved by updating the foundation models used when alternatives become available. New models may be of better quality, faster, or require less memory. Alternatively, models can be trained or fine-tuned to better meet the requirements of the real-time, multi-speaker, overlapping speech recognition task. For example, speech separation models are typically trained by mixing multiple speakers into a single mixture, then applying the separation model and comparing the output to the original individual signals with a signal-to-noise ratio metric. An additional loss function could be introduced based on the speech recognition results on the separated outputs. This would penalize the separation model for leaving words from other speakers in the separated output and force the separation model to focus on components in the audio signal that are important for successful speech recognition.

While the previously discussed methods directly process the input audio and provide diarization and transcription results, there are also post-processing approaches to improve the speaker assignments of other methods. DiarizationLM [53], which also provided the implementation for some of the metrics, is a system that prompts large language models with the existing diarized transcript in text form with the task of correcting words or word positions that appear incorrect. It is designed to correct common errors in the unreliable segmentation of utterances by Whisper, and therefore may be of less use to systems that do not rely on Whisper segmentation. However, it may be useful for correcting speaker assignments when utterances are inadvertently split due to changes in the speaker's voice.

# 7. Summary

In this thesis, a pipeline is proposed for joint speaker diarization and speech recognition in real-time environments using multiple foundation models. Unlike existing methods that use transcription-first approaches, the pipeline uses a diarization-first approach. Instead of adding speaker information as a post-processing step, the pipeline has access to speaker information and overlapping speech prior to the speech recognition step. This architecture allows the pipeline to perform additional pre-processing in the form of speech separation before the audio is transcribed. In addition, the costly inference of large speech separation or speech recognition models can be skipped if no speech overlap or silence is detected. Because the pipeline uses pre-trained foundation models and builds on their respective advances, both in research and in large-scale training, it can be conveniently improved with new state-of-the-art models as they become available.

In an evaluation, the proposed pipeline is compared to baseline methods modeled after existing methods and tested in single-speaker, multi-speaker, and overlapping multi-speaker settings from the LibriSpeech and AMI Corpus datasets. For the single-speaker scenario, the pipeline produces accurate diarization and transcription results with a cpWER of about 5%. Increasing the number of speakers still produces accurate transcriptions, but speaker assignment errors are introduced due to imperfect results from the segmentation and embedding models. The WDER increases from about 2% to 15% for the four-speaker overlap-free scenario, which is also observed for existing state-of-the-art methods.

For overlapping speech, examples are shown where the speech separation step helps to recover the correct transcriptions, while errors in the baseline methods are shown. Overall, however, overlapping speech dramatically increases the error rates, as some errors can either strongly influence individual pipeline steps or propagate to the processing results of later utterances. For the multi-speaker scenario with overlapping speech, neither the pipeline nor the baseline methods, consistently produce accurate results with average cpWER values between 30 and 60%. Despite the fact that the proposed pipeline uses online processing and therefore has less information available, it still matches the performance of the existing offline systems.

Evaluation of the pipeline's processing time shows that it is fast enough to run in real time. The average processing time for time steps that produce transcription output is 0.8 seconds. Combined with the artificial latency set by the chosen pipeline parameters, this results in an utterance-level latency of about 1.4 seconds. The word-level latency is higher, ranging from 1 to 6 seconds in most cases, due to the length of the utterances generated. While this falls short of human-level responsiveness, waiting 1.4 seconds after the end of a sentence is a feasible latency for applications in robotics and virtual assistants.

# A. Additional Data on Clustering and Speaker Embedding Models

Figure A.1 and Figure A.2 have been omitted in section 5.2 for the sake of brevity. They show the evaluation results with the two alternative speaker embedding models in the same way as Figure 5.1. These models performed significantly worse and were therefore not selected.

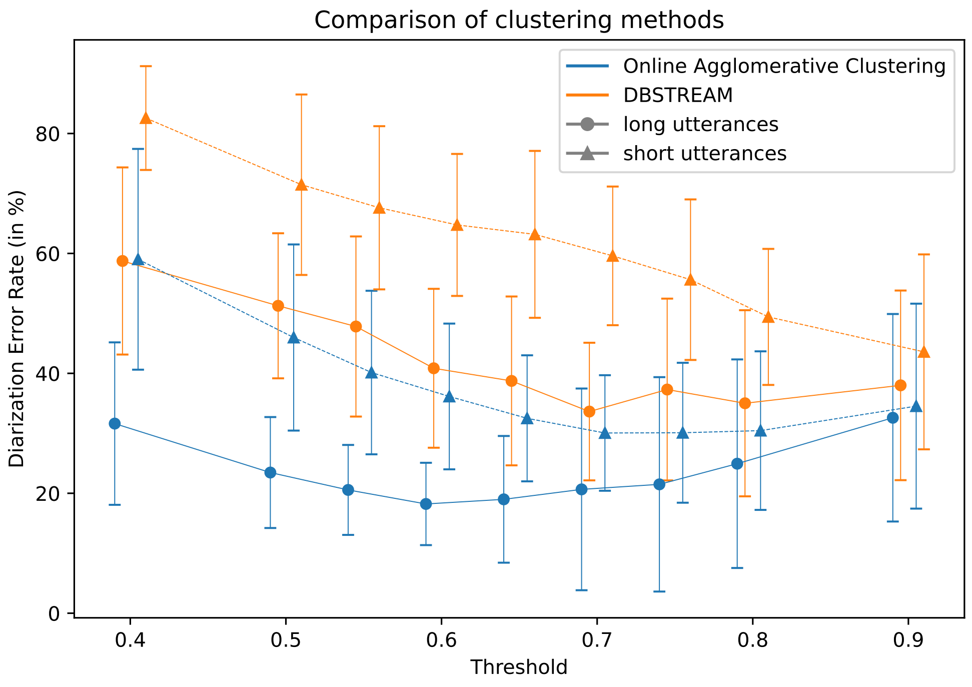

Figure A.1: A comparison of clustering methods with different clustering threshold settings. The same evaluation as in Figure 5.1, but with `pyannote/embedding` used as the speaker embedding model.

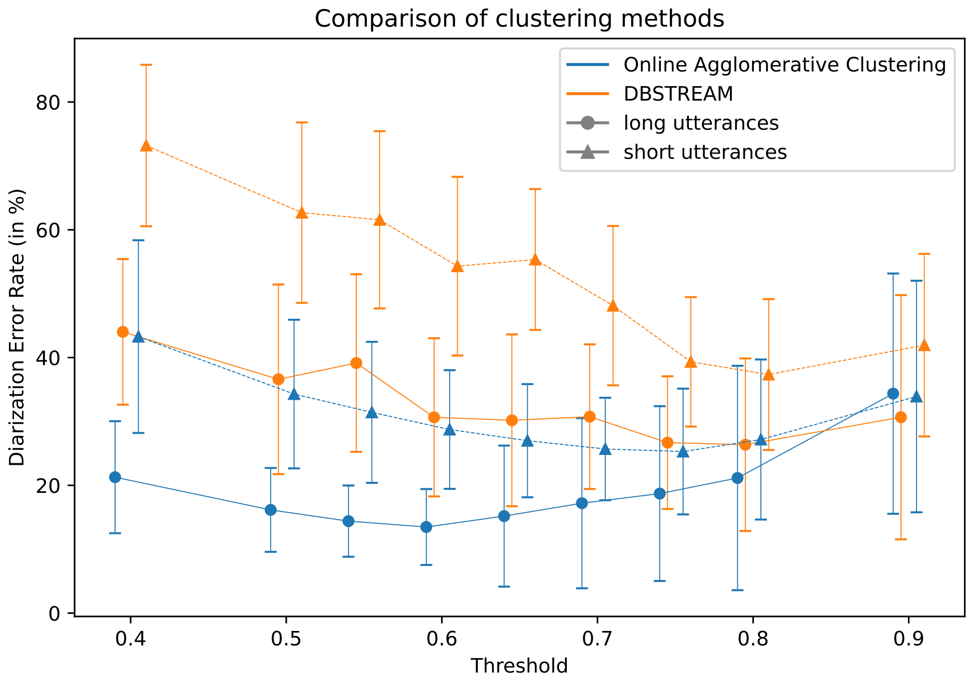

Figure A.2: A comparison of clustering methods with different clustering threshold settings. The same evaluation as in Figure 5.1, but with `speechbrain/spkrec-ecapa-voxceleb` used as the speaker embedding model.

# B. Evaluation Data

The tables below show the detailed statistics of the evaluation results sorted by dataset. Selections of these values are displayed during the evaluation in Figure 5.7, Figure 5.8 and Figure 5.9.

| Method | DER (in %) | | WER (in %) | | WDER (in %) | | cpWER (in %) | |
|---|---|---|---|---|---|---|---|---|
| | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ |
| Pipeline (ours) | 9.50 | 7.11 | 3.86 | 8.93 | 1.08 | 5.50 | 4.82 | 10.46 |
| SileroVAD+Whisper | 7.47 | 3.60 | 2.80 | 6.97 | 0.00 | 0.00 | 2.80 | 6.97 |
| WhisperLookup | 22.24 | 12.69 | 2.88 | 8.64 | 0.33 | 3.15 | 3.16 | 9.11 |
| WhisperClustering | 22.36 | 12.93 | 2.88 | 8.64 | 0.42 | 3.48 | 3.23 | 9.23 |

Table B.1: Results of various methods on the test set of the LibriSpeech [47] dataset.

| Method | DER (in %) | | WER (in %) | | WDER (in %) | | cpWER (in %) | |
|---|---|---|---|---|---|---|---|---|
| | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ |
| Pipeline (ours) | 7.88 | 6.52 | 3.96 | 5.90 | 1.79 | 5.88 | 5.55 | 8.15 |
| SileroVAD+Whisper | 3.48 | 1.50 | 4.17 | 7.25 | 0.00 | 0.00 | 4.17 | 7.25 |
| WhisperLookup | 13.18 | 10.11 | 4.56 | 8.65 | 1.78 | 6.28 | 5.68 | 9.89 |
| WhisperClustering | 12.27 | 8.51 | 4.43 | 8.50 | 0.34 | 2.58 | 4.09 | 8.77 |

Table B.2: Results of various methods on the Monologue dataset sampled from the test set of the LibriSpeech [47] dataset.

| Method | DER (in %) | | WER (in %) | | WDER (in %) | | cpWER (in %) | |
|---|---|---|---|---|---|---|---|---|
| | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ |
| Pipeline (ours) | 19.64 | 11.25 | 4.71 | 5.72 | 14.79 | 10.90 | 28.02 | 19.55 |
| WhisperLookup | 30.78 | 12.29 | 3.09 | 4.44 | 23.30 | 11.01 | 37.21 | 17.90 |
| WhisperClustering | 55.97 | 16.73 | 3.15 | 4.59 | 46.43 | 17.87 | 105.11 | 58.41 |

Table B.3: Results of various methods on the Conversation dataset sampled from the test set of the LibriSpeech [47] dataset.

| Method | DER (in %) | | WER (in %) | | WDER (in %) | | cpWER (in %) | |
|---|---|---|---|---|---|---|---|---|
| | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ |
| Pipeline (ours) | 30.33 | 9.88 | 23.63 | 14.11 | 25.28 | 11.72 | 51.90 | 18.19 |
| WhisperLookup | 24.35 | 11.16 | 10.11 | 9.51 | 17.26 | 10.09 | 31.65 | 16.24 |
| WhisperClust. | 36.18 | 18.60 | 10.00 | 9.48 | 28.60 | 22.64 | 59.92 | 54.77 |

Table B.4: Results of various methods on the Conversation dataset with overlapping segments sampled from the test set of the LibriSpeech [47] dataset.

| Method | DER (in %) | | WER (in %) | | WDER (in %) | | cpWER (in %) | |
|---|---|---|---|---|---|---|---|---|
| | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ |
| Pipeline (ours) | 38.56 | 12.05 | 45.55 | 7.73 | 22.28 | 9.90 | 54.37 | 9.37 |
| WhisperLookup | 40.20 | 8.45 | 42.11 | 5.38 | 12.09 | 4.71 | 46.68 | 7.41 |
| WhisperClustering | 46.66 | 11.98 | 42.96 | 7.42 | 16.16 | 9.46 | 49.63 | 10.29 |

Table B.5: Results of various methods on the test split of the AMI Corpus Mix-Headset dataset [39].

# List of Figures

# List of Tables

# Bibliography

[1] Justin Hart, Alexander Moriarty, Katarzyna Pasternak, Johannes Kummert, Alina Hawkin, Vanessa Hassouna, Juan Diego Pena Narvaez, Leroy Ruegemer, Leander von Seelstrang, Peter Van Dooren, Juan Jose Garcia, Akinobu Mitzutani, Yuqian Jiang, Tatsuya Matsushima, and Riccardo Polvara. *RoboCup@Home 2024: Rules and regulations.* 2024. URL: `https://github.com/RoboCupAtHome/RuleBook/releases/tag/2024.1`.

[2] R. Stiefelhagen, C. Fugen, R. Gieselmann, H. Holzapfel, K. Nickel, and A. Waibel. "Natural human-robot interaction using speech, head pose and gestures." In: *IEEE International Conference on Intelligent Robots and Systems (IROS).* Vol. 3. 2004, pp. 2422–2427.

[3] Adelbert Bronkhorst. "The cocktail party phenomenon: A review of research on speech intelligibility in multiple-talker conditions." In: *Acta Acustica united with Acustica* 86 (2000), pp. 117–128.

[4] Stephen C. Levinson and Francisco Torreira. "Timing in turn-taking and its implications for processing models of language." In: *Frontiers in Psychology* 6 (2015). ISSN: 1664-1078.

[5] Raphael Memmesheimer, Jan Nogga, Bastian Pätzold, Evgenii Kruzhkov, Simon Bultmann, Michael Schreiber, Jonas Bode, Bertan Karacora, Juhui Park, Alena Savinykh, and Sven Behnke. "RoboCup@Home 2024 OPL winner NimbRo: Anthropomorphic service robots using foundation models for perception and planning." In: *RoboCup 2024: Robot World Cup XXVII.* Springer, 2025. Forthcoming.

[6] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. "Robust speech recognition via large-scale weak supervision." In: *International Conference on Machine Learning (ICML).* JMLR.org, 2023.

[7] Max Bain, Jaesung Huh, Tengda Han, and Andrew Zisserman. "WhisperX: Time-accurate speech transcription of long-form audio." In: *Interspeech.* 2023, pp. 4489–4493.

[8] Hervé Bredin. "pyannote.audio 2.1 speaker diarization pipeline: Principle, benchmark, and recipe." In: *Interspeech.* 2023.

[9] Alexis Plaquet and Hervé Bredin. "Powerset multi-class cross entropy loss for neural speaker diarization." In: *Interspeech.* 2023, pp. 3222–3226.

[10] NVIDIA Corporation. *NVIDIA NeMo framework developer docs – speaker diarization.* 2024. URL: `https://docs.nvidia.com/nemo-framework/user-guide/latest/nemotoolkit/asr/speaker_diarization/intro.html` (visited on 07/10/2024).

[11] David Snyder, Daniel Garcia-Romero, Gregory Sell, Daniel Povey, and Sanjeev Khudanpur. "X-Vectors: Robust DNN embeddings for speaker recognition." In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* 2018, pp. 5329–5333.

[12] Brecht Desplanques, Jenthe Thienpondt, and Kris Demuynck. "ECAPA-TDNN: Emphasized channel attention, propagation and aggregation in TDNN based speaker verification." In: *Interspeech.* Ed. by Helen Meng, Bo Xu, and Thomas Fang Zheng. ISCA, 2020, pp. 3830–3834.

[13] Ke-Ming Lyu, Ren-yuan Lyu, and Hsien-Tsung Chang. "Real-time multilingual speech recognition and speaker diarization system based on Whisper segmentation." In: *PeerJ Computer Science* 10 (2024), e1973.

[14] Juan Manuel Coria, Hervé Bredin, Sahar Ghannay, and Sophie Rosset. "Overlap-aware low-latency online speaker diarization based on end-to-end local segmentation." In: *IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)* (2021), pp. 1139–1146.

[15] Henri J. Nussbaumer. "The fast Fourier transform." In: *Fast Fourier Transform and Convolution Algorithms.* Springer, 1981, pp. 80–111. ISBN: 978-3-662-00551-4.

[16] S. Davis and P. Mermelstein. "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences." In: *IEEE Transactions on Acoustics, Speech, and Signal Processing (TSP)* 28.4 (1980), pp. 357–366.

[17] C.E. Shannon. "Communication in the presence of noise." In: *Proceedings of the IRE* 37.1 (1949), pp. 10–21.

[18] Brian B. Monson, Eric J. Hunter, Andrew J. Lotto, and Brad H. Story. "The perceptual significance of high-frequency energy in the human voice." In: *Frontiers in Psychology* 5 (2014). ISSN: 1664-1078.

[19] S.E. Tranter and D.A. Reynolds. "An overview of automatic speaker diarization systems." In: *IEEE Transactions on Audio, Speech, and Language Processing (TASLPRO)* 14.5 (2006), pp. 1557–1565.

[20] Yusuke Fujita, Naoyuki Kanda, Shota Horiguchi, Kenji Nagamatsu, and Shinji Watanabe. "End-to-end neural speaker diarization with permutation-free objectives." In: *Interspeech.* 2019.

[21] Najim Dehak, Patrick J. Kenny, Réda Dehak, Pierre Dumouchel, and Pierre Ouellet. “Front-end factor analysis for speaker verification.” In: *IEEE Transactions on Audio, Speech, and Language Processing (TASLPRO)* 19.4 (2011), pp. 788–798.

[22] Michael Hahsler and Matthew Bolaños. “Clustering data streams based on shared density between micro-clusters.” In: *IEEE Transactions on Knowledge and Data Engineering* 28 (2016), pp. 1449–1461.

[23] D.R. Reddy. “Speech recognition by machine: A review.” In: *Proceedings of the IEEE* 64.4 (1976), pp. 501–531.

[24] Mirco Ravanelli and Yoshua Bengio. “Speaker recognition from raw waveform with SincNet.” In: *IEEE Spoken Language Technology Workshop (SLT)* (2018), pp. 1021–1028.

[25] Jonathan Fiscus, Jerome Ajot, Martial Michel, and John Garofolo. “The rich transcription 2006 spring meeting recognition evaluation.” In: 2006, pp. 309–322. ISBN: 978-3-540-32549-9.

[26] Laurent El Shafey, Hagen Soltau, and Izhak Shafran. “Joint speech recognition and speaker diarization via sequence transduction.” In: *Interspeech*. 2019.

[27] Shinji Watanabe, Michael Mandel, Jon Barker, Emmanuel Vincent, Ashish Arora, Xuankai Chang, Sanjeev Khudanpur, Vimal Manohar, Daniel Povey, Desh Raj, David Snyder, Aswin Shanmugam Subramanian, Jan Trmal, Bar Ben Yair, Christoph Boeddeker, Zhaoheng Ni, Yusuke Fujita, Shota Horiguchi, Naoyuki Kanda, Takuya Yoshioka, and Neville Ryant. “CHiME-6 challenge: Tackling multispeaker speech recognition for unsegmented recordings.” In: *International Workshop on Speech Processing in Everyday Environments*. 2020, pp. 1–7.

[28] Hervé Bredin. *GitHub repository of pyannote.audio*. 2025. URL: `https://github.com/pyannote/pyannote-audio` (visited on 02/20/2025).

[29] Hervé Bredin and Antoine Laurent. “End-to-end speaker segmentation for overlap-aware resegmentation.” In: *Interspeech*. 2021, pp. 3111–3115.

[30] 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.” In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2023, pp. 1–5.

[31] Hossein Zeinali, Shuai Wang, Anna Silnova, Pavel Matejka, and Oldrich Plchot. *BUT system description to VoxCeleb Speaker Recognition Challenge 2019*. 2019. arXiv: `1910.12592 [eess.AS]`.

[32] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 770–778.

[33] Mirco Ravanelli, Titouan Parcollet, Peter Plantinga, Aku Rouhe, Samuele Cornell, Loren Lugosch, Cem Subakan, Nauman Dawalatabad, Abdelwahab Heba, Jianyuan Zhong, Ju-Chieh Chou, Sung-Lin Yeh, Szu-Wei Fu, Chien-Feng Liao, Elena Rastorgueva, François Grondin, William Aris, Hwidong Na, Yan Gao, Renato De Mori, and Yoshua Bengio. *SpeechBrain: A general-purpose speech toolkit.* 2021. arXiv: `2106.04624 [eess.AS]`.

[34] Shang-Hua Gao, Ming-Ming Cheng, Kai Zhao, Xin-Yu Zhang, Ming-Hsuan Yang, and Philip Torr. "Res2Net: A new multi-scale backbone architecture." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 43.2 (2021), pp. 652–662. ISSN: 1939-3539.

[35] Shengkui Zhao, Yukun Ma, Chongjia Ni, Chong Zhang, Hao Wang, Trung Nguyen, Kun Zhou, Jia Yip, Dianwen Ng, and Bin Ma. "MossFormer2: Combining transformer and RNN-free recurrent network for enhanced time-domain monaural speech separation." In: 2024, pp. 10356–10360.

[36] Max Bain, Jaesung Huh, Tengda Han, and Andrew Zisserman. *GitHub repository of WhisperX.* 2025. URL: `https://github.com/m-bain/whisperX/` (visited on 02/20/2025).

[37] Harold W. Kuhn. "The Hungarian method for the assignment problem." In: *Naval Research Logistics (NRL)* 52 (1955).

[38] F.J. Harris. "On the use of windows for harmonic analysis with the discrete Fourier transform." In: *Proceedings of the IEEE* 66.1 (1978), pp. 51–83.

[39] Steve Renals, Thomas Hain, and Herve Bourlard. "Recognition and understanding of meetings the AMI and AMIDA projects." In: *IEEE Automatic Speech Recognition and Understanding Workshop (ASRU).* 2007, pp. 238–247.

[40] OpenAI et al. *GPT-4 technical report.* 2023. arXiv: `2303.08774 [cs.CL]`.

[41] OpenAI. *Hello GPT-4o.* 2024. URL: `https://openai.com/index/hello-gpt-4o/` (visited on 02/20/2025).

[42] SYSTRAN. *GitHub repository of faster-whisper.* 2025. URL: `https://github.com/SYSTRAN/faster-whisper/` (visited on 02/20/2025).

[43] Hugging Face, Inc. *Hugging Face website.* 2025. URL: `https://huggingface.co/` (visited on 02/20/2025).

[44] Alibaba Cloud. *Model Scope website.* 2025. URL: `https://modelscope.cn/` (visited on 02/20/2025).

[45] Jacob Montiel, Max Halford, Saulo Martiello Mastelini, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, Adil Zouitine, Heitor Murilo Gomes, Jesse Read, Talel Abdessalem, and Albert Bifet. "River: Machine learning for streaming data in Python." In: *Journal of Machine Learning Research (JMLR)* 22.1 (2021). ISSN: 1532-4435.

[46] Google. *GitHub repository of DiarizationLM.* 2025. URL: `https://github.com/google/speaker-id/tree/master/DiarizationLM/` (visited on 02/20/2025).

[47] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. "LibriSpeech: An ASR corpus based on public domain audio books." In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* 2015, pp. 5206–5210.

[48] *LibriVox: Free public domain audiobooks.* 2025. URL: `https://librivox.org/` (visited on 02/20/2025).

[49] Corentin Jemine. *GitHub repository LibriSpeech Alignments.* 2025. URL: `https://github.com/CorentinJ/librispeech-alignments/` (visited on 02/20/2025).

[50] Michael McAuliffe, Michaela Socolof, Sarah Mihuc, Michael Wagner, and Morgan Sonderegger. "Montreal Forced Aligner: Trainable text-speech alignment using Kaldi." In: *Interspeech.* 2017, pp. 498–502.

[51] Silero Team. *Silero VAD: pre-trained enterprise-grade voice activity detector (VAD), number detector and language classifier.* 2024. URL: `https://github.com/snakers4/silero-vad` (visited on 02/20/2025).

[52] Joonas Kalda, Clément Pagés, Ricard Marxer, Tanel Alumäe, and Hervé Bredin. "PixIT: Joint training of speaker diarization and speech separation from real-world multi-speaker recordings." In: *Odyssey.* 2024.

[53] Quan Wang, Yiling Huang, Guanlong Zhao, Evan Clark, Wei Xia, and Hank Liao. "DiarizationLM: Speaker diarization post-processing with large language models." In: *Interspeech.* 2024, pp. 3754–3758.