OVERLAP-AWARE LOW-LATENCY ONLINE SPEAKER DIARIZATION BASED ON END-TO-END LOCAL SEGMENTATION

Juan Manuel Coria, Hervé Bredin, Sahar Ghannay, Sophie Rosset

To cite this version:

Juan Manuel Coria, Hervé Bredin, Sahar Ghannay, Sophie Rosset. OVERLAP-AWARE LOW-LATENCY ONLINE SPEAKER DIARIZATION BASED ON END-TO-END LOCAL SEGMENTATION. IEEE Automatic Speech Recognition and Unserstanding Workshop, Dec 2021, Cartagena, Colombia. hal-03375330

HAL Id: hal-03375330
https://hal.science/hal-03375330
Submitted on 12 Oct 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
OVERLAP-AWARE LOW-LATENCY ONLINE SPEAKER DIARIZATION BASED ON END-TO-END LOCAL SEGMENTATION

Juan M. Coria\textsuperscript{1}, Hervé Bredin\textsuperscript{2}, Sahar Ghannay\textsuperscript{1}, Sophie Rosset\textsuperscript{1}

\textsuperscript{1}Université Paris-Saclay CNRS, LISN, Orsay, France
\textsuperscript{2}IRIT, Université de Toulouse, CNRS, Toulouse, France
\textsuperscript{1}\{juan-manuel.coria, sahar.ghannay, sophie.rosset\}@lisn.upsaclay.fr
\textsuperscript{2}herve.bredin@irit.fr

ABSTRACT
We propose to address online speaker diarization as a combination of incremental clustering and local diarization applied to a rolling buffer updated every 500ms. Every single step of the proposed pipeline is designed to take full advantage of the strong ability of a recently proposed end-to-end overlap-aware segmentation to detect and separate overlapping speakers. In particular, we propose a modified version of the statistics pooling layer (initially introduced in the x-vector architecture) to give less weight to frames where the segmentation model predicts simultaneous speakers. Furthermore, we derive cannot-link constraints from the initial segmentation step to prevent two local speakers from being wrongfully merged during the incremental clustering step. Finally, we show how the latency of the proposed approach can be adjusted between 500ms and 5s to match the requirements of a particular use case, and we provide a systematic analysis of the influence of latency on the overall performance (on AMI, DIHARD and VoxConverse).

Index Terms— speaker diarization, low latency, overlapped speech detection, speaker embedding

1. INTRODUCTION
Speaker diarization aims at answering the question “who spoke when”, effectively partitioning an audio sequence into segments with a particular speaker identity. Most dependable diarization approaches consist of a cascade of several steps \cite{1 2}: voice activity detection to discard non-speech regions, speaker embedding \cite{3 4} to obtain discriminative speaker representations, and clustering \cite{2 3 6} to group speech segments by speaker identity. The main limitation of this family of multi-stage approaches relates to how they handle overlapped speech (which is known to be one of the main sources of errors): either they simply ignore the problem or they address it \textit{a posteriori} as a final post-processing step based on a dedicated overlapped speech detection module \cite{7 8 9 10}. A new family of approaches have recently emerged, rethinking speaker diarization completely. Dubbed end-to-end diarization (EEND), the main idea of this approach is to train a single neural network – in a permutation-invariant manner – that ingests the audio recording and directly outputs the overlap-aware diarization output \cite{11 12}. We propose to meet half-way between multi-stage and overlap-aware end-to-end diarization and design a multi-stage pipeline where overlapped speech is a first-class citizen in every single step: from segmentation to incremental clustering. In particular, our first contribution (discussed in Section \textsuperscript{2.2.1}) is a modified version of the statistics pooling layer (initially introduced in the x-vector architecture) to give less weight to frames where the initial segmentation step predicts simultaneous speakers.

Despite being competitive with multi-stage approaches, the main limitation of the overlap-aware end-to-end approaches is the strong assumption that the number of speakers is upper bounded or even known \textit{a priori}. While reasonable for some particular use cases (e.g. one-to-one phone conversations), this assumption does not hold in many other situations (e.g. physical meetings or conference calls). One solution to this problem is to augment end-to-end approaches with mechanisms to automatically estimate the number of speakers. For instance, EEND-EDA \cite{13} extends EEND \cite{11 12} with a recurrent Encoder-Decoder network to generate a variable number of Attractors – similar to speaker centroids. Multi-stage approaches usually do not suffer from this limitation as they rely on a clustering step for which a growing number of techniques exist to accurately estimate the number of speakers \cite{14}. We propose to combine the best of both worlds \cite{15} by first applying the end-to-end approach on audio chunks small enough to reasonably estimate an upper bound on the local number of speakers and, then only, apply global constrained clustering on top of the resulting local speakers. As discussed in Section \textsuperscript{2.2.2}, we say that cluster-
ing is constrained because cannot-link constraints are inferred from the output of the local end-to-end diarization. The main difference between this work and [15] is that we target low-latency online speaker diarization while they address offline speaker diarization.

This work relies heavily on the speaker segmentation model introduced in [10] and summarized in Section 2.1 for convenience. However, they address two very different problems with radically different constraints. While [10] performs local offline speaker diarization of extremely short 5s chunks of audio, this work addresses online speaker diarization of (possibly infinite) audio streams. Hence, this work extends [10] with a mechanism to track speakers over the duration of a conversation, with a latency much lower than 5s and real-time processing.

Low-latency online speaker diarization differs from its offline counterpart in several ways. While the latter assumes that the whole audio sequence is available at once (and hence can rely on multiple passes over the whole sequence to output its final prediction), the former ingests a possibly infinite audio stream and can only afford a short delay between when it receives a buffer of audio and when it outputs the corresponding prediction (without the option to correct it afterwards). These additional constraints prevent state-of-the-art multi-stage approaches like VBx [16] from being used in that setting as they heavily rely on the possibility to pass several times over the audio sequence. EEND-like approaches are not suitable either because they expect large chunks of audio (30 seconds or more), leading to prohibitively high latency. One notable exception is FlexSTB [17] that astutely relies on an adaptive internal buffer to both simulate large audio chunks and support low (1s) latency.

A comprehensive set of experiments on AMI, DIHARD II, DIHARD III and VoxConverse datasets is reported and discussed in Section 4 – where FlexSTB and a state-of-the-art offline approach based on VBx [16] respectively serve as baseline and topline. In particular, we show how the latency of the proposed approach can easily be adjusted (without retraining) between 500ms and 5s to match the requirements of a particular use case.

2. OVERLAP-AWARE ONLINE DIARIZATION

As depicted in Figure 1, we propose to address online speaker diarization as the iterative interplay between two main steps: segmentation and incremental clustering. Every few hundred milliseconds (500ms in our case), the segmentation module first performs a fine-grained overlap-aware diarization of a 5s rolling buffer. This local diarization is then ingested by the incremental clustering module that relies on speaker embeddings to map local speakers to the appropriate global speakers (or create new ones), before updating its own internal state.

Fig. 1. Block diagram of the proposed approach.

2.1. Segmentation

The segmentation step is the direct application of the end-to-end speaker segmentation neural network introduced in [10], used to obtain a fine-grained local speaker diarization. As depicted in Figure 1, it ingests the 5s audio rolling buffer and outputs speaker activity probabilities \( \{s_1, \ldots, s_F\} \) where \( F \) is the number of output frames and \( s_f \in [0, 1]^{K_{\text{max}}} \), with \( K_{\text{max}} \) the estimated maximum number of different speakers that a 5s chunk may contain (\( K_{\text{max}} = 4 \) in our case). Speakers whose activity probability exceeds a tunable threshold \( \tau_{\text{active}} \) at least once during the chunk constitute the set of local speakers. Inactive speakers are simply discarded.

Active speaker probabilities are then passed unchanged (i.e. with continuous values between 0 and 1) to the incremental clustering step. In particular, it means that overlapping speech (i.e. when two or more speakers have high probabilities simultaneously) is handled from the very beginning of the pipeline. This is in contrast with most dependable speaker diarization approaches that handle overlapping speech as a post-processing step [6, 10]. This early detection of overlapping speech will prove very useful for the incremental clustering.

2.2. Incremental clustering

Because the segmentation model is trained in a permutation-invariant manner and applied locally to the rolling buffer, one cannot guarantee that one particular speaker consistently activates the same index over time. Figure 2 illustrates this lim-
ituation for two states of the rolling buffer: despite being only 500ms apart from each other and therefore having most of their audio content in common, notice how both active speakers are swapped. This section describes how we use incremental clustering to circumvent this limitation by tracking speakers (and detecting new ones) over the whole duration of the audio stream.

![Fig. 2. Actual output $s_f$ of the segmentation model on two consecutive positions of the 5s rolling buffer in file DH_DEV_0001 from DIHARD III. Because of permutation-invariant training, green and red speakers are swapped.](image)

### 2.2.1. Segmentation-driven speaker embedding

Like most recent speaker diarization systems, we rely on neural speaker embeddings to represent and compare speakers. Our model is based on the canonical x-vector TDNN-based architecture, with the difference that the statistics pooling layer \(^{[4]}\) is modified to return the concatenation of weighted mean $\mu_k$ and weighted standard deviation $\sigma_k$ for each active speaker $k –$ instead of the regular mean $\mu$ and standard deviation $\sigma$:

$$
\begin{align*}
\mu &= \frac{\sum_f w_{fk} \cdot x_f}{F} & \mu_k &= \frac{\sum_f w_{fk} \cdot x_f}{\sum_f w_{fk}} \\
\sigma^2 &= \frac{\sum_f (x_f - \mu)^2}{F - 1} & \sigma_k^2 &= \frac{\sum_f w_{fk} \cdot (x_f - \mu_k)^2}{\sum_f w_{fk}} - \frac{\sum_f w_{fk}^2}{\sum_f w_{fk}}
\end{align*}
$$

(1)

where $x_f$ is the output of frame $f$ of the last TDNN layer. One straightforward option is to derive $w_{fk}$ from the speaker activity probability and use $w_{fk} = s_{fk}$ directly, so that the final (pooled) speaker embedding mostly relies on frames where the segmentation model is confident that speaker $k$ is active. This generates exactly one embedding per active speaker in the current buffer, even when split into multiple speech turns (e.g. the red speaker in the lower row of Figure 2).

Furthermore, as summarized in \(^{[10]}\), the segmentation model is also very good at detecting overlapped speech regions (where two or more speakers are active simultaneously). Therefore, another option is to make the speaker embedding focus on frames where it is confident that speaker $k$ is the only active speaker:

$$
W_f = \left( s_f \cdot \text{softmax}(\beta \cdot s_f) \right)^\gamma
$$

(2)

where the effect of this transformation is illustrated in Figure 3. The use of $\text{softmax}_k$ weights down frames where two or more speakers are active, and the exponent $\gamma > 1$ weighs down frames where the segmentation model is not quite confident about the activity of a speaker. Embeddings extracted with this weighing scheme are called overlap-aware speaker embeddings in the rest of the paper.

![Fig. 3. Effect of Equation 2 with $\gamma = 3$ and $\beta = 10$. Top row is the actual output $s_f$ of the segmentation model on a 5s excerpt of file DH_DEV_0007 from DIHARD III. Bottom row depicts $w_f$ used for statistics pooling. Both low confidence and overlapped speech regions are weighed down.](image)

### 2.2.2. Constrained incremental clustering

Given the initial content of the rolling buffer, the segmentation and embedding steps are combined to extract one embedding for each active speaker in the first 5s of the audio stream. These speaker embeddings $\{c_1, \ldots, c_K\}$ are stacked to form the initial centroid matrix $C$ with shape $K \times D$ where $K$ is the number of active speakers so far, and $D$ is the dimension of the speaker embedding.

Every few hundred milliseconds (e.g. 500ms), the rolling buffer is updated, and the segmentation and speaker embedding steps are combined to extract one embedding for each of the $K_{\text{buffer}} \leq K_{\text{max}}$ locally active speakers. Those $K_{\text{buffer}}$ speaker embeddings are then compared to the current state of the centroid matrix $C$ to find the optimal mapping $m^*$ between local and global speakers. Denoting $d(c,e)$ the distance between centroid $c$ and local speaker embedding $e$, one option is to assign the $k$th local speaker to the closest centroid:

$$
m^*(k) = \arg\min_{c \in C} d(c, e_k)
$$

(3)

Yet, this simple option does not take full advantage of the output of the segmentation model, as two local speakers might end up being assigned to the same centroid. This would be in contradiction to the output of the segmentation model that already chose to discriminate local speakers. Therefore, we add the constraint that any two local speakers cannot be assigned to the same centroid, while keeping the objective of minimizing the overall distance between local speakers and their assigned centroids:

$$
m^* = \arg\min_{m \in M} \sum_{k} d(m(k), e_k)
$$

(4)
where \( \mathcal{M} \) is the set of mapping functions between local speakers and centroids with the following property:

\[
    k \neq k' \implies m(k) \neq m(k') \quad (5)
\]

In practice, this optimal mapping is obtained by applying the Hungarian algorithm on the speaker-to-centroid distance matrix, and can be seen as an incremental clustering step with cannot-link constraints.

### 2.2.3. Detecting new speakers and updating centroids

Once the optimal mapping \( m^* \) is determined, for any given local speaker \( k \) and their local embedding \( e_k \):

- if \( d(m^*(k), e_k) > \delta_{\text{new}} \), they are marked as new speaker (i.e., it is the first time they are active since the beginning of the audio stream) and their embedding \( e_k \) is appended to the pool of centroids:

\[
    C \leftarrow C \cup e_k
\]

- otherwise, they are marked as returning speaker, and their embedding \( e_k \) is used to update the corresponding centroid.

Because of the weighing scheme described in Section 2.2.1, the quality of a speaker embedding \( e_k \) is expected to be positively correlated with the estimated duration during which local speaker \( k \) is active: \( \Delta_k = \sum_f s_{fk} \). Therefore, we propose to only update a centroid when this duration is long enough:

\[
    c_{m^*(k)} \leftarrow \begin{cases} 
    c_{m^*(k)} + e_k & \text{if } \Delta_k > \rho_{\text{update}} \\
    c_{m^*(k)} & \text{otherwise}
    \end{cases} \quad (6)
\]

where \( \rho_{\text{update}} \) is the minimum duration below which a speaker embedding \( e_k \) is considered to be too noisy to help refine the centroid. Equation 6 assumes that speaker embeddings \( e_k \) are unit-normalized and optimized for cosine similarity.

### 2.3. Adjusting the latency

Even though the whole \([t - 5s, t]\) buffer is used to extract embeddings and assign local speakers to an existing (or new) cluster, only the (active) speaker activity probabilities \( s_{fk} \) at its rightmost part \([t - \lambda, t]\) are output: \( \lambda \) effectively controls the latency of the whole system.

The lowest possible value for \( \lambda \) corresponds to the period between two consecutive updates of the rolling buffer (500ms in our case). In this configuration, the rightmost parts of two consecutive buffer states \([t - 5s, t]\) and \([t + \lambda - 5s, t + \lambda]\) do not overlap: \([t - \lambda, t]\) and \([t, t + \lambda]\). Therefore, they are simply concatenated and frame-level speaker activity probabilities are passed through a final thresholding step. Local speaker \( k \) is marked as active at frame \( f \) if \( s_{fk} > \tau_{\text{active}} \).

The careful reader might have noticed that, at the very beginning of the audio stream, the initial buffer must be filled entirely before a first output can be provided—effectively leading to a much larger latency of 5s, an order of magnitude larger than the promised \( \lambda = 500\text{ms} \). However, once this initial 5s warm-up period has passed, the latency is indeed \( \lambda = 500\text{ms} \). If having a low latency from the very beginning of the stream is critical, one can simply left-pad the \([0, \lambda]\) initial incomplete buffer with zeros.

![Fig. 4](image-url) Depending on the allowed latency \( \lambda \) (between 500ms and 5s), multiple positions of the rolling buffer can be aggregated to obtain a (hopefully better) prediction. Speakers are colored according to the cluster they were found to belong to.

Figure 4 shows that, for cases where longer latency \( \lambda \) is permitted, several positions of the rolling buffer can be combined in an ensemble-like manner to obtain a more robust output. In practice, for a given frame \( f \), the final speaker activity probabilities are computed as the average of the speaker activity probabilities obtained from each buffer position.

### 3. EXPERIMENTS

#### 3.1. Datasets

We ran experiments on three different datasets covering a wide range of domains and number of speakers.

**DIHARD III** [22, 23] does not provide a training set. Therefore, we split its development set into two parts: 192 files used as training set, and the remaining 62 files used as a smaller development set. The latter is simply referred to as development set in the rest of the paper. When defining this split (shared at [huggingface.co/pyannote/segmentation]), we made sure that the 11 domains were equally distributed between both subsets. The test set is kept unchanged. We
Table 1. Experimental results on test sets. FA, Miss. and Conf. stand for false alarm, missed detection and speaker confusion rates respectively. (• = hyper-parameters optimized with latency λ = 1s for fair comparison)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>DER</td>
<td>DER</td>
<td>DER</td>
</tr>
<tr>
<td></td>
<td>FA</td>
<td>Miss.</td>
<td>Conf.</td>
<td>FA</td>
</tr>
<tr>
<td>VxB [16]</td>
<td>∞</td>
<td>3.6</td>
<td>12.5</td>
<td>22.3</td>
</tr>
<tr>
<td>w/ overlap-aware segm.</td>
<td>∞</td>
<td>4.7</td>
<td>9.7</td>
<td>4.9</td>
</tr>
<tr>
<td>w/o overlap-aware emb.</td>
<td>5s</td>
<td>5.3</td>
<td>10.0</td>
<td>9.7</td>
</tr>
<tr>
<td>w/ oracle segmentation</td>
<td>5s</td>
<td>4.6</td>
<td>11.3</td>
<td>9.3</td>
</tr>
<tr>
<td></td>
<td>5s</td>
<td>2.1</td>
<td>1.4</td>
<td>6.9</td>
</tr>
<tr>
<td>FlexSTB [17]</td>
<td>1s</td>
<td>6.2</td>
<td>9.7</td>
<td>11.8</td>
</tr>
<tr>
<td>Ours</td>
<td>1s</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

VoxConverse does not provide a proper training set either [20]. Therefore, we also split its development set into two parts: the first 144 files (abjxc to qquur, in alphabetical order) constitute the training set, leaving the remaining 72 files (qppll to zyffh) for the actual development set. Furthermore, multiple versions of VoxConverse test set have been circulating: we rely on version 0.0.2 available at github.com/joonson/voxconverse.

AMI provides an official \{training, development, test\} partition of the Mix-Headset audio files [19] that is further described as the Full partitioning in [16].

3.2. Implementation details

We use the pretrained segmentation model available at hf.co/pyannote/segmentation, which was trained on the composite training set made of the union of AMI, DIHARD III, and VoxConverse respective training sets. It ingests 5 second audio chunks and outputs one prediction every 16ms with $K_{\text{max}} = 4$ speakers. More details about the training process can be found in [10]. The speaker embedding model is based on the canonical x-vector TDNN-based architecture [4], but with filter banks replaced by trainable SincNet features [24]. It was trained with additive angular margin loss [25] using chunks of variable duration (from 2 to 5 seconds) drawn from VoxCeleb [26, 27], augmented with reverberation based on impulse responses from EchoThief [28], and additive background noise from MUSAN [29]. It reaches an equal error rate of 2.8% on VoxCeleb 1 test set using cosine distance only. We share the pretrained model and more details about the training process at hf.co/pyannote/segmentation. Weights used in the statistics pooling layer of the overlap-aware speaker embeddings were obtained with $\gamma = 3$ and $\beta = 10$. Those values were not optimized with the rest of the hyper-parameters. Instead, we handpicked them based on examples like the one in Figure 5.

3.3. Experimental protocol

While the same pretrained segmentation and embedding models were used for all three datasets, we rely on their respective development sets to optimize hyper-parameters ($r_{\text{active}}, r_{\text{update}}$ and $\delta_{\text{new}}$) specifically for each dataset. More precisely, we use the pyannote.pipeline optimization toolkit that relies on a tree-structured Parzen estimator algorithm [30] to minimize the overall diarization error rate (DER) – computed with pyannote.metrics [31] without any forgiveness collar and including overlapped speech regions. To ensure a fair comparison between different approaches, the optimization process is applied for all of them independently. In other words, it means that every row of Table 1 results from one dedicated optimization process. This includes the offline topline, the proposed online approach and its ablative variants, but excludes both FlexSTB (as we unfortunately did not have access to its implementation) and experiments on DIHARD II (where we use respective hyper-parameters tuned for DIHARD III).

4. RESULTS AND DISCUSSION

Table 1 summarizes the whole set of experiments.

Offline vs. online. We start by reporting the performance of a strong offline topline that consists of VxB [16] followed by the overlap-aware resegmentation step introduced in [10]. Because this latter resegmentation step relies on the exact same pretrained segmentation model as our proposed approach, most of the reported decrease in performance with $\lambda = 5$s is caused by speaker confusion errors (relative $+100\%$ for DIHARD III, $+160\%$ for AMI, $+130\%$ for VoxConverse). Incremental clustering still has a long way to go to be on par with offline multi-pass clustering.

Overlap-aware speaker embedding. The first ablative experiment shows that the overlap-aware weighing scheme introduced in Equation 2 brings a relative performance improvement of 10% on AMI, 18% on VoxConverse and 1% on DIHARD III. Given that they respectively contain 17%, 3%, and 11% of overlapped speech, there is still room for improvement on this particular aspect. In particular, while we handcrafted this weighing scheme, it should be possible to...
train the segmentation and speaker embedding models jointly for the latter to fully take advantage of the former’s capability at detecting and separating simultaneous speakers.

**Overlap-aware speaker segmentation.** In a second ablative experiment, we replace the segmentation model by an oracle that provides perfect binary (i.e. $s_{fk} \in \{0,1\}$) overlap-aware segmentation. As expected, missed detection is where most of the difference occurs (caused by overlapped speech), while speaker confusion only marginally improves. The community has yet to solve the problem of overlapped speech detection.

**Adjustable latency.** Figure 5 shows how the performance of our online approach evolves as we decrease the allowed latency from $\lambda = 5s$ to $\lambda = 500ms$. Speaker confusion error rate consistently increases as the latency decreases – while false alarm and missed detection remain constant. This can be explained by the ensemble-like aggregation process described in Section 2.3 that combines more views of the same problem as the allowed latency increases. Note that we kept the hyper-parameters ($\tau_{active}$, $\rho_{update}$, $\delta_{new}$) optimized for latency $\lambda = 5s$ and still get reasonable performance for lower latencies. However, it is also possible to re-optimize the hyper-parameters for a specific latency. This is what we did for the $\lambda = 1s$ setting marked with ● in Table 1 for comparison with FlexSTB [17]. Not only do we get better overall performance, but our approach also has the advantage of a lower memory footprint, as it never ingests nor runs inference on more than 5s of audio at a time (compared to 100s of FlexSTB) and keeps a single vector per speaker in memory (compared to 100s of acoustic features and per-speaker scores in FlexSTB). Furthermore, our approach with $\lambda = 3s$ reaches the same performance as the official offline baseline [18] of the DIHARD III challenge (25.5% vs 25.4%).

**Continual learning.** Figure 6 compares the performance over time of our online system against the offline topline [10]. While the performance of the latter remains somewhat constant, the former gets better as conversations unfold, almost bridging the gap after 5 minutes of conversation. As new information becomes available, our system learns better speaker centroids, hence decreasing speaker confusion error. While very long conversations can become rather expensive (if not impossible) to process with most offline models, our system can handle daylong audio streams at a practically constant memory cost, while getting better and better.

**Reproducible research.** We share an open-source implementation of this work, as well as expected outputs in RTTM format at this address to facilitate future comparisons: github.com/juanmc2005/StreamingSpeakerDiarization.

**Real time.** Computation time for one step of the rolling buffer is 165ms on a CPU Intel Cascade Lake 6248 (20 cores at 2.5Ghz) or 50ms on a GPU Nvidia Tesla V100 SXM2. This is suitable for real time applications, as the rolling buffer can be processed before its next update (every 500ms).

5. CONCLUSION

We have proposed an overlap-aware online speaker diarization system combining the end-to-end local segmentation of a 5 second long rolling buffer with incremental clustering. Apart from handling overlapping speech at every stage, our system benefits from an adjustable latency between 500ms and 5s. We show that our system outperforms FlexSTB [17] with a lower memory consumption, and that it is capable of bridging the gap to offline performance as conversations unfold. This last advantage may make it preferable to an offline system when recordings are long and resources low.
6. REFERENCES


