INTERSPEECH 2023
20-24 August 2023, Dublin, Ireland

pyannote.audio 2.1 speaker diarization pipeline: principle, benchmark, and recipe

Hervé Bredin

IRIT, Université de Toulouse, CNRS, Toulouse, France
herve.bredin@irit.fr

Abstract

pyannote.audio is an open-source toolkit written in Python for speaker diarization. Version *2.1* introduces a major overhaul of *pyannote.audio* default speaker diarization pipeline, made of three main stages: speaker segmentation applied to a short sliding window, neural speaker embedding of each (local) speakers, and (global) agglomerative clustering. One of the main objectives of the toolkit is to democratize speaker diarization. Therefore, on top of a pretrained speaker diarization pipeline that gives good results out of the box, we also provide a recipe that practitioners can follow to improve its performance on their own (manually annotated) dataset. It has been used for various challenges and reached 1st place at Ego4D 2022, 1st place at Albayzin 2022, and 6th place at VoxSRC 2022.
Index Terms: speaker diarization, open source, toolkit

1. Introduction

pyannote.audio is an open-source toolkit written in Python for speaker diarization. Version *2.1* introduces a major overhaul of *pyannote.audio* default speaker diarization pipeline (with respect to branch *1.x*) and is very similar in spirit to the line of work developed by *Kinoshita* at NTT [1, 2] that *"integrates clustering-based and end-to-end neural network-based diarization approaches into one framework"*. Hence, the proposed approach is composed of three main stages: speaker segmentation applied to a (local) sliding window, neural speaker embedding of each (local) speakers, and (global) agglomerative clustering. Section 2 goes into details about the proposed approach but we highlight here the main differences with [1, 2].

First, local neural speaker diarization is applied to much shorter overlapping windows (5s with a 500ms step) than the original one (30s with 30s step, *i.e.* no overlap), making the whole task much easier to solve:

- the upper bound on the number of speakers is smaller and the training sequences are shorter, hence reducing the computational and memory cost of training such networks;
- the use of strongly overlapping windows can be seen as test time augmentation, leading to better speaker segmentation and denser (hence easier to cluster) speaker embeddings.

Second, one of the main advantages of the joint (diarization + embedding) approach used in [1, 2] lies in embeddings that are both overlap-aware and computed from longer audio (hence more reliable). Despite relying on two separate networks applied in cascade (first segmentation, then embedding), we claim in Section 2.2 that our speaker embeddings enjoy the same properties. Training speaker embedding networks is notoriously data-hungry and it is not always possible for practitioners to gather a training dataset that both contains a large set of conversations as well as speaker labels which are consistent across conversations. Therefore, we claim that using two different networks makes the whole approach easier to adapt to a particular dataset. On top of a pretrained speaker diarization pipeline that gives good results out of the box, we also provide a set of recipes that practitioners can choose from, depending on the size of their (manually annotated) dataset.

2. Principle

Figure 1 depicts the manual speaker diarization of a 30s conversation between two speakers that we will use throughout the paper for illustration purposes.

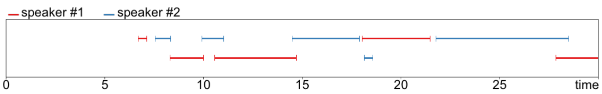

Figure 1: *Expected speaker diarization output of the sample conversation used throughout this paper.*

2.1. Local neural speaker segmentation

The first step consists in applying the end-to-end neural speaker segmentation model introduced in [3] using a sliding window of 5s with a step of 500ms. Figure 2 illustrates the output of this stage on the 30s sample whose manual annotation is depicted in Figure 1.

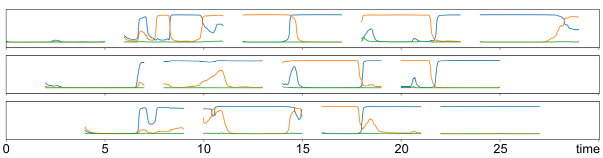

Figure 2: *Local neural speaker segmentation. For each step of the 5s window and each one of $K_{max} = 3$ speakers, the segmentation model outputs the probability of the speaker being active every 16ms. For readability, we use a sliding step of 2s and plot (otherwise overlapping) windows on three rows, but the actual practical step is 500ms.*

At this point, there is no guarantee that the same (local) speaker is consistently assigned to the same (global) speaker index. Since the speaker segmentation model has been trained in a permutation-invariant manner and is limited to at most $K_{\text{max}} = 3$ active speakers, a particular speaker might be assigned two different indices in two different windows as can be observed in Figure 2 around $t = 16$s.

A binarization step is further applied using a threshold $\theta \in [0, 1]$, which constitutes the first hyper-parameter of the

10.21437/Interspeech.2023-105

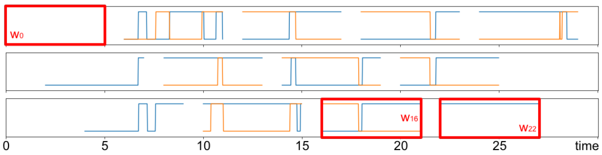

Figure 3: *Binary local speaker segmentation. For each step of the 5s window w, only $K_w \in \{0, 1, 2, 3\}$ speakers whose probability goes above θ are kept.*

proposed speaker diarization pipeline. The effect of this binarization step on the 30s audio sample is depicted in Figure 3.

2.2. Local speaker embedding

The second step consists in extracting K_w speaker embeddings for each window w: exactly one embedding per speaker who is active within the window w. Therefore, the number of speaker embeddings may vary depending on the window w. For instance, window $w_t = [t \rightarrow t + 5]$ may contain $K_w = 0$ speaker like in $w_0 = [0 \rightarrow 5]$ (because no speaker ever passes the segmentation threshold θ), $K_w = 1$ speaker like in w_{22}, or $K_w = 2$ speakers like in w_{16} (surrounded in red in Figure 3).

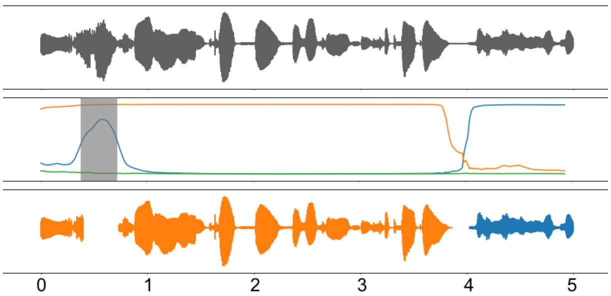

Figure 4: *Speaker embedding. Top row: 5s audio chunk. Middle row: two speakers are active according to the neural speaker segmentation model (the orange one and the blue one). Bottom row: the speaker embedding of the blue speaker is computed using only the blue audio signal, while the concatenation of orange audio signals is used to compute the speaker embedding of the orange speaker. No embedding is extracted for the green speaker as their probability never goes above θ segmentation threshold.*

As depicted by the gray overlay in Figure 4, speakers may overlap partially within the considered window. To account for this possibility, the embedding of speaker k is computed from the concatenation of audio samples during which (1.) speaker k is active and (2.) other speakers $k' \neq k$ are inactive. This is similar in spirit to overlap-aware speaker embeddings in [4].

Compared to the *standard* approach that consists in extracting exactly one speaker embedding using a short (typically 1 or 2 seconds) periodic window [5], the proposed speaker embeddings are expected to be more reliable for two main reasons:

- they are extracted from audio excerpts that only contain speech samples from one single speaker while the *standard* approach may extract speaker embeddings from a mixture of speakers (and non-speech);
- they are extracted from potentially longer audio excerpts (up to 5s in case a speaker speaks during the whole window w) while the *standard* approach is limited to 1 or 2 seconds.

The main drawback of this approach is that it depends on the upstream speaker segmentation model whose errors could lead to degraded speaker embeddings.

2.3. Global agglomerative clustering

The third step consists in clustering the resulting set of speaker embedding in order to assign each local speaker to a global cluster, as depicted by colors in Figure 5.

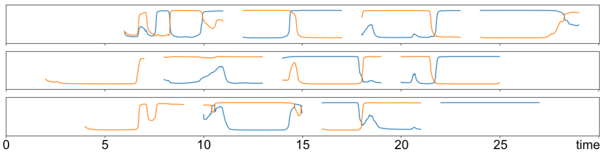

Figure 5: *Local speaker segmentation after global clustering.*

Although spectral clustering [6] and variational Bayesian hidden Markov models [5] have been the prefered clustering techniques in recent speaker diarization literature [7], the proposed pipeline relies on a *classical* agglomerative hierarchical clustering with *centroid* linkage (also known as the *UPGMC* algorithm) for two main reasons:

- the latter only introduces a second hyper-parameter (the distance threshold δ used as stopping criterion of the agglomerative clustering process) while both spectral clustering and variational Bayesian hidden Markov models rely on at least a couple of hyper-parameters;
- while variational Bayesian hidden Markov models (and, to a lesser extent spectral clustering[1]) expects that speaker embeddings are ordered chronologically with a strict periodicity (*e.g.* one embedding every second), the speaker embedding process introduced in Sections 2.1 and 2.2 cannot guarantee these properties because a variable number (zero, one, or more) of speaker embeddings may be extracted every 500ms (or whichever step is used by the 5s sliding window).

The choice of *centroid* linkage over variants (such as *average*, *single*, *complete*, or *Ward* linkage) derives from the fact that the former consistently outperforms the latter on every single validation sets later discussed in Section 3 (the runner-up being the more common *average* linkage).

2.4. Final aggregation

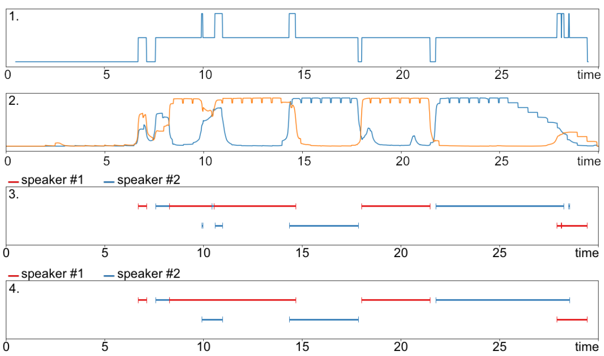

Figure 6: *Final aggregation.*

The fourth and final step aims at aggregating the clustered local speaker segmentation into an actual speaker diarization

[1]when combined with the critical step of Gaussian blur refinement of the Laplacian matrix used for auto-tuning [6]

output. Depicted in Figure 6, it can be summarized as follows (from top to bottom):

1. estimating the instantaneous (*i.e.* for each frame f) number of speakers K_f, by summing the binary local speaker segmentation obtained in Section 2.1 and Figure 3 and averaging over overlapping windows (rounding to closest integer);
2. estimating the instantaneous score of each cluster by summing the clustered local speaker segmentation obtained in Section 2.3 and Figure 5 over overlapping windows;
3. selecting the K_f (given by step 1) clusters with highest instantaneous score (step 2) and converting from discrete frame indices to the temporal domain;
4. filling within-speaker gaps shorter than a (usually short) predefined duration Δ.

The last step is optional as the value of Δ usually depends more on the instructions given to the pool of human annotators than to the data itself. For instance, DIHARD evaluation plan says that *"small pauses [shorter than] 200 ms by a speaker are not considered to be segmentation breaks and should be bridged into a single continuous segment"* [8]; VoxConverse guidelines say that *"[s]peech segments are split when pauses are greater than [250 ms]"* [9]; the Albayzin 2022 evaluation plan goes even further by requesting that *"[c]onsecutive segments of the same speaker with a silent [sic] of less that [sic] 2 seconds [...] are considered as a single segment"* [10].

3. Reproducible benchmark

Despite the availability of several benchmarking initiatives (such as *DIHARD*, *VoxSRC*, or *Albayzin* challenges, whose organizers are heartily thanked), it remains very difficult to gauge the many speaker diarization approaches proposed by the research community, for various reasons:

- Despite a growing number of freely available datasets such as *AISHELL-4* [11], *Albayzin/RTVE* [10], *AliMeeting* [12], *AMI* [13], *VoxConverse* [9], *Ego4D* [14], or *This American Life* [15], some papers only report results on a limited set of datasets either behind paywalls (such as *CALLHOME* [16], *DIHARD* [17], or *REPERE* [18]), on purely synthetic datasets, or even private in-house datasets – effectively preventing others (and newcomers in particular) from comparing their approach to the so-called *state-of-the-art*.
- Two papers reporting results on the *same dataset* often use *different experimental protocols* without even noticing. For instance, they might use a slightly different test set[2], different versions of the gold standard [5], different configuration of the reported diarization error rate (*e.g.* with or without forgiveness collars), or different assumption about the (unrealistic) availability of an oracle voice activity detector.

We apologize for the tone of the above rant. The objective was to convince the reader that they should at the very least share the actual output of their proposed approaches (ideally in *de facto* standard RTTM format) to solve (part of) those problems. We go one step further in Figure 7 by providing open source code to produce those RTTMs and therefore allow evaluating the approach on any other (possibly private) dataset.

The leftmost part of Table 1 summarizes the performance of these few lines of code. There, the processing is fully automatic (no oracle voice activity detection, no oracle number of speakers, no fine-tuning of the internal models nor tuning of the pipeline hyper-parameters to each specific dataset) with the least forgiving diarization error rate (DER) setup (no forgiveness collar, evaluation of overlapped speech). Unless stated otherwise in the first column, we report results on the official test sets of 11 benchmarks. We claim state-of-the-art performance on *AISHELL-4* [11], *AMI headset mix* and *array1/ channel1* [13], *REPERE phase2* [18], *Albayzin RTVE 2022* [10], *Ego4D* [14], and *This American Life* [15]. Precomputed RTTMs are available for download at `hf.co/pyannote/speaker-diarization/tree/2.1.1`.

```
# install pyannote.audio
pip install pyannote-audio==2.1.1
```

```
# load pretrained pipeline
from pyannote.audio import Pipeline
pipeline = Pipeline.from_pretrained(
    "pyannote/speaker-diarization")

# apply pipeline and dump RTTM
diarization = pipeline("audio.wav")
with open("audio.rttm", "w") as f:
    diarization.write_rttm(f)
```

Figure 7: *From zero to RTTMs with pyannote.audio*

Using one Nvidia Tesla V100 SXM2 GPU (for neural inference described in Sections 2.1 and 2.2) and one Intel Cascade Lake 6248 CPU (for the clustering and aggregation described in Sections 2.3 and 2.4), the proposed pipeline is 40 times faster than real time, with most of the time spent in the speaker embedding extraction step. In particular, all experiments reported in Table 1 relies on the implementation of ECAPA-TDNN [22] available in SpeechBrain [23] because it was found to outperform three open-source alternatives. For instance, on *VoxConverse v0.3*, the fine-tuned pipeline reaches DER $= 14.9\%$ with the xvector implementation available in *pyannote.audio* [20], 12.0% with *NeMo*'s *TitaNet* [24], 10.8% with *RawNet3* [25], and 10.7% with *SpeechBrain*'s ECAPA-TDNN.

4. Recipe

While the leftmost part of Table 1 reports performance of the pretrained speaker diarization pipeline (with default hyper-parameters and default internal models), this section provides a recipe to adapt it to a particular target domain and (hopefully) get better performance. Depending on the number and duration of labeled conversations, the practitioner may either focus on optimizing hyper-parameters (θ, δ and Δ, introduced in Sections 2.1, 2.3, and 2.4 respectively), additionally fine-tune the internal speaker segmentation model, or both. Fine-tuning speaker embedding might also be an option in case even more data is available for a particular domain but this is out of the scope of both this paper and *pyannote.audio* (since we rely on external libraries for this model).

4.1. Optimizing pipeline hyper-parameters

In case a small development set of labeled conversations is available, optimizing pipeline hyper-parameters (with the few lines of code in Figure 8) may lead to significant performance improvement.

Datasets listed in Table 1 are split into two groups: *in domain* datasets whose training subsets have been used to train the underlying segmentation model (available at

[2] `github.com/BUTSpeechFIT/CALLHOME_sublists/issues/1`

Table 1: *Performance of the (default, optimized, and fine-tuned) pipelines on 11 different benchmarks. The grey background marks the best results for each dataset as well as those less than 5% worse relatively. DER stands for diarization error rate, which is the sum of two terms: CONF for speaker confusion rate, and FA+MISS for false alarm and missed detection rates. We also report the scale of development (for optimizing hyper-parameters, in number of files) and training sets (for fine-tuning the segmentation model, in number of hours) as well as the best baseline we could find in the literature as of February 2023. No comparable baseline was found for VoxConverse (either because of slightly different test sets or different metric configuration) and Albayzin/RTVE2022 (because the test set labels have only been released very recently).*

		Default pipeline			**Dev.**	**+ optimized hyper-parameters**			**Train**	**+ finetuned segmentation model**			**Baseline**	
		DER%	FA+MISS%	CONF%		DER%	FA+MISS%	CONF%		DER%	FA+MISS%	CONF%	DER%	Source
AISHELL-4	[11]	14.1	8.4	5.7	20 files	14.0	7.9	6.1	96h	14.5	7.9	6.6	16.1	[19]
AMI *headset mix*	[13]	18.9	14.0	4.9	18 files	18.9	14.0	4.9	80h	18.5	14.0	4.4	19.0	[5]
DIHARD 3 *full*	[17]	26.9	18.9	8.0	62 files	22.2	15.1	7.1	25h	21.9	14.4	7.5	16.8	[17]
REPERE *phase 2*	[18]	8.2	4.7	3.5	27 files	8.3	4.9	3.4	33h	8.3	4.8	3.4	12.6	[20]
VoxConverse *v0.3*	[9]	11.2	7.3	3.9	72 files	10.8	7.2	3.7	15h	10.7	7.4	3.3	NA	see caption
Average (in domain)		15.9	10.7	5.2		14.8	9.8	5.0		14.8	9.7	5.0		
Albayzin/RTVE *2022*	[10]	25.6	12.4	13.2	40 files	23.1	10.9	12.2	126h	21.8	10.0	11.8	NA	see caption
AliMeeting *channel 1*	[12]	27.4	18.8	8.6	8 files	29.0	19.0	10.0	110h	23.8	15.6	8.2	23.5	[19]
AMI *array 1, channel 1*	[13]	27.1	21.9	5.2	18 files	25.9	20.7	5.2	80h	22.2	16.0	6.2	23.7	[19]
CALLHOME *part 2*	[16]	32.4	20.0	12.4	40 files	32.4	20.0	12.4	7h	29.3	17.6	11.7	21.8	[5]
Ego4d *v1, validation*	[14]	64.0	48.3	15.7	50 files	60.3	44.5	15.8	32h	51.8	33.7	18.0	67.2	[14]
This American Life	[15]	20.8	13.9	6.9	34 files	18.4	10.4	8.1	580h	15.2	2.6	12.6	25.0	[21, 20]
Average (out of domain)		32.9	22.6	10.3		31.5	20.9	10.7		27.4	15.9	11.5		
Average (overall)		25.2	17.2	8.0		23.9	15.9	8.0		21.6	13.1	8.5		

```python
# load dataset with pyannote.database
from pyannote.database import get_protocol
dataset = get_protocol(...)
dev_files = list(dataset.development())

# optimize hyper-parameters with pyannote.pipeline
from pyannote.pipeline import Optimizer
optimizer = Optimizer(pipeline)
optimizer.tune(dev_files)

# apply optimized pipeline
diarization = optimizer.best_pipeline("audio.wav")
```

Figure 8: *Optimizing hyper-parameters with pyannote.pipeline*

```python
# load pretrained model
from pyannote.audio import Model
model = Model.from_pretrained(
    "pyannote/segmentation")

# prepare model for fine-tuning
from pyannote.audio.tasks import Segmentation
model.task = Segmentation(dataset)

# fine-tune model with pytorch-lightning
from pytorch_lightning import Trainer
trainer = Trainer()
trainer.fit(model)
```

Figure 9: *Fine-tuning segmentation model with lightning.*

`hf.co/pyannote/segmentation`) and the remaining datasets that have not (hence considered *out of domain*). *In domain* datasets benefit the most from hyper-parameter optimization (with a relative 7% DER decrease) while it might even degrade performance when only a limited set of development files is available (see *AliMeeting* results for instance). A closer look at the actual values of the hyper-parameters before and after optimization shows that θ (used for binarizing speaker segmentation) is the most important hyper-parameter to tune, followed by Δ (for filling short intra-speaker gaps) and then only δ (that serves as stopping criterion for the clustering stage).

4.2. Fine-tuning segmentation model

When a larger training set of labeled conversations is available, fine-tuning the internal speaker segmentation model (with the few lines of code in Figure 9) leads to significant performance boost. The best performance is almost always obtained with this configuration (as highlighted by the gray background in the rightmost part of Table 1). As expected, *out of domain* datasets benefit the most from this fine-tuning step (witnessing a 17% relative DER decrease *vs.* only 7% for *in domain*).

Furthermore, a nice side effect of this fine-tuning step is that it completely removes the need for the final post-processing step (numbered #4 in Section 2.4). Hence, the optimal value for Δ systematically converges towards zero second when the segmentation model has first been fine-tuned to the target domain (which is equivalent to not filling any intra-speaker gaps). This is to be compared with the following optimal values for Δ when the pipeline relies on the pretrained speaker segmentation model: 10ms for *AISHELL-4*, 400ms for *REPERE* and *VoxConverse*, 500ms for *AliMeeting*, 1.5s for *Albayzin*, or even 2s for *The American Life*. In other words, fine-tuning the segmentation not only improves the performance but also reduces the dimensionality of the hyper-parameter search space, from 3 (δ, θ, and Δ) to only 2 hyper-parameters (δ and θ).

5. Conclusion

We introduced version 2.1 of *pyannote.audio* open source speaker diarization pipeline, evaluated its performance on a large collection of benchmarking datasets, and described a recipe that practitioners can follow to make the most of their own labeled data and adapt the pretrained pipeline to their particular use case. This recipe has been used to reach 6th place at VoxSRC 2022 challenge, 1st place at Ego4D 2022 challenge, and 1st place at Albayzin 2022 challenge. The source code, pretrained models and expected outputs are openly shared with the community at `github.com/pyannote/pyannote-audio`.

This work was granted access to the HPC resources of IDRIS under the allocations AD011012177R1 and AD011012177R2 made by GENCI.

6. References

[1] K. Kinoshita, M. Delcroix, and N. Tawara, "Integrating End-to-End Neural and Clustering-Based Diarization: Getting the Best of Both Worlds," in *Proc. ICASSP 2021*, 2021.

[2] ——, "Advances in Integration of End-to-End Neural and Clustering-Based Diarization for Real Conversational Speech," in *Proc. Interspeech 2021*, 2021.

[3] H. Bredin and A. Laurent, "End-to-end speaker segmentation for overlap-aware resegmentation," in *Proc. Interspeech 2021*, 2021.

[4] J. M. Coria, H. Bredin, S. Ghannay, and S. Rosset, "Overlap-Aware Low-Latency Online Speaker Diarization Based on End-to-End Local Segmentation," in *Proc. ASRU 2021*, 2021.

[5] F. Landini, J. Profant, M. Diez, and L. Burget, "Bayesian HMM Clustering of x-vector Sequences (VBx) in Speaker Diarization: Theory, Implementation and Analysis on Standard Tasks," *Computer Speech & Language*, 2022.

[6] T. J. Park, K. J. Han, M. Kumar, and S. Narayanan, "Auto-Tuning Spectral Clustering for Speaker Diarization Using Normalized Maximum Eigengap," *Signal Processing Letters*, 2020.

[7] T. J. Park, N. Kanda, D. Dimitriadis, K. J. Han, S. Watanabe, and S. Narayanan, "A Review of Speaker Diarization: Recent Advances with Deep Learning," *Computer Speech & Language*, 2022.

[8] N. Ryant, K. Church, C. Cieri, J. Du, S. Ganapathy, and M. Liberman, "Third DIHARD Challenge Evaluation Plan," *arXiv preprint arXiv:2006.05815*, 2020.

[9] J. S. Chung, J. Huh, A. Nagrani, T. Afouras, and A. Zisserman, "Spot the Conversation: Speaker Diarisation in the Wild," in *Proc. Interspeech 2020*, 2020.

[10] A. Ortega, A. Miguel, E. Lleida, V. Bazàn, C. Pérez, and A. de Prada, "Albayzin Evaluation IberSPEECH-RTVE 2022 Speaker Diarization and Identity Assignment," http://catedrartve.unizar.es/reto2022/SDIAC2022_Evalplan.pdf.

[11] Y. Fu, L. Cheng, S. Lv, Y. Jv, Y. Kong, Z. Chen, Y. Hu, L. Xie, J. Wu, H. Bu, X. Xu, J. Du, and J. Chen, "AISHELL-4: An Open Source Dataset for Speech Enhancement, Separation, Recognition and Speaker Diarization in Conference Scenario," in *Proc. Interspeech 2021*, 2021.

[12] F. Yu, S. Zhang, Y. Fu, L. Xie, S. Zheng, Z. Du, W. Huang, P. Guo, Z. Yan, B. Ma, X. Xu, and H. Bu, "M2MeT: The ICASSP 2022 Multi-Channel Multi-Party Meeting Transcription Challenge," in *Proc. ICASSP 2022*, 2022.

[13] J. Carletta, S. Ashby, S. Bourban, M. Flynn, M. Guillemot, T. Hain, J. Kadlec, V. Karaiskos, W. Kraaij, and M. Kronenthal, "The AMI Meetings Corpus," in *Proc. Symposium on Annotating and Measuring Meeting Behavior*, 2005.

[14] K. Grauman, A. Westbury, E. Byrne, Z. Chavis, A. Furnari, R. Girdhar, J. Hamburger, H. Jiang, M. Liu, X. Liu, M. Martin, T. Nagarajan, I. Radosavovic, S. K. Ramakrishnan, F. Ryan, J. Sharma, M. Wray, M. Xu, E. Z. Xu, C. Zhao, S. Bansal, D. Batra, V. Cartillier, S. Crane, T. Do, M. Doulaty, A. Erapalli, C. Feichtenhofer, A. Fragomeni, Q. Fu, C. Fuegen, A. Gebreselasie, C. Gonzalez, J. Hillis, X. Huang, Y. Huang, W. Jia, W. Khoo, J. Kolar, S. Kottur, A. Kumar, F. Landini, C. Li, Y. Li, Z. Li, K. Mangalam, R. Modhugu, J. Munro, T. Murrell, T. Nishiyasu, W. Price, P. R. Puentes, M. Ramazanova, L. Sari, K. Somasundaram, A. Southerland, Y. Sugano, R. Tao, M. Vo, Y. Wang, X. Wu, T. Yagi, Y. Zhu, P. Arbelaez, D. Crandall, D. Damen, G. M. Farinella, B. Ghanem, V. K. Ithapu, C. V. Jawahar, H. Joo, K. Kitani, H. Li, R. Newcombe, A. Oliva, H. S. Park, J. M. Rehg, Y. Sato, J. Shi, M. Z. Shou, A. Torralba, L. Torresani, M. Yan, and J. Malik, "Ego4D: Around the World in 3,000 Hours of Egocentric Video," in *Proc. CVPR 2022*, 2022.

[15] H. H. Mao, S. Li, J. J. McAuley, and G. W. Cottrell, "Speech Recognition and Multi-Speaker Diarization of Long Conversations," in *Proc. Interspeech 2020*, 2020.

[16] F. Castaldo, D. Colibro, E. Dalmasso, P. Laface, and C. Vair, "Stream-Based Speaker Segmentation using Speaker Factors and Eigenvoices," in *Proc. ICASSP 2008*, 2008.

[17] N. Ryant, P. Singh, V. Krishnamohan, R. Varma, K. Church, C. Cieri, J. Du, S. Ganapathy, and M. Liberman, "The Third DIHARD Diarization Challenge," in *Proc. Interspeech 2021*, 2021.

[18] J. Kahn, O. Galibert, L. Quintard, M. Carré, A. Giraudel, and P. Joly, "A Presentation of the REPERE Challenge," in *Proc. CBMI 2012*, 2012.

[19] D. Raj, D. Povey, and S. Khudanpur, "Gpu-accelerated guided source separation for meeting transcription," in *Proc. Interspeech 2023*, 2023.

[20] H. Bredin, R. Yin, J. M. Coria, G. Gelly, P. Korshunov, M. Lavechin, D. Fustes, H. Titeux, W. Bouaziz, and M.-P. Gill, "pyannote.audio: Neural Building Blocks for Speaker Diarization," in *Proc. ICASSP 2020*, 2020.

[21] M. I. Tanveer, D. Casabuena, J. Karlgren, and R. Jones, "Unsupervised Speaker Diarization that is Agnostic to Language, Overlap-Aware, and Tuning Free," in *Proc. Interspeech 2022*, 2022.

[22] B. Desplanques, J. Thienpondt, and K. Demuynck, "ECAPA-TDNN: Emphasized Channel Attention, Propagation and Aggregation in TDNN Based Speaker Verification," in *Proc. Interspeech 2020*, 2020.

[23] M. Ravanelli, T. Parcollet, P. Plantinga, A. Rouhe, S. Cornell, L. Lugosch, C. Subakan, N. Dawalatabad, A. Heba, J. Zhong, J.-C. Chou, S.-L. Yeh, S.-W. Fu, C.-F. Liao, E. Rastorgueva, F. Grondin, W. Aris, H. Na, Y. Gao, R. D. Mori, and Y. Bengio, "SpeechBrain: A General-Purpose Speech Toolkit," 2021, arXiv:2106.04624.

[24] N. R. Koluguri, T. Park, and B. Ginsburg, "TitaNet: Neural Model for Speaker Representation with 1D Depth-Wise Separable Convolutions and Global Context," in *Proc. ICASSP 2022*, 2022.

[25] J.-w. Jung, Y. J. Kim, H.-S. Heo, B.-J. Lee, Y. Kwon, and J. S. Chung, "Pushing the Limits of Raw Waveform Speaker Recognition," *Proc. Interspeech 2022*, 2022.