Future Generation Computer Systems 185 (2026) 108648

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Speaker-attributed meeting transcription refinement with constrained open-weight language models☆

Costin-Alexandru Deonise [a], Taisia-Maria Coconu [a], Muhammad Khurram Zahur Bajwa [b], Catalin Negru [a], Bodgan-Costel Mocanu [a,*], Aniello Castiglione [b], Florin Pop [a,c,d]

[a] National University of Science and Technology POLITEHNICA Bucharest, Splaiul Independentei 313, Bucharest, Romania
[b] University of Salerno, Fisciano, Salerno, Italy
[c] National Institute for Research and Development in Informatics – ICI Bucharest, Romania
[d] Academy of Romanian Scientists, Bucharest, Romania

ARTICLE INFO

Keywords:
Speaker diarization
VoIce Activity Detection
Speech-to-Text
Large Language Models

ABSTRACT

Speaker diarization and Automatic Speech Recognition (ASR) are traditionally evaluated as independent components. A particular use case, is practical meeting transcription that requires the seamless integration of accurate speaker segments and reliable speaker-attributed text. This paper presents a modular offline pipeline for speaker-attributed meeting transcription that integrates Pyannote for diarization, Whisper for ASR, and a constrained open-weight Large Language Model (LLM) for transcript refinement. The evaluation of the proposed solution is conducted using the AMI Meeting Corpus. We use a rigorous methodology that converts manual annotations into time-aligned speaker activity segments and reference transcripts. This enables a multi-dimensional performance analysis using standard metrics, including Diarization Error Rate (DER), Jaccard Error Rate (JER), Word Error Rate (WER), and concatenated minimum-permutation Word Error Rate (cpWER). The post-processing stage is formulated as a constrained correction task, where small, instruction-tuned models are deployed locally to ensure data privacy and eliminate reliance on proprietary APIs. These models are restricted to conservative corrections, specifically targeting speaker-attribution inconsistencies and repetitive ASR artifacts. Our experimental results demonstrate that while unconstrained LLM post-processing effectively reduces repetitions, it often compromises lexical fidelity. Conversely, our proposed conservative validation and acceptance filtering mechanism maintains WER and cpWER parity with the baseline while significantly mitigating transcript artifacts. Our findings suggest that local, open-weight LLMs are more effective as selective verification modules than as autonomous rewriters in high-fidelity transcription systems.

1. Introduction

Traditionally, speaker diarization, is defined as the task of determining "who spoke when" as mentioned in [1,2]. It represents only a partial solution for automated meeting transcription. High-utility systems must bridge the gap between segmentation and recognition to resolve the "who said what" challenge through speaker-attributed text. In realistic multi-speaker environments, this joint problem is not trivial. The complexity comes from the cascading interactions between diarization imprecision [3], fragmented utterances, and Automated Speech Recognition (ASR) instability, in particular, the emergence of repetitive artifacts and attribution errors that compromise transcript readability. There are already some state-of-the-art tasks, such as ASR [4], for extracting text, and Speaker Segmentation for assigning timestamps to each speaker. The main limitation of this approach is that it must be combined with other methods to extract meaningful information from meetings or calls.

There are only two types of Speaker Diarization: offline and live. The offline type is useful when there are hours of recording and the main objective is to get the transcript from those conversations. This implies audio features extraction such as Mel-Frequency Cepstral Coefficients (MFCCs), log-mel spectrograms, or other spectral features and it utilize libraries such as `librosa` [5,6] or built-in methods. Moreover, it implies to normalize and window the features appropriately to capture relevant information while minimizing noise and irrelevant variations. In contrast, live diarization is used when there is no opportunity to record or when real-time results are needed. This

☆ This article is part of a Special issue entitled: 'MLOps' published in Future Generation Computer Systems.

* Corresponding author.
E-mail address: bogdan_costel.mocanu@upb.ro (B.-C. Mocanu).

https://doi.org/10.1016/j.future.2026.108648
Received 27 February 2026; Received in revised form 15 May 2026; Accepted 3 June 2026
Available online 9 June 2026
0167-739X/© 2026 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

diarization type relies on Voice Activity Detection (VAD) [7–9] to detect and segment the portions of the audio where human speech is present. To achieve VAD we need to employ VAD models like 'speech brain' or other pretrained VAD systems to accurately identify speech segments. and to filter out non-speech segments (silence, noise, etc.).

In general, this type of solution suffers from the reliance on circular evaluation frameworks that use ASR-generated ground truths. To address this limitation, we propose in this paper a rigorous evaluation baseline that uses the AMI Meeting Corpus (Augmented Multiparty Interaction) [10]. By parsing word-level manual transcriptions and speaker timing data, we derive two distinct reference objects: (i) speaker-attributed transcripts and (ii) time-aligned speaker activity segments. This enables a robust, human-centric benchmark for assessing integrated system performance.

We propose a novel offline modular pipeline for speaker-attributed meeting transcription. Our approach uses Pyannote-based diarization with Whisper-based ASR to produce a baseline speaker-attributed output. Moreover, we introduce a constrained post-processing module that uses open-weight Large Language Models (LLMs). Unlike unconstrained generative approaches, this module is strictly parameterized to perform conservative candidate corrections. The main advantage of this approach is the fact that it focuses on the mitigation of speaker-attribution errors and repetitive ASR artifacts while maintaining maximum lexical fidelity.

Meeting information ranges from public to sensitive, and cloud-based solutions could lead to information leakage. A viable solution is to use small, instruction-tuned open-weight models that eliminate the privacy risks associated with proprietary APIs. Other benefits of this approach are: (i) reproducibility, (ii) cost, and (iii) data sovereignty. Furthermore, this approach facilitates systematic experimentation with prompting and validation strategies within controlled computational environments, which provides a scalable and secure framework for high-fidelity meeting transcription.

Our proposed system improves the performance of speaker diarization and text extraction by improving the following metrics: Diarization Purity [11], Diarization Coverage (for the text) [12–14], Word Diarization Error Rate (WDER) and Concatenated Minimum-Permutation Word Error Rate (cpWER).

The original contributions of this paper are as follows:

- Design a robust evaluation framework: we design the evaluation protocol for speaker-attributed transcription using manually annotated data from the AMI Meeting Corpus. This approach mitigates the circular dependencies introduced by ASR-generated ground truths.
- Develop a modular transcription pipeline: we develop an offline architecture that orchestrates Pyannote-based diarization and Whisper-based ASR, unified by an LLM-driven refinement stage specifically optimized for speaker attribution.
- Design and implement constrained edit formulation: We use LLMs for post-processing as a constrained editing task. By implementing a validation stage that rejects over-modified or invalid outputs, the system prioritizes lexical preservation and limits the model to conservative, but also high-confidence corrections.
- Propose a privacy-centric local inference: We measure the efficiency for small, open-weight instruction-tuned models as viable alternatives to proprietary APIs. This approach enhances local reproducibility, minimizes operational costs, and ensures data sovereignty for sensitive meeting content.
- Conduct comprehensive multi-metric assessment: the system is evaluated using an exhaustive suite of metrics, including DER, JER, WER, cpWER, Purity, and Coverage, with explicit algorithmic handling of speaker-label permutation to ensure statistical accuracy.

Our solution focuses on offline speaker-attributed transcription for streaming environments. Real-time diarization under strict latency constraints is deferred to future research.

The remainder of this paper is organized as follows. In Section 2, we review the state of the art in diarization, ASR, and LLM-based refinement. In Section 3, we detail the proposed pipeline architecture. In Section 4, we describe the experimental setup, the dataset, and evaluation metrics, while in Section 5, we present a critical analysis of the obtained results. Finally, in Section 6, we conclude with directions for future work.

2. Related work

Compared to existing research, the findings showcase novel approaches to VAD that incorporate advanced techniques such as data augmentation, hyperparameter optimization, and the integration of multilingual speech corpora. These methodologies represent a forward-thinking perspective that addresses evolving challenges in speech processing.

Although speaker diarization applies in applications such as meeting transcription, conversational interaction analysis, and audio indexing, there is still much to improve in various areas. One important research issue is the integration of speaker diarization with VAD [15] systems to create joint frameworks that perform better than modular systems. Another research challenge is to deal with speaker overlapping [16] in the diarization process by implementing the application of speech separation, post-processing, and joint modeling of speech separation and speaker diarization.

The third important research area is the online processing of speaker diarization [17], as many speaker diarization methods require the entire recording to perform speaker diarization, whilst applications such as meeting transcription require very short latency for assigning the speaker.

The preliminary results outlined in previous research demonstrate a notable degree of novelty and relevance for high-accuracy, real-time speech voice detection at national and international levels [18]. These results contribute to advancing the field by introducing innovative methodologies and demonstrating significant improvements in VAD. These findings are particularly significant given the diverse range of models, algorithms, and pipelines. The various models used, including SpeechBrain, Picovoice [19], WebRTC [20], and In a Speech Segmenter [21], highlight the breadth of approaches. Each model represents a cutting-edge advancement in the field, with distinct strengths and limitations.

Using Whisper, a generative model, presents a significant problem because of its propensity to have hallucinations when speaker segmentation is not ideal. As a result, the words of the next speaker may be mistakenly identified with the present one [22].

Text extraction and speaker diarization are mostly hampered by the datasets and corpora on which the system was built. In [23], the authors discuss the Broadcast News (BN) domain, where speech is recorded in two locations: a quiet studio and a noisy field. The recordings are done using boom or lapel microphones. On the other hand, desktop, telephone, or single microphones are typically used to record meetings since they are more user-friendly than head-mounted or lapel microphones. Consequently, BN data often have a higher signal-to-noise ratio than meeting recordings. Furthermore, variations in recording quality might result from variations in meeting room layouts and microphone locations, such as background noise, reverberation, and varied speech volumes (depending on the speakers-microphones distance) or speakers overlapping.

Another important research challenge in speaker diarization is managing overlapping speakers. Several recent studies produced positive findings for this task. By employing a novel Two-stage OverLap-aware Diarization framework (TOLD) [24] that involves a speaker overlap-aware post-processing (SOAP) model to iteratively refine the diarization results of EEND-OLA [25], this one was able to achieve a 14.39%

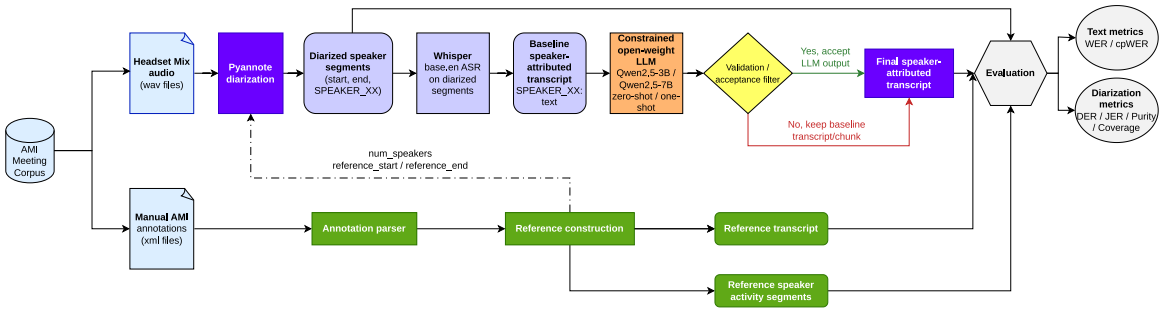


Fig. 1. Architecture of the proposed processing pipeline: (i) data ingestion, (ii) diarization-transcription output construction, (iii) application of constrained language-model post-processing and (iv) output validation using WER, cpWER, DER, JER, Purity and Coverage metrics.

relative improvement in terms of Diarization Error Rates (DER). Thus far, the most successful method has been end-to-end neural diarization using an attractor based on encoder–decoder [26], which is also effective for an unspecified number of speakers.

Models such as SpeechBrain [27] have demonstrated high recall and precision, indicating their ability to detect speech segments while minimizing false positives and negatives. However, challenges persist in noisy or overlapping speech scenarios, highlighting areas for further improvement.

In general, the existing findings represent a significant contribution to the field of speech voice detection. They offer valuable insights and methodologies that advance the state-of-the-art in real-time, high-accuracy speech processing applications.

3. Proposed pipeline for speaker diarization

Given an input meeting recording, the system first estimates speaker activity regions using Pyannote. The resulting diarization output contains time intervals and anonymous speaker labels. Each diarized segment is then passed to Whisper for automatic speech recognition. The segment-level ASR outputs are concatenated into a speaker-attributed transcript of the form `SPEAKER_XX: text`.

Finally, the constraint post-processing proposed module uses a language model. Unlike free-form transcript rewriting, the language model avoids introducing new lexical content and applies only conservative candidate corrections. Its role is limited to correcting speaker-attribution errors or reducing repetitive ASR artifacts when the surrounding dialogue context suggests that the baseline output is unreliable.

The complete processing flow is as follows:

1. Load the meeting audio file as wav files, and their corresponding manual AMI word-level annotations;
2. Build the speaker-attributed reference transcript and reference speaker activity segments from the manual annotations as XML files;
3. Use Pyannote tool to estimate speaker diarization segments;
4. Transcribe each diarized segment using Whisper;
5. Build the baseline speaker-attributed transcript from the Pyannote–Whisper output;
6. Apply constrained language-model post-processing using zero-shot and one-shot prompts;
7. Validate the language-model output by checking speaker-tag format and lexical preservation;
8. Evaluate the resulting transcripts using WER and cpWER, and evaluate speaker segmentation using DER, JER, Purity, and Coverage metrics.

We show this conceptual model for the proposed speaker diarization service in Fig. 1.

Pyannote version 3.1 [28], the latest, outputs the list of speakers along with their respective start and end times. Whisper utilizes large-v3 twice: (i) once to transcribe the entire audio and (ii) another time for each segment provided by Pyannote. Instead of relying on proprietary closed-source models, this study utilizes the Qwen series of open-weight, instruction-tuned Large Language Models (LLMs) for the constrained post-processing of transcripts. The selection of the Qwen architecture is motivated by its superior instruction-following benchmarks and its capacity for local deployment, which ensures experimental reproducibility, optimizes inference costs, and maintains data sovereignty by eliminating the need for external API calls for sensitive meeting data.

We specifically evaluate Qwen2.5-3B-Instruct and Qwen2.5-7B-Instruct [29] to investigate two distinct computational configurations:

- the 3B variant serves as a lightweight baseline suitable for resource-constrained edge environments;
- the 7B variant offers an enhanced capacity for adhering to complex formatting and conservative correction constraints.

By evaluating models within the same architectural family, we facilitate a controlled analysis of how model scaling influences the precision of speaker-attributed refinement. Furthermore, we assess performance across both zero-shot and one-shot prompting paradigms to determine the sensitivity of these local models to in-context demonstrations within the transcription workflow.

3.1. Pyannote for speaker diarization

Pyannote is a Python library specifically designed for audio analysis tasks, including Speaker Diarization. It provides tools and pre-trained models that enable accurate identification and segmentation of speakers within audio data. Pyannote operates effectively on mono audio samples at 16 kHz and generates segments with annotations. To integrate Pyannote into our Speaker Diarization pipeline, the following setup and configuration steps were undertaken:

- **Installation**: Pyannote was installed using standard Python package management tools (pip install Pyannote). Dependencies such as PyTorch were also installed to leverage GPU acceleration for improved processing speed.
- **Model Initialization**: the pre-trained diarization pipeline (Pyannote/speaker-diarization-3.1) was initialized with an authentication token (`use_auth_token`) for secure access to the model repository.

Our implementation that integrates Pyannote in the processing pipeline is based on [30]. The authors demonstrate how to integrate Pyannote and how to configure it to run on GPU-based systems. The library allows control over the number of speakers from the outset, including options to set *min_speakers* and *max_speakers*. One important aspect is the fact that Pyannote allows loading WAV files directly into memory for faster processing.

The output format is an RTTM file named "`sample.rttm`", which is suitable for further analysis and evaluation tasks such as calculating DER. This provides a comprehensive listing of speaker enumerations as well.

Table 1
Comparison of Whisper models.

Size	Parameters	English-only	Multilingual
`tiny`	39M	x	x
`base`	74M	x	x
`small`	244M	x	x
`medium`	769M	x	x
`large`	1550M	NaN	x
`large-v2`	1550M	NaN	x
`large-v3`	1550M	NaN	x

3.2. Whisper for text extraction

Whisper [31] is a pretrained model designed for ASR and speech translation. Trained on 680k hours of labeled data, Whisper models demonstrate robust generalization across various datasets and domains without the need for fine-tuning. Whisper offers several model variants tailored to different tasks: `tiny`, `base`, `small`, `medium`, `large`, `large-v2`, and the latest, `large-v3`, which is used in this paper. In the same article, the authors of present a comprehensive analysis of Wisper's model variants emphasizing that model `large-v3` has better performances than model `large-v2` (see Table 1).

To integrate Whisper into proposed speaker diarization pipeline, the following setup and configuration steps were undertaken:

- **Installation**: Whisper was installed using the Transformers library. First, install the Transformers library from the GitHub repository[1];
- **Model Initialization**: the pre-trained model pipeline was initialized.

For implementing Whisper into out processing pipeline, we used this code example presented in [32], which demonstrates how to integrate and run Whisper on GPU for optimized performance. To achieve a better GPU utilization we adjust parameters like: temperature, batch size (`batch_size=16`), maximum number of tokens used in the model (`max_new_tokens=128`), and chunk lengths (`chunk_length_s = 30`).

The output is a dictionary containing the resulting text (`"text"`), segment-level details (`"segments"`), and detected language (`"language"`), which is automatically identified unless specified otherwise (by using the option `decode_options["language"]`). In our case, English is explicitly set as the default language since the dataset is entirely in English. Whisper's capability to detect language dynamically enhances its versatility.

3.3. Integration with language models

The language-model component is used as a post-processing module for speaker-attribution correction. The input to the model is the speaker-attributed transcript generated by the Pyannote–Whisper baseline. The output is expected to preserve the transcript as much as possible while improving the assignment of words or utterance fragments to speaker labels.

The task is intentionally constrained. The model is not asked to rewrite the transcript directly, but to perform line-level speaker reassignment. Each speaker-attributed line is converted into a JSON object containing a unique line identifier, the current speaker label, and the corresponding text. The prompt explicitly provides the set of allowed speaker labels and instructs the model to return only a JSON array that assigns one of the existing speaker labels to each input line. The model is not allowed to rewrite the text, add or remove words, merge or split lines, change punctuation, or invent new speaker labels. If the model is uncertain, it is instructed to keep the original speaker assignment. This formulation ensures that the lexical content of the transcript is preserved by construction, while allowing the model to correct speaker-attribution errors near diarization boundaries.

The system message used for the local LLM was: *"You correct only speaker labels. Return only valid JSON"*. The user prompt then specified the allowed speaker labels and the critical constraints of the task. In the zero-shot setting, the model receives only these instructions and the input JSON. In the one-shot setting, the prompt additionally includes a short example showing how an incorrectly assigned fragment can be reassigned to the appropriate speaker while preserving the original text.

```
System:
You correct only speaker labels. Return only valid JSON
     .

User:
You are correcting speaker attribution in an ASR
     diarization transcript.

Your task is ONLY to decide which existing speaker
     label should be assigned
to each existing line.

Allowed speaker labels:
SPEAKER_00, SPEAKER_01, ..., SPEAKER_NN

Critical rules:
- Do NOT rewrite the text.
- Do NOT add words.
- Do NOT remove words.
- Do NOT merge lines.
- Do NOT split lines.
- Do NOT change punctuation.
- Do NOT invent speaker labels.
- Use ONLY the allowed speaker labels.
- Keep every input id exactly once.
- If unsure, keep the original speaker.
- Return ONLY valid JSON.

Input JSON:
[
  {"id": 0, "speaker": "SPEAKER_00", "text": "..."},
  {"id": 1, "speaker": "SPEAKER_01", "text": "..."}
]

Output JSON:
[
  {"id": 0, "speaker": "SPEAKER_00"},
  {"id": 1, "speaker": "SPEAKER_01"}
]
```

Listing 1 Constrained speaker reassignment prompt used for LLM post-processing.

For the one-shot configuration, the same prompt is extended with an example in which a short fragment initially assigned to the wrong speaker is reassigned to the adjacent speaker. The example reinforces the central constraint of the task: the output may modify only the speaker labels, while the textual content is reconstructed from the original input lines and therefore remains unchanged.

To improve robustness and reduce context-length issues, long meeting transcripts are processed in chunks. Each chunk contains a fixed number of speaker-attributed lines. After correction, the chunks are concatenated to reconstruct the full transcript. Each corrected chunk is validated before being accepted. If the output does not contain valid speaker tags or if the lexical content differs excessively from the input after normalization, the system discards the language-model output and keeps the baseline transcript for that chunk.

This validation mechanism prevents artificial improvements caused by paraphrasing, deletion, or hallucination. Therefore, changes in cp-WER can be interpreted as the effect of conservative post-processing rather than unconstrained transcript rewriting.

3.4. Evaluation metrics

All word-level metrics are computed against manual AMI reference transcripts. We report two categories of metrics. First, diarization

[1] https://github.com/huggingface/transformers.git

quality is evaluated using Diarization Error Rate (DER) and Jaccard Error Rate (JER), complemented by Purity and Coverage. Second, transcription quality is evaluated using Word Error Rate (WER), while speaker-attributed transcription quality is evaluated using concatenated minimum-permutation Word Error Rate (cpWER).

While the Diarization Error Rate (DER) provides a convenient method for comparing different diarization approaches, it often falls short in fully capturing the types of errors committed by the system. Our objective is to develop a robust pipeline capable of achieving speaker diarization and text alignment for each speaker at the word level. DER and JER are computed using the reference speaker activity segments extracted from AMI manual annotations and the hypothesis speaker segments generated by Pyannote. In our experiments, DER is computed with a collar of 0.0 s and with overlapping speech included in the scoring region. Purity and Coverage are reported as secondary metrics because they provide additional insight into over-clustering and under-clustering behavior. Purity and Coverage [33] are two complementary evaluation metrics that offer deeper insights into system behavior, particularly in correctly identifying speaker clusters. Word Diarization Error Rate (WDER) [34] and concatenated minimum-permutation Word Error Rate (cpWER) [35] are micro-level metrics used to gauge the accuracy of Speech-to-Text (STT) systems.

WER is computed after removing speaker tags from both the reference and hypothesis transcripts. cpWER is computed by grouping words by speaker, concatenating each speaker's transcript, and selecting the speaker-label permutation that minimizes the global WER. This is necessary because diarization systems assign arbitrary speaker labels, and a direct comparison between `SPEAKER_00` and `SPEAKER_01` may be misleading.

We do not report WDER as a primary metric in the revised experiments because its reliable computation requires word-level speaker alignment between ASR hypotheses and manual references. Instead, we use WER and cpWER as the main word-level and speaker-attributed transcription metrics.

3.4.1. Diarization purity

If every label in a hypothesized annotation overlaps exclusively with segments belonging to a single reference label, then the annotation is considered perfectly pure. The purity metric is defined in Eq. (1).

$$\text{Purity} = \frac{\sum_{\text{cluster}} \max_{\text{speaker}} |\text{cluster} \cap \text{speaker}|}{\sum_{\text{cluster}} |\text{cluster}|} \tag{1}$$

where $|\text{cluster} \cap \text{speaker}|$ represents the duration of their intersection, where speaker denotes the total speech length of a specific reference speaker. Under-segmented results, such as when two speakers are merged into one large cluster, typically yield lower purity and higher coverage. Conversely, over-segmented results, where there are numerous speaker clusters, tend to yield higher purity and lower coverage.

3.4.2. Diarization coverage

When all segments from a particular reference label are clustered into the same cluster, a hypothesized annotation is said to have perfect coverage as shown in Eq. (2).

$$\text{Coverage} = \frac{\sum_{\text{speaker}} \max_{\text{cluster}} |\text{speaker} \cap \text{cluster}|}{\sum_{\text{speaker}} |\text{speaker}|} \tag{2}$$

where $|\text{cluster} \cap \text{speaker}|$ is the duration of their intersection, and cluster is the hypothesized cluster.

3.4.3. Word diarization error rate (WDER)

The word diarization error rate (WDER) quantifies the frequency with which words are incorrectly attributed to the wrong speaker relative to the actual data. WDER is a key metric used to assess the quality of diarization, where lower values indicate better diarization quality. We compute WDER using Eq. (3):

$$\text{WDER} = \frac{S_{IS} + C_{IS}}{\text{S} + \text{C}} \tag{3}$$

where:

1. S_{IS} is the quantity of ASR substitutions with incorrect speaker tokens;
2. C_{IS} is the amount of Correct ASR words with Incorrect Speaker tokens;
3. S is the total amount of ASR replacements;
4. C is the number of Correct ASR words.

3.4.4. Concatenated minimum-permutation word error rate (cpWER)

The concatenated minimum-permutation word error rate (cpWER) assesses the alignment of recognized words with ground truth transcripts considering all possible permutations of the recognized words. It is defined as follows:

1. Combine every speaker's transcript for reference and research purposes;
2. Determine the WER between the hypothesis's reference and every speaker permutation that might exist;
3. Select the permutation with the lowest WER, which is thought to be the best one, out of all of them.

4. Implementation details

4.1. Dataset description: AMI meeting corpus

The experimental setup uses the AMI Meeting Corpus, a multi-speaker meeting corpus designed for research on meeting transcription, speaker diarization, and multimodal interaction analysis. AMI contains face-to-face meetings with multiple participants, overlapping speech, variable turn-taking patterns, and realistic meeting-room acoustics.

We evaluate the proposed pipeline on a subset of the AMI Meeting Corpus containing 143 meetings with manual word-level speaker annotations. The subset comprises 77.63 meeting-hours and 732,982 reference words. The annotated speech accounts for 61.98 h, including 7.57 h of overlapping speech, corresponding to an average overlap ratio of 11.85%. Each meeting contains between 4 and 5 speakers, with an average of 4.02 speakers per meeting.

To analyze the robustness of the pipeline under different acoustic conditions, we consider three AMI audio streams: Headset Mix, Individual Headsets, and Microphone Array recordings (see Table 2). The Headset Mix condition provides one mixed close-talking audio stream per meeting and represents the cleanest single-stream multi-speaker setting. The Individual Headsets condition contains separate close-talking microphone recordings for each participant, resulting in 314.10 channel-hours. The Microphone Array condition contains far-field table-top microphone recordings, totaling 1204.57 channel-hours, and represents a more challenging setting due to distance from the speakers, room acoustics, reverberation, and overlapping speech.

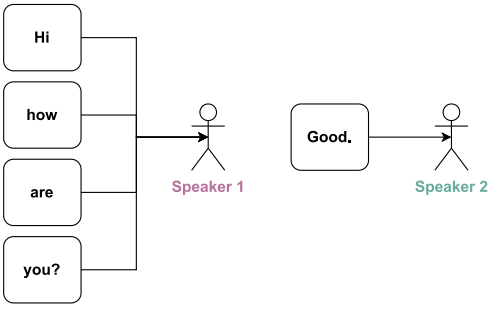
For the experiments reported in this paper, we use the Headset Mix audio condition as the primary input stream. The individual-headset and microphone-array conditions are retained for future robustness experiments.

Manual AMI annotations are distributed in XML format and include word-level timing information for each speaker. These annotations are parsed to extract the word text, speaker identifier, start time, and end time. Consecutive words produced by the same speaker are merged into speaker turns when the temporal gap between them is below a fixed threshold of 0.8 s. This produces two reference objects for each meeting: a speaker-attributed reference transcript and a list of reference speaker activity segments.

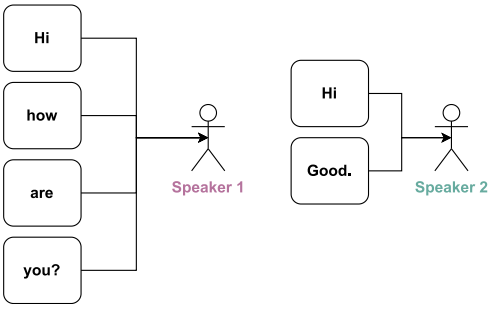
Because the manual annotations do not always start at time zero, evaluation is restricted to the time interval covered by the reference annotations. For each meeting, the reference interval $[t_{\text{start}}, t_{\text{end}}]$ is defined in Eq. (4), where s_i and e_i denote the start and end times of the manually annotated speaker segments. Hypothesis segments outside this interval are excluded from text-based evaluation. This avoids

Table 2
Summary of the AMI evaluation subset across the three audio recording conditions. Meeting hours denote the actual duration of the meetings, while channel hours denote the cumulative duration across all available audio channels/files.

Audio condition	Meetings	Files	Meeting h	Channel h	Avg. spk.	Spk. range	Ref. words	Speech h	Overlap
Headset mix	143	143	77.63	77.63	4.02	4–5	732,982	61.98	11.85%
Individual headsets	143	575	77.63	314.10	4.02	4–5	732,982	61.98	11.85%
Microphone array	143	2189	77.63	1204.57	4.02	4–5	732,982	61.98	11.85%



(a) Illustration of Whisper's tendency to generate context independently.



(b) The proposed method's correction using LLM context.

Fig. 2. Context generated by Whisper, despite Pyannote's segmentation.

penalizing the system for speech or audio regions that are not covered by the manual reference transcript.

$$t_{\text{start}} = \min_i\{s_i\}, \quad t_{\text{end}} = \max_i\{e_i\} \tag{4}$$

The experimental setup we use is built upon commodity hardware equipped with a NVIDIA RTX 3060 GPU with 12 GB of VRAM. Despite the performances of our setup, the full baseline evaluation for the Headset Mix condition took 652.65 min (real-time). The evaluated pipeline include Pyannote diarization and Whisper `base.en` transcription, without language-model post-processing.

Despite Pyannote's segmentation, Fig. 2 shows how Whisper sometimes generates context on its own because of brief discussion pauses. The suggested approach makes use of contextual data from the LLM in an attempt to remedy this.

```
from datasets import load_dataset
ds = load_dataset("talkbank/callhome", "eng")
dataset = ds["data"]
for audio_number, row in enumerate(dataset):
    audio_data = row["audio"]
    speakers_GT = row["speakers"]
    audio_array = audio_data["array"]
    sampling_rate = audio_data["sampling_rate"]
    print(type(audio_array))
    print(type(speakers_GT))
```

Listing 2 WAV conversion.

To convert the `audio_array` to a WAV file, which is of type `numpy.ndarray`, and obtain the `sampling_rate`, we used the `soundfile` library as show in Listing 2. Listing 3 normalizes the audio data (`audio_array`) to a 16-bit integer format suitable for saving as a WAV file.

```
audio_array_normalized = np.int16(audio_array * 32767)
output_wav_file = os.path.join(
    os.path.dirname(output_csv_path),
    os.path.splitext(output_csv_path)[0] + "_audio.wav"
    )
sf.write(
    output_wav_file,
    audio_array_normalized,
    sampling_rate,
    subtype="PCM_16")
```

Listing 3 Converting array data to WAV file.

4.2. Software requirements and environment setup

For this project, we utilized Anaconda with an image of `devel cuda 11.6.0 cudnn miniconda`, ensuring compatibility with the required CUDA and cuDNN versions for GPU acceleration. The following outlines the key software and versions used:

- *Python Version*: 3.10.12;
- *Anaconda*: managed environments and dependencies;
- *Torch*: required for PyTorch-based applications;
- *Environment*: configured with the latest libraries and dependencies to support machine learning tasks in speech processing.

We propose the following execution environment:

- *Virtual Machine*: all computations and processes were conducted on a virtual machine environment, ensuring isolated and controlled execution;
- *Continuous Inference*: the pipeline operated continuously with a consistent temperature setting, ensuring stable performance without manual intervention;
- *Checkpoint Management*: checkpoints were implemented to safeguard against unforeseen errors, enabling recovery without restarting from the beginning.

4.3. Implementation of pyannote

The implementation of Pyannote for Speaker Diarization involves several critical steps, from initializing the diarization pipeline to processing and converting the diarization results. Below, we detail the implementation process.

Initialization of Pyannote. The initial phase of the implementation involves the instantiation of the Pyannote framework, as detailed in Listing 4. This process begins with the retrieval of a state-of-the-art, pretrained speaker diarization model from the Hugging Face Hub, ensuring the integration of high-performance neural architectures. Beyond simple model loading, this stage encompasses the configuration of secure authentication protocols and the strategic optimization of the computational environment. Specifically, the pipeline is mapped to utilize GPU acceleration, a critical step to ensure high-throughput processing and minimize inference latency during the analysis of extensive audio datasets.

```
class PyannoteProcessor:
    """
    Class to perform Speaker Diarization using
    the Pyannote library.
    """
    def __init__(self):
        self.pipeline = Pipeline.from_pretrained(
            "Pyannote/speaker-diarization-3.1",
              use_auth_token="your_huggingface_token")
```

Listing 4 `PyannoteProcessor` class.

The `Pipeline.from_pretrained` function loads the pre-trained diarization model. The `use_auth_token` parameter is required to access the model on the Hugging Face model hub. Make sure to replace "`your_huggingface_token`" with your actual Hugging Face authentication token.

Performing Diarization. Once the pipeline is initialized, the next step is to perform diarization on the given audio file. This involves passing the audio file through the Pyannote pipeline, which outputs the diarization results. The results are then saved in RTTM format using Listing 5.

```
def perform_diarization(self, audio_file_path):
    self.pipeline.to(torch.device('cuda'))
    \textcolor{blue}{diarization = self.pipeline(
        audio_file_path, num_speakers=num_speakers)}
    with open("sample.rttm", "w") as rttm:
        diarization.write_rttm(rttm)
```

Listing 5 Performing diarization with Pyannote.

For AMI meetings, the number of speakers is constrained using the known speaker range of the evaluation subset, which contains between four and five speakers per meeting.

Converting RTTM to DataFrame. The diarization results are saved in RTTM format, which needs to be processed further. The RTTM file is read and converted into a DataFrame for easier manipulation and analysis as shown in Listing 6.

```
def rttm_to_dataframe(self, rttm_file_path):
    columns = [
        "Type", "File␣ID",
        "Channel", "Start␣Time",
        "Duration", "Orthography",
        "Confidence", "Speaker",
        "x", "y",]
    data = []
    with open(rttm_file_path, "r") as rttm_file:
        lines = rttm_file.readlines()
        data = [line.strip().split() for line in lines]
    df = pd.DataFrame(data, columns=columns)
    df = df.drop(["x", "y", "Orthography", "Confidence"
        ], axis=1)
    return df
```

Listing 6 Converting RTTM to DataFrame.

The `rttm_to_dataframe` method reads the RTTM file, processes each line to extract relevant information, and stores it in a Pandas DataFrame. Unnecessary columns are dropped to retain only essential information, such as Type, File ID, Channel, Start Time, Duration, and Speaker.

Complete Workflow Integration. To integrate the Pyannote processor within a complete workflow, an `AudioProcessor` class is used. It manages interactions between Pyannote and other components like Whisper and the language model.

The `process_and_append_to_csv` method in the `AudioProcessor` class integrates Pyannote for diarization, normalizes audio data, transcribes using Whisper, and integrates the language model for diarization correction.

Segmentation Refinement. To enhance data interpretability, consecutive segments attributed to the same speaker are merged into cohesive intervals. This step ensures that uninterrupted speech segments by the same speaker are treated as continuous units for analysis.

By following these steps, Pyannote is effectively implemented for Speaker Diarization, leveraging its capabilities for accurate and efficient speaker segmentation.

4.4. Implementation of whisper

Whisper is used on chunks of audio that were previously saved after processing with Pyannote. Since we aimed to create a reliable Speaker Diarization model, we opted for the best model available in Whisper, which is the 1550M model. Given its size, it was essential to utilize GPU acceleration. For this purpose, we employed the NVIDIA GeForce RTX 4090. Another crucial setup involved setting the temperature to 0. This ensures that there are no hallucinations during the language model's processing of input data.

Segmenting the audio. To process specific segments of audio, we implemented the `process_audio_segment` method. This method extracts a segment from the specified audio file (`audio_file`) based on the provided `start_time` and `end_time` parameters. It then transcribes the segment using `transcribe_audio_with_whisper`, collapses the resulting transcription segments using `collapse_segments`, and removes temporary audio files once processing is complete. Our goal was to optimize efficiency throughout this process as shown in Listing 7.

```
def process_audio_segment(self, audio_file, start_time,
     end_time,
    detected_language):
    start_time = float(start_time * 1000)
    end_time = float(end_time * 1000)
    audio = AudioSegment.from_file(audio_file)
    audio_segment = audio[start_time:end_time]
    audio_segment_path = f"audio_segment_{start_time}.
        wav"
    audio_segment.export(audio_segment_path, format="
        wav")

    # Transcribe the audio segment
    transcription_result = self.
        transcribe_audio_with_whisper(
    audio_segment_path, detected_language)
    whisper_transcript = transcription_result["text"]

    # Split the transcript into segments
    segments = whisper_transcript.split("\n")

    # Collapse the segments
    collapsed_transcript = self.collapse_segments(
        segments)

    # Delete the temporary audio segment file
    os.remove(audio_segment_path)
    return collapsed_transcript
```

Listing 7 Usage of pre-trained Whisper.

Combining the results. The `collapse_segments` method processes individual transcript segments by combining words and spaces into a cohesive transcript format as shown in Listing 8. This ensures the readability and coherence of the transcribed output from Whisper ASR.

```
def collapse_segments(self, transcript_segments):
    segment_counter = 0
    collapsed_segments = []
    for segment in transcript_segments:
        if segment.startswith("Segment"):
            segment_counter += 1
            collapsed_segments.append(segment)
        else:
            words = segment.split()
            for word in words:
                collapsed_segments.append(word)
                collapsed_segments.append("␣")
    collapsed_transcript = "".join(collapsed_segments)
    return collapsed_transcript
```

Listing 8 Collapse transcript segments into a single string.

Table 3
Approximate normalized transcript-level results for constrained language-model post-processing. The filtered setting accepts only conservative outputs that preserve speaker-tag format and lexical fidelity.

Method	Model/setup	Filtering	norm-WER ↓	norm-cpWER ↓	Token F1 ↑	Compression	Rep. red. ↑
Baseline	None	None	≈0.50	≈0.52	≈0.86	1.00	0.00
LLM	Qwen2.5-3B zero-shot	No	≈0.82	≈0.83	≈0.42	≈0.33	≈0.75
LLM	Qwen2.5-3B one-shot	No	≈0.70	≈0.70	≈0.59	≈0.45	≈0.51
LLM	Qwen2.5-7B zero-shot	No	≈0.62	≈0.65	≈0.70	≈0.68	≈0.40
LLM	Qwen2.5-7B one-shot	No	≈0.55	≈0.58	≈0.78	≈0.78	≈0.35
LLM	Qwen2.5-7B one-shot	Yes	≈0.49	≈0.51	≈0.85	0.90–0.98	≈0.22

The `process_and_append_to_csv` method (from the `AudioProcessor` class) integrates the Whisper result as a final step of diarization. It handles the dataset, processes each audio file, and performs the necessary operations. Additionally, the language is hard-coded as English because the entire dataset is in English. However, leaving the language parameter blank or None is also possible since Whisper can identify the language automatically, even when different languages are spoken in the same conversation. By following these steps, Whisper is effectively implemented for Speaker Diarization, leveraging its capabilities for accurate and efficient Speech-to-Text conversion.

We investigate the efficacy of the Qwen2.5 architectural family for transcript refinement, specifically evaluating the 3B and 7B instruction-tuned variants without further fine-tuning. This selection allows for a comparative analysis of model scale versus performance: the 3B configuration target scenarios with restricted computational resources, while the 7B variant offers enhanced linguistic capacity and stricter adherence to instruction-following constraints, while still remaining viable for local, privacy-preserving inference.

The robustness of these models is assessed through two prompting paradigms:

- **Zero-shot prompting**: evaluating the models' inherent ability to perform the refinement task based solely on the instruction set;
- **One-shot prompting**: enhancing the context with a task-specific demonstration that illustrates the transformation of speaker-attributed fragments. This example explicitly reinforces the constraint against free-form rewriting, emphasizing the preservation of lexical integrity.

By maintaining a consistent model family and varying the prompting depth, we isolate the impact of model capacity and in-context learning on the stability of the speaker-attribution correction process.

5. Experimental results

5.1. Experimental setup

The baseline system consists of Pyannote speaker diarization followed by Whisper ASR applied to diarized segments. We then evaluate constrained language-model post-processing using open-weight instruction-tuned models. Each language model is tested in zero-shot and one-shot configurations.

The evaluated configurations are:

- `without_LM`: Pyannote diarization followed by Whisper transcription, without language-model correction;
- `Qwen2.5-3B_zero_shot`: constrained correction using Qwen2.5-3B-Instruct with a zero-shot prompt;
- `Qwen2.5-3B_one_shot`: constrained correction using Qwen2.5-3B-Instruct with a one-shot prompt;
- `Qwen2.5-7B_zero_shot`: constrained correction using Qwen2.5-7B-Instruct with a zero-shot prompt;
- `Qwen2.5-7B_one_shot`: constrained correction using Qwen2.5-7B-Instruct with a one-shot prompt;
- `Qwen2.5-7B_one_shot_filtered`: one-shot correction with conservative validation and acceptance filtering.

For the current experiments we have used the headset-mix stream, corresponding to files of the form `*.Mix-Headset.wav`. This choice provides a single audio stream suitable for the Pyannote–Whisper pipeline while preserving the realistic multi-speaker meeting setting. For each AMI meeting, we downloaded the headset-mix audio, the individual close-talking headset recordings, and the microphone-array recordings. The individual-headset recordings can be used for oracle-style or speaker-isolated analyses, while the microphone-array recordings provide a more challenging far-field condition and are reserved for future robustness experiments. So, for each meeting, manual word-level annotations have been parsed to build the reference transcript and the reference speaker activity segments. Furthermore, the system has been evaluated only on the annotated time interval of each meeting.

The language-model outputs have been validated before evaluation to prevent artificial improvements such as paraphrasing, deletion, or hallucination. In the case when the corrected transcript changes the lexical content excessively or fails to preserve the required speaker-tag format, the output is rejected and the baseline transcript is retained (see Table 3).

5.2. Language-model post-processing results

The used model not only removes repetitions but also deletes or paraphrases valid content, as naive LLM post-processing substantially reduces repetitive ASR artifacts. We must note that this reduction comes at the cost of lexical fidelity. In particular, the unfiltered Qwen2.5-3B variants strongly compress the transcript and remove many repeated tokens, but they also increase normalized WER and normalized cpWER and decrease token-level F1.

We managed to keep normalized WER and normalized cpWER close to the baseline while still reducing a meaningful fraction of repetitive ASR artifacts by leveraging the filtered strategy used by LLM, which accepts only conservative edits that preserve speaker labels and lexical content. This indicates that open-weight LLMs are more reliable as selective transcript post-processors than as unconstrained transcript rewriters.

Results show that the main benefit of the LLM component is conservative transcript cleanup under explicit fidelity constraints, not direct ASR correction, because unconstrained LLM correction is unsafe for meeting transcription, especially when evaluation is based on WER and cpWER. However, an acceptance-based post-processing strategy can improve transcript readability by reducing repetitive ASR artifacts while preserving lexical accuracy.

5.3. Baseline diarization and transcription results

The baseline performance of the integrated Pyannote–Whisper pipeline on the AMI Headset Mix condition is presented in Table 4. By utilizing the pyannote/speaker-diarization-3.1 model in conjunction with the Whisper base.en architecture, the system processed the 143-meeting corpus in approximately 10.9 h of wall-clock time.

For the Headset Mix condition, the Pyannote–Whisper baseline obtained a WER of 0.802 and a cpWER of 0.790. Also, the diarization component achieved a DER of 0.160 and a JER of 0.209, with a diarization purity of 0.911 and coverage of 0.886. The full evaluation over 143 AMI meetings required approximately 10.9 h using Whisper base.en and pyannote/speaker-diarization-3.1.

Table 4
Baseline Pyannote–Whisper performance on the AMI Headset Mix condition over 143 meetings. The baseline uses pyannote/speaker-diarization-3.1 and Whisper base.en, without language-model post-processing.

Metric	Headset Mix baseline
WER ↓	0.802
cpWER ↓	0.790
DER ↓	0.160
JER ↓	0.209
Purity ↑	0.911
Coverage ↑	0.886
Runtime	652.65 min

6. Conclusions

In this paper, we have proposed an offline processing pipeline for speaker attribution from meeting transcriptions based on Pyannote-based speaker diarization, Whisper-based automatic speech recognition, and constrained open-weight language-model post-processing. Furthermore, acknowledging the limitations of the evaluation protocols based on generated references, we propose a revised methodology that uses manual AMI annotations to construct speaker-attributed reference transcripts and speaker activity segments.

To overcome a higher intrusion of the language-model component into the processing pipeline, which could lead to excessive modification of the transcript, it was introduced as a constrained post-processing task. So, instead of allowing the model to freely rewrite the transcript, the model was instructed to apply only conservative candidate corrections, which primarily target speaker-attribution errors and repetitive ASR artifacts. Moreover, a validation and acceptance mechanism rejects outputs that do not preserve the required speaker-tag format.

The experimental results are based on a protocol that evaluates diarization quality and speaker-attributed transcription qualit, leveraging metrics to assess speaker segmentation such as DER and JER and also, metrics to assess transcription and speaker-attributed transcription performance like WER and cpWER. The experimental protocol evaluates both diarization quality and speaker-attributed transcription quality. A more rigorous evaluation than the initial setup has been achieved by using ASR-generated transcripts as references.

The obtained results show the importance of separating diarization errors from ASR errors, as the baseline Pyannote–Whisper system achieves reliable speaker segmentation on the AMI Headset Mix condition, but word-level performance remains affected by ASR errors, segmentation choices, and short or fragmented diarized regions. Naive language-model post-processing can reduce repetitive artifacts but may degrade lexical fidelity, whilst conservative filtering preserves WER and cpWER close to the baseline while still improving transcript readability.

As future work, we will focus on three main directions. First, we will extend the evaluation to additional AMI acoustic conditions, especially far-field microphone-array recordings. Second, we will compare the proposed pipeline against stronger diarization baselines such as VBx and EEND-based systems. Third, we will investigate streaming variants of the pipeline with explicit latency constraints, allowing the method to be evaluated under real-time operating conditions.

CRediT authorship contribution statement

Costin-Alexandru Deonise: Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Conceptualization. **Taisia-Maria Coconu:** Writing – review & editing, Writing – original draft, Visualization, Validation, Conceptualization. **Muhammad Khurram Zahur Bajwa:** Writing – original draft, Formal analysis, Data curation. **Catalin Negru:** Writing – review & editing, Validation, Conceptualization. **Bodgan-Costel Mocanu:** Writing – review & editing, Visualization, Validation, Supervision, Methodology, Formal analysis. **Aniello Castiglione:** Writing – review & editing, Validation. **Florin Pop:** Writing – review & editing, Resources, Project administration, Methodology, Formal analysis, Supervision.

Funding

The authors did not receive support from any organization for the submitted work.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially supported by the project HRIA: Romanian Hub for Artificial Intelligence, Smart Growth, Digitization and Financial Instruments Program, 2021–2027, MySMIS no. 334906 and DACISLab: Virtual Laboratory on Open Data and Open Science in the New Generation of Continuum Computing Systems, project number PN-IV-PCB-RO-MD-2024-0364, within PNCDI IV.

Data availability

No data was used for the research described in the article.

References

[1] T.J. Park, N. Kanda, D. Dimitriadis, K.J. Han, S. Watanabe, S. Narayanan, A review of speaker diarization: Recent advances with deep learning, Comput. Speech Lang. 72 (2022) 101317.
[2] C. Zhang, Q. Wang, Speaker diarization: A journey from unsupervised to supervised approaches, in: Odyssey: The Speaker and Language Recognition Workshop, 2022, p. 5.
[3] M.K.Z. Bajwa, C. Negru, B.-C. Mocanu, A. Castiglione, C. Pero, Near real-time deepfake audio detection based on speaker diarization techniques, in: 2025 25th International Conference on Control Systems and Computer Science, CSCS, IEEE, 2025, pp. 412–419.
[4] G. Efstathiadis, V. Yadav, A. Abbas, LLM-based speaker diarization correction: A generalizable approach, Speech Commun. 170 (2025) 103224.
[5] M.K.Z. Bajwa, A. Castiglione, C. Pero, Mel spectrogram-based CNN framework for explainable audio deepfake detection, in: L. Barolli (Ed.), Advanced Information Networking and Applications, Springer Nature Switzerland, Cham, 2025, pp. 407–416.
[6] B. McFee, C. Raffel, D. Liang, D.P. Ellis, M. McVicar, E. Battenberg, O. Nieto, Librosa: Audio and music signal analysis in python, SciPy 2015 (2015) 18–24.
[7] S.W. Chin, K.P. Seng, L.M. Ang, K.H. Lim, Improved voice activity detection for speech recognition system, in: 2010 International Computer Symposium, ICS2010, Tainan, Taiwan, 2010, pp. 518–523.
[8] C.A. Deonise, T.M. Coconu, T. Rebedea, F. Pop, Improved speech activity detection model using convolutional neural networks, in: 2023 22nd RoEduNet Conference: Networking in Education and Research, RoEduNet, IEEE, 2023, pp. 1–8.
[9] S. Sharma, P. Rattan, A. Sharma, Recent developments, challenges, and future scope of voice activity detection schemes—a review, in: Information and Communication Technology for Competitive Strategies (ICTCS 2020) Intelligent Strategies for ICT, Springer, 2021, pp. 457–464.
[10] W. Kraaij, T. Hain, M. Lincoln, W. Post, The AMI meeting corpus, in: Proc. International Conference on Methods and Techniques in Behavioral Research, 2005, pp. 1–4.
[11] S. S, D. Jayaram, V.R. Ajith, J. Jacob Mathew, R. Rajan, Speaker diarization using X-vectors-DNN framework, in: 2023 IEEE 20th India Council International Conference, INDICON, 2023, pp. 317–321.
[12] A. Ayasi, J. Joshy, R. Rajan, Speaker diarization using BiLSTM and BiGRU with self-attention, in: 2022 Second International Conference on Next Generation Intelligent Systems, ICNGIS, 2022, pp. 1–5.
[13] L.E. Shafey, H. Soltau, I. Shafran, Joint speech recognition and speaker diarization via sequence transduction, 2019, arXiv:1907.05337.
[14] T. Von Neumann, C. Boeddeker, T. Cord-Landwehr, M. Delcroix, R. Haeb-Umbach, Meeting recognition with continuous speech separation and transcription-supported diarization, in: 2024 IEEE International Conference on Acoustics, Speech, and Signal Processing Workshops, ICASSPW, 2024, pp. 775–779.

[15] G. Morrone, S. Cornell, L. Serafini, E. Zovato, A. Brutti, S. Squartini, End-to-end integration of speech separation and voice activity detection for low-latency diarization of telephone conversations, Speech Commun. 161 (2024) 103081.
[16] A. Polok, D. Klement, M. Kocour, J. Han, F. Landini, B. Yusuf, M. Wiesner, S. Khudanpur, J. Černocký, L. Burget, DiCoW: Diarization-conditioned whisper for target speaker automatic speech recognition, Comput. Speech Lang. 95 (2026) 101841.
[17] W. Beccaro, M. Arjona Ramírez, W. Liaw, H.R. Guimarães, Analysis of oral exams with speaker diarization and speech emotion recognition: A case study, IEEE Trans. Educ. 67 (1) (2024) 74–86.
[18] T. Kasakowskij, J.M. Haake, T3 Talk2Text - a model for near real-time voice transcription in virtual group meetings, Discov. Educ. 4 (1) (2025) 146.
[19] A. Picovoice, A. Kenarsari, D. Bartle, R. Rostam, Picovoice/porcupine: On-device wake word detection powered by deep learning, 2018, GitHub. https://github.com/Picovoice/porcupine.
[20] N. Blum, S. Lachapelle, H. Alvestrand, WebRTC: real-time communication for the open web platform, Commun. ACM 64 (8) (2021).
[21] D. Doukhan, E. Lechapt, M. Evrard, J. Carrive, Ina's mirex 2018 music and speech detection system, in: Music Information Retrieval Evaluation Exchange, MIREX 2018, 2018.
[22] Y. Wang, A. Alhmoud, S. Alsahly, M. Alqurishi, M. Ravanelli, Calm-whisper: Reduce whisper hallucination on non-speech by calming crazy heads down, 2025, arXiv preprint arXiv:2505.12969.
[23] S. Hernawan, Speaker diarization: Its developments, applications, and challenges, in: Proceedings Intl Conf Information System Business Competitiveness, 2012, pp. 255–259.
[24] J. Wang, Z. Du, S. Zhang, Told: A novel two-stage overlap-aware framework for speaker diarization, in: ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, IEEE, 2023, pp. 1–5.
[25] J. Wang, Z. Du, S. Zhang, TOLD: A novel two-stage overlap-aware framework for speaker diarization, 2023, arXiv:2303.05397.
[26] S. Horiguchi, Y. Fujita, S. Watanabe, Y. Xue, P. Garcia, Encoder-decoder based attractors for end-to-end neural diarization, IEEE/ACM Trans. Audio Speech Lang. Process. 30 (2022) 1493–1507.
[27] M. Ravanelli, T. Parcollet, P. Plantinga, A. Rouhe, S. Cornell, L. Lugosch, C. Subakan, N. Dawalatabad, A. Heba, J. Zhong, et al., SpeechBrain: A general-purpose speech toolkit, 2021, arXiv preprint arXiv:2106.04624.
[28] A. Plaquet, H. Bredin, Powerset multi-class cross entropy loss for neural speaker diarization, 2023, arXiv preprint arXiv:2310.13025.
[29] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Lu, et al., Qwen2. 5-coder technical report, 2024, arXiv preprint arXiv:2409.12186.
[30] H. Bredin, R. Yin, J.M. Coria, G. Gelly, P. Korshunov, M. Lavechin, D. Fustes, H. Titeux, W. Bouaziz, M.P. Gill, Pyannote. audio: neural building blocks for speaker diarization, in: ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, IEEE, 2020, pp. 7124–7128.
[31] A. Radford, J.W. Kim, T. Xu, G. Brockman, C. McLeavey, I. Sutskever, Robust speech recognition via large-scale weak supervision, 2022, arXiv:2212.04356.
[32] J.R. Batista, Learn Openai Whisper: Transform Your Understanding of Genai Through Robust and Accurate Speech Processing Solutions, Packt Publishing Ltd, 2024.
[33] A. Xu, T. Feng, H. Tager-Flusberg, C. Lord, S. Narayanan, Data efficient child-adult speaker diarization with simulated conversations, in: ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, IEEE, 2025, pp. 1–5.
[34] L.E. Shafey, H. Soltau, I. Shafran, Joint speech recognition and speaker diarization via sequence transduction, in: Proc. Interspeech, 2019, pp. 396–400.
[35] S. Watanabe, M. Mandel, J. Barker, E. Vincent, A. Arora, X. Chang, S. Khudanpur, V. Manohar, D. Povey, D. Raj, et al., CHiME-6 challenge: Tackling multispeaker speech recognition for unsegmented recordings, 2020, arXiv preprint arXiv:2004.09249.