Interspeech 2024
1-5 September 2024, Kos, Greece

gryannote open-source speaker diarization labeling tool

Clément Pagés, Hervé Bredin

IRIT, Université de Toulouse, CNRS, Toulouse INP, UT3, Toulouse, France
clement.pages@irit.fr herve.bredin@irit.fr

Abstract

gryannote is a collection of *Gradio* custom components focusing on the labeling of speaker diarization data. Integrated with the *pyannote* speaker diarization ecosystem, it allows to build web applications to load pretrained *pyannote* pipelines and customize their hyper-parameters, upload or record an audio file, process it with the pipeline, visualize and interact with its outputs, correct them if needed, and export the final annotation in RTTM format. Each of these components can be used independently from each other.
Index Terms: speaker diarization, annotation process, pyannote, gradio

1. Introduction

Speaker diarization is the task of partitioning audio into speaker turns according to the identity of the speaker, basically answering the question "who spoke when?". Speaker diarization systems are often used as a pre-processing step for automatic speech recognition. *pyannote.audio* [1, 2] is an open-source Python library focused on the speaker diarization task (including voice activity or overlapped speech detection), providing dedicated pretrained models and pipelines. The main speaker diarization pipeline is made of three steps: local segmentation of an audio into speaker turns, extraction of speaker embeddings from these segments, and finally clustering of the speaker turns. It relies on deep learning models trained on large amount of data, labeled as accurately as possible, which is a very costly process (in terms of human resources mostly). Therefore, it is critical for annotation tools to allow fast and accurate labeling. That is the main purpose of gryannote [1], a collection of Gradio custom components. The rest of this paper introduces this framework, before describing in further details the annotation application and its different components.

2. Gradio framework

Gradio is an open-source Python package designed to build web application interface [3]. While *Streamlit* focuses mostly on data visualization and dashboard web applications, *Gradio* is more suitable for machine learning purposes. It relies on a set of graphic components that can be considered as subsystems taking inputs from the user interface or other components, and predicting one or more outputs according to these inputs. Each component allows interaction between the user and the machine learning part (in our case the pyannote.audio library). These components are divided into two distinctive parts: the backend and the frontend. On one side, the backend is for the logical and data processing, and is implemented in Python. On the other side, the frontend is used to retrieve user inputs from the interface and shows prediction results provided by the backend. It is based on the Svelte framework. One of the strengths of Gradio is that it is possible to build custom components with specific behavior and user interface. These customs components can in turn be deployed as web applications or used in Jupyter notebooks, for example.

[1] https://github.com/clement-pages/gryannote

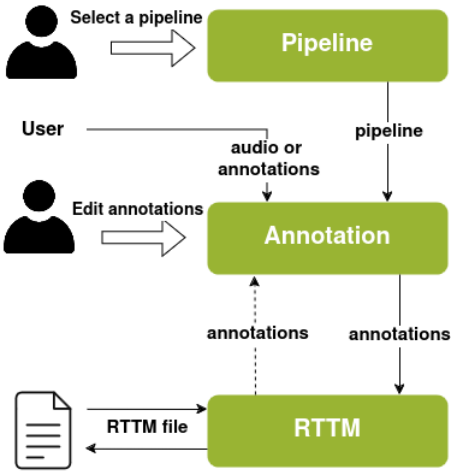

Figure 1: *Possible interactions between the user and gryannote components. Dotted arrows indicate planned (but not yet available) features.*

3. Custom components

gryannote consists of three distinct custom components, named *pipeline*, *annotation*, and *rttm*, and described in the following sections. Figure 1 represents how they interact between them and with the user, in terms of input and outputs. First, the user selects a pipeline and its configuration hyper-parameters. Then, one can apply this pipeline to audio and optionally correct automatic annotations manually. When the user is satisfied with the result, they can finally download the corresponding annotations in Rich Transcription Time Marked (RTTM) format for future reuse. A view of the complete *gryannote* interface is depicted in Figure 2, and shows how these components are visually organized on the screen. The three components are independent from each other and can be used in a standalone manner if needed.

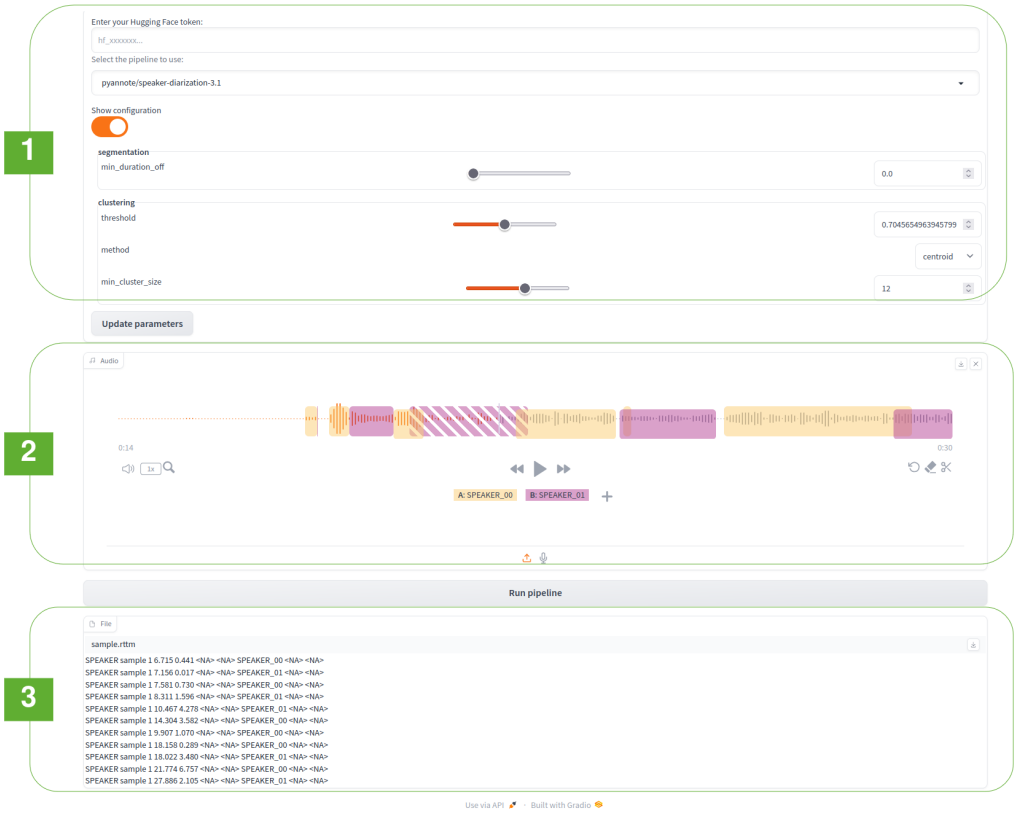

Figure 2: *Screen of the app's interface. (1): pipeline, (2): annotation, (3): rttm*

3.1. Pipeline selection

The *pipelineselector* component is used to select a pretrained pipeline used to produce the first (automatic) version of the annotations. This pipeline can be supplied in two different ways: either the user provides a predefined pyannote pipeline instance, or chooses it interactively with the user interface. In the latter case, the user chooses a pipeline from a dropdown list, automatically filled by all supported pyannote pipelines available on HuggingFace Hub. The component also allows the user to modify the hyper-parameters of the selected pipeline.

3.2. Annotation of the audio

This component lets the user apply the selected pipeline on an audio recording. This audio is either loaded from disk or generated directly using the microphone. The component then displays the annotations predicted automatically by the pipeline. These annotations can be edited by the user, who can modify the start and end times of each annotation, as well as adding and deleting annotations, and splitting annotations. The user can also modify the label of each annotation. Each of these functions is associated with a keyboard shortcut, enabling the user to make the annotation process more efficient. Rather than playing an audio file, it is also possible to use the recording function to directly record an audio conversation, and then obtain the corresponding annotations. Alternatively, the component can take previously produced annotations as input. In this case, the component displays the annotations, which can also be modified by the user.

3.3. Export in RTTM format

This component allows the user to export annotations in RTTM format. These annotations are dynamically updated according to changes made in the *annotatedaudio* component. The component allows one to download the RTTM file, but also to upload an RTTM file to visualize annotations for instance.

4. Future work

We plan to add the ability to use the application, especially the *annotatedaudio* component, in a streaming mode. We also aim to use it in the context of interactive speaker diarization, where the user can correct the annotations produced by a diarization system, letting the system provide a new prediction that takes the correction into account.

5. References

[1] H. Bredin, "pyannote.audio 2.1 speaker diarization pipeline: principle, benchmark, and recipe," in *Proc. INTERSPEECH 2023*, 2023.

[2] A. Plaquet and H. Bredin, "Powerset multi-class cross entropy loss for neural speaker diarization," in *Proc. INTERSPEECH 2023*, 2023.

[3] A. Abid, A. Abdalla, A. Abid, D. Khan, A. Alfozan, and J. Zou, "Gradio: Hassle-Free Sharing and Testing of ML Models in the Wild," Jun. 2019, arXiv:1906.02569 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1906.02569