Pyannote Colab Issues

Today I want to ask you about handling, managing, adopting, or deploying a tool-kit/library called "pyannote." Recently, my team decided to go with Pyannote for Speaker Diarization task, however our Colab/CUDA/python/Chrome(or Edge) seem to have troubles with Pyannote. What is up with pyannote tool-kit/library? Why isn't it working properly on Colab? Do you know about this, by any chance?

Short answer: nothing's "wrong" with pyannote itself — it's that Colab's default stack (Python/Torch/CUDA/decoders) often doesn't match what the **current** pyannote pipelines expect, and recent pyannote releases changed a few things (names, tokens, deps). Those two together cause most "it doesn't work on Colab" headaches.

Here's what's tripping teams up — and how to fix it fast.

What changed recently (and why your old notebook breaks)

- **New pipeline names & API in v4 (2025):** there's now an open-source pipeline `pyannote/speaker-diarization-community-1` and a paid **precision** service. The v4 quick-start uses `token=...` (not `use_auth_token=`) and the result object changed a bit. If your code was written for 3.1, it will error or silently run on CPU. PyPI
- **Model access is gated:** even the "community" pipeline is a gated HF repo. You must accept the model's terms and use a Hugging Face token (or you get 403 / "not authorized"). PyPI
- **FFmpeg is now required:** v4 relies on PyTorch's `torchcodec` which needs a system `ffmpeg`. Colab VMs don't have it by default. PyPI
- **3.1 still exists (legacy):** if you stick to `pyannote/speaker-diarization-3.1`, you still need to accept that model's terms and pass a HF token. Code samples on its card show the older API (`use_auth_token=`). Hugging Face+1

Why it "doesn't use the GPU" or looks idle

- If Torch is the **CPU wheel** or CUDA/Torch are mismatched, pyannote falls back to CPU (GPU stays at 0%). You must install the **CUDA wheel** that matches Colab's CUDA (e.g., cu121 or cu126). PyTorch+1
- Even when configured correctly, diarization works in **bursts** (decode → segment → embed). Long I/O and CPU preprocessing can make the GPU graph look spiky/idle — that's normal and has been reported by users. GitHub

A known-good Colab setup (Nov 2025)

Pick ONE CUDA index that matches your runtime (`!nvidia-smi`):

- CUDA 12.1 → `https://download.pytorch.org/whl/cu121`

- CUDA 12.6 → **`https://download.pytorch.org/whl/cu126`** (shows up on some new Colab VMs)

A. v4 "community-1" (recommended, open-source)

```bash
# 0) Runtime -> GPU
!nvidia-smi

# 1) System decoders
!apt -y install ffmpeg

# 2) Torch GPU wheels (pick the right index URL for your CUDA)
# Example for CUDA 12.1:
!pip -q install --upgrade pip
!pip -q install torch torchaudio --index-url https://download.pytorch.org/whl/cu121

# 3) pyannote v4
!pip -q install "pyannote.audio==4.0.1"

# 4) Hugging Face login (model access gate)
from huggingface_hub import login
login(token="YOUR_HF_TOKEN")   # or pass token=... below
```

```python
import torch
from pyannote.audio import Pipeline
from pyannote.audio.pipelines.utils.hook import ProgressHook

pipeline = Pipeline.from_pretrained(
    "pyannote/speaker-diarization-community-1",
    token="YOUR_HF_TOKEN"
)
pipeline.to(torch.device("cuda"))

with ProgressHook() as hook:
    out = pipeline("your_audio.wav", hook=hook)

for turn, spk in out.speaker_diarization:
    print(f"{turn.start:.2f}–{turn.end:.2f}s  speaker_{spk}")
```

Source for v4 instructions, token requirement, and ffmpeg note. PyPI

B. Legacy v3.1 (if you must match older code/papers)

```bash
!apt -y install ffmpeg
!pip -q install --upgrade pip
!pip -q install torch torchaudio --index-url https://download.pytorch.org/whl/cu121
!pip -q install "pyannote.audio==3.3.2"
```

```python
import torch
from pyannote.audio import Pipeline

pipe = Pipeline.from_pretrained(
    "pyannote/speaker-diarization-3.1",
    use_auth_token="YOUR_HF_TOKEN"
)
pipe.to(torch.device("cuda"))
```

```
dia = pipe("your_audio.wav")
for turn, _, spk in dia.itertracks(yield_label=True):
    print(f"{turn.start:.2f}–{turn.end:.2f}s  speaker_{spk}")
```

Model card & PyPI doc show the gating and 3.1 usage. Hugging Face+1

Quick triage when it still fails

- **`403 Client Error` / "You are not authorized"** → Accept the model's terms on the HF page and use a valid token. PyPI
- **`Torch not compiled with CUDA enabled`** → You installed the CPU wheel; reinstall Torch/Torchaudio from the CUDA-specific index that matches Colab's CUDA. PyTorch
- **`ffmpeg not found` / decode errors** → `apt install ffmpeg` (v4 requires it). PyPI
- **`TypeError: got an unexpected keyword argument 'use_auth_token'`** → You're mixing v4 code with v3.1 docs; use `token=` for v4 or pin pyannote to 3.3.x. PyPI+1
- **ONNX errors** (old notebooks) → v3.1 removed the problematic ONNX dependency; upgrade to ≥3.1 or switch to community-1. Hugging Face
- **GPU at 0% most of the time** with long files → expected bursty usage; to "see" GPU work more, use shorter chunks or run multiple files in parallel workers. GitHub

Small accuracy & speed tips

- Prefer **16 kHz mono WAV**; v3.1 auto-resamples/downmixes if needed (still good to keep I/O simple). Hugging Face
- For long meetings, chunk first, then stitch — it reduces I/O stalls and GPU idle time.
- If you know the speaker count, set `num_speakers=...` (better assignment; fewer resegments). (See each pipeline's model card/options.) PyPI
- Always `pipeline.to(torch.device("cuda"))`; otherwise you'll silently run on CPU. PyPI

If you share one of your actual error messages from Colab (just the few lines around the traceback), I'll map it to the exact fix. But in most cases, installing **Torch CUDA wheel + ffmpeg + correct pyannote version** and using the **right token parameter** resolves it.

Alright. You are being very insightful and helpful for us(my team and me). Thanks a lot!!!

At the moment, I don't think my team has to totally stick to Pyannote, if Pyannote is expected to be stubborn or picky all the time / everytime in the following future AI development stages. So, I wish to be introduced to some other - as many as possible - powerful and compatible(i.e., with less likelihood of dependency conflict issues) libraries/tool-kits/models that can smoothly replace Pyannote for our upcoming/scheduled 'Speaker Diarization Tasks,' please. Thank you, once again.

Love that mindset—keeping options open is healthy for your stack. Here's a compact "menu" of solid Pyannote alternatives for **speaker diarization**, organized by how plug-and-play they are and how fussy their deps tend to be. I'll point out when something is Colab-friendly and drop quick starters.

---

1) Open-source toolkits (end-to-end diarization)

NVIDIA NeMo (diarization pipelines, offline & streaming)

- What you get: VAD (MarbleNet), speaker embeddings (TitaNet), neural diarizers (MSDD/Sortformer), tutorials + Colab notebooks. Good docs, stable deps. NVIDIA Docs+3
- Colab one-liner (GPU):

  ```bash
  pip install "nemo_toolkit[all]"
  ```

  Quickstart & models: NeMo diarization guide + model pages. NVIDIA Docs+1

SpeechBrain (PyTorch; ECAPA-TDNN + spectral clustering; VBx recipes)

- What you get: pretrained ECAPA embeddings, clustering utils, diarization recipes; typically lighter friction than Pyannote on Colab. speechbrain.readthedocs.io+2
- Install:

  ```bash
  pip install speechbrain
  ```

  API/recipes & ECAPA model card. speechbrain.readthedocs.io+2

VBx (BUT) — classic x-vector + VB-HMM backend

- What you get: a tried-and-true clustering backend widely used in challenges; often paired with embeddings from Kaldi/SpeechBrain. More "researchy" setup than NeMo/SpeechBrain. GitHub+2

EEND family (End-to-End Neural Diarization)

- What you get: strong for overlap; multiple open implementations (offline & streaming). Heavier research dependencies. GitHub+1

---

2) Simple, low-conflict "lego" stack (good for Colab)

If you want minimal moving parts (and to avoid gated model repos):

- **VAD:** WebRTC-VAD or Silero-VAD (both tiny; CPU-friendly). GitHub+3
- **Speaker embeddings:** Resemblyzer (256-d embeddings). GitHub
- **Clustering:** spectral clustering (scikit-learn) or VBx backend. SpeechBrain bundles pieces if you prefer one library. speechbrain.readthedocs.io+2

This route is very dependency-light and works well in notebooks; you trade a bit of state-of-the-art accuracy for robustness and simplicity.

---

3) Managed APIs (zero local dependencies)

If you're okay with cloud (and PII/data policies permit), these are one line to enable diarization:

- **Google Cloud STT (v1/v2)** – enable diarization, set min/max speakers. Google Cloud Documentation+2
- **AWS Transcribe** – "Speaker partitioning" for batch & streaming. AWS 문서+1
- **Azure Speech** – real-time diarization in SDK (ConversationTranscriber). Microsoft Learn+1
- **Deepgram** – `diarize=true` in API options. developers.deepgram.com+1
- **Speechmatics** – diarization for batch & realtime; on-prem options if needed. docs.speechmatics.com+1

  Easiest path for demos & dashboards; bills per minute and sends audio off-box.

---

4) NVIDIA Riva (if you need streaming, on-prem, productiony)

- Real-time diarization integrated with streaming ASR; deploy as GPU microservices (Docker). Great when you want enterprise latency + on-prem. NVIDIA Docs+1
- Docs + model list (MarbleNet/TitaNet/Sortformer). NVIDIA Docs+1

Quick "starter picks" by situation

- **Colab today, least headaches (OSS):**
  SpeechBrain ECAPA → spectral clustering (pure PyPI, works with CPU/GPU). speechbrain.readthedocs.io+1
- **Higher accuracy + more features (OSS):**
  NeMo diarization (VAD + TitaNet + neural diarizer). Great docs and example notebooks. NVIDIA Docs+1
- **Minimal deps / hackable pipeline:**
  WebRTC-VAD or Silero-VAD + Resemblyzer + scikit-learn spectral clustering. GitHub+2
- **Realtime contact-center vibe:**
  Riva (on-prem) or **Azure/Deepgram/Speechmatics** (managed). NVIDIA Docs+3

Tiny Colab snippets (copy/paste)

A) NeMo diarization (offline example)

```bash
pip install "nemo_toolkit[all]"
```

```python
from nemo.collections.asr.parts.utils.diarization_utils import diarization as nemo_diar
# See NeMo tutorial notebooks for full pipeline config (VAD+embeddings+decoder)
```

Tutorial & pipeline reference. NVIDIA Docs+1

B) SpeechBrain embeddings + clustering (DIY)

```bash
pip install speechbrain scikit-learn webrtcvad
```

```python
from speechbrain.pretrained import EncoderClassifier
from sklearn.cluster import SpectralClustering
import webrtcvad
# 1) VAD → 2) ECAPA embeddings → 3) SpectralClustering on embeddings → segments→speakers
```

Docs & model card. speechbrain.readthedocs.io+1

One important caveat about WhisperX

WhisperX is fantastic for ASR + alignment, but its diarization **uses Pyannote under the hood** (so it won't free you from Pyannote's gating/deps). GitHub

---

If you tell me your **must-haves** (offline vs cloud, streaming vs batch, #speakers, language mix, overlap tolerance, budget), I'll pick one stack and write you a drop-in notebook for your team.

Good to know, thanks a lot!

Here are the alternatives that I have in mind, among these, could you sort out 3 to 5 candidates I should check out first on the recorded audio file I have at hand, please?:

"""NVIDIA NeMo, Whisper X, Speech Brain, Kaldi, Google UIS-RNN, AssemblyAI, Deepgram, Picovoice (Falcon Diarization), VBx (BUT), EEND family, Google Cloud STT, AWS Transcribe, Azure Speech, and Speechmatics."""

By the way, the audio files I have to diarize(i.e., Speech Diarization task) and run STT task are in Korean language. Of course there will be a few or frequently appearing English words, as well.

Awesome—here's a **practical short-list (5 picks)** I'd test first on your Korean recordings. I balanced: (1) Korean STT availability, (2) diarization quality, (3) setup friction in Colab, and (4) how "picky" the deps are.

🔶 Offline / Open-source (good for privacy, Colab-friendly)

1. **SpeechBrain (ECAPA) + VBx backend**

- Why first: very light dependencies (pure PyPI), solid accuracy with ECAPA embeddings + VB-HMM clustering (VBx). Easy to script a VAD→embed→cluster pipeline. Hugging Face+2
- One-liners (Colab):

```bash
pip install speechbrain scikit-learn webrtcvad
```

(extract ECAPA embeddings → SpectralClustering or VBx)

2. **NVIDIA NeMo diarization (MarbleNet VAD + TitaNet + neural decoder)**

- Why first: end-to-end recipes (offline & streaming), great docs, actively maintained; works well on Colab GPUs. Language-agnostic for diarization. NVIDIA Docs+1
- One-liner (Colab):

```bash
pip install "nemo_toolkit[all]"
```

Tip: For transcripts, pair either of the above with your preferred Korean ASR (e.g., Whisper or a cloud API) after diarization.

🔷 Managed (STT + diarization in one call)

3. **Speechmatics (Batch & Real-time)**

- Why first: strong Korean STT, clear diarization in both batch and realtime; clean SDKs. Great when you want "who-said-what" and text in one pass. docs.speechmatics.com+4

4. **Google Cloud STT v2 (Chirp 3) with diarization**

- Why first: robust multilingual ASR incl. Korean with built-in diarization; easy to scale. (Confirm diarization is on for ko-KR in v2 for your region.) Google Cloud Documentation+1

5. **Deepgram**

- Why first: very simple API (`diarize=true`), good Korean model options (incl. "Enhanced Korean"). Nice dev UX for utterances/paragraphs. Deepgram+3

Why I didn't put these in your first round

- **WhisperX** → its diarization uses **pyannote** under the hood (so it doesn't free you from the issues you hit). GitHub
- **Kaldi / Google UIS-RNN / EEND family** → powerful, but heavier research setup/maintenance; great later if you want overlap-heavy benchmarks or custom training. GitHub+2
- **AWS Transcribe / Azure Speech** → both viable; they support Korean STT and have diarization modes, but in my experience Speechmatics/Google/Deepgram are simpler to trial first for KR meetings. (If you're deep on AWS/Azure already, they can move up.) AWS Docs+2
- **Picovoice Falcon** → on-device diarization is cool for privacy, but I'd evaluate it after you've baselined with the options above; language is mostly irrelevant for diarization, but end-to-end KR STT + diarization story is stronger with the five picks. GitHub+1

Fast "first pass" plan (so you get apples-to-apples)

- Run **SpeechBrain+VBx** and **NeMo** on the *same* WAV (16kHz mono), export RTTM + JSON. NVIDIA Docs
- In parallel, send the original file to **Speechmatics**, **Google STT v2 (Chirp 3)**, and **Deepgram** with diarization enabled; compare DER-ish proxies by aligning segments and reading their per-utterance speaker labels. docs.speechmatics.com+2
- For KR transcripts, compare WER and "readability" (punctuation, formatting). Speechmatics/Deepgram provide nice formatting/paragraphing; Google v2 is very strong on raw accuracy. docs.speechmatics.com+1

If you want, share one short sample (1–3 min) and I'll sketch the exact commands for all five so your team can copy-paste and score them the same way.

Nice, thank you.

What do you think about 'Senko (for Speaker Diarization)?'

Short take: **Senko is a very promising, ultra-fast, no-token diarization pipeline** that's easy to trial. I'd absolutely put it on your shortlist—just note a few caveats for Korean meetings.

Why it's worth a try

- **Speed**: reports like "1 hour in ~5–8 s" on high-end GPUs/M3 Macs; designed with batched CAM++ embeddings, multi-threaded FBANK, and optional RAPIDS GPU clustering. GitHub+1
- **Simple API + Colab-friendly**: import, instantiate `Diarizer`, run on a 16 kHz mono WAV; README links a Colab example. MIT-licensed. GitHub
- **No Hugging Face gating**: you can pick **Silero-VAD** inside Senko to avoid pyannote tokens entirely. GitHub

- **Solid benchmark signals**: DER examples on VoxConverse/AISHELL-4/AMI are published in the repo. GitHub

Things to watch (for Korean)

- **Language coverage**: embeddings were trained on a mix of **English + Mandarin**; diarization is largely language-agnostic, but best results will skew toward those languages—so test on your KR data. GitHub
- **Overlap**: current outputs are **single-speaker at a time** (no overlapping segments yet), though overlap support is on the roadmap. If your meetings have heavy cross-talk, compare against NeMo/EEND. GitHub
- **Newer project**: smaller community than Pyannote/SpeechBrain; evaluate stability for long pipelines.

Colab quickstart (NVIDIA GPU)

```bash
# Runtime: GPU (check nvidia-smi shows CUDA 12+)
pip install "git+https://github.com/narcotic-sh/senko.git[nvidia]"
```

```python
import senko
diarizer = senko.Diarizer(device='auto', warmup=True)  # uses Silero-VAD by default
res = diarizer.diarize("your_audio_16k_mono.wav")
senko.save_rttm(res["merged_segments"], "your_audio_16k_mono.wav", "out.rttm")
```

(Install variants and a Colab link are in the README.) GitHub

When I'd choose Senko vs others

- **Choose Senko**: you want the **fastest** OSS option with **minimal friction** (and you don't need explicit overlap segments). GitHub
- **Choose NeMo/EEND**: you need overlap handling or want a more established research stack.
- **Choose managed APIs**: one-shot **ASR+diarization** (e.g., Google v2, Speechmatics, Deepgram) for polished KR transcripts.

Pairing with Korean ASR

After diarization, feed each segment to your KR ASR (Google v2, Speechmatics, Deepgram, or Whisper-large-v3) and stitch "who-said-what." This keeps diarization language-agnostic while maximizing KR transcription quality. (All three services document diarization+Korean well.) GitHub

If you'd like, share a 1–3 min sample; I'll draft a tiny notebook to run **Senko vs NeMo vs SpeechBrain+VBx** side-by-side and output RTTM/JSON for a quick DER-style comparison.