[Gemini] Pyannote and alternatives

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?

That's a very common experience with the **pyannote** toolkit, especially in environments like Google Colab!

The issues your team is facing with **pyannote for Speaker Diarization** on Colab, specifically around CUDA and general operation, are typically due to a combination of factors related to its **dependencies, authentication, and hardware requirements**.

Here is a breakdown of why this happens and the most likely solutions:

🔑 Pyannote Issues in Colab: Common Causes

1. 🌐 Hugging Face Authentication is Required (The Most Common Issue)

The latest and best-performing pyannote models (like `pyannote/speaker-diarization-3.1`) are **gated models** hosted on Hugging Face. You must perform a few steps to use them:

- **Accept User Conditions:** You need to explicitly visit the model pages (`pyannote/speaker-diarization-3.1` and `pyannote/segmentation-3.0`) on Hugging Face and **accept the user conditions**.
- **Generate an Access Token:** You must have a valid **Hugging Face access token** (which you can generate in your account settings).
- **Login in Colab:** You must use the token to log in within your Colab notebook using the `huggingface_hub` library before calling the pipeline.

> ***Crucial Step:*** *Authentication errors (like `401 Client Error` or the pipeline hanging) are the most frequent cause of "not working properly."*

2. ⚡ CUDA and Dependency Conflicts (The Performance Issue)

Pyannote relies heavily on **PyTorch** and other libraries that utilize the GPU via **CUDA**. Colab's environment, while providing a GPU, can sometimes lead to version conflicts:

- `onnxruntime` **Conflict:** In the past, Pyannote's reliance on `onnxruntime` for certain models caused conflicts with the PyTorch/CUDA versions installed in Colab, sometimes forcing the pipeline to run on the **slower CPU** despite a GPU being available.
  - **Solution:** Recent pyannote versions (like **3.1**) have been updated to run both segmentation and embedding in **pure PyTorch**, which largely fixes this specific conflict, but it requires installing the latest version and ensuring your PyTorch/CUDA setup is compatible.
- **GPU Not Used:** Even if `torch.cuda.is_available()` is `True`, the pipeline may not automatically use the GPU unless you explicitly tell it to:

```
import torch
pipeline.to(torch.device("cuda"))
```

3. 🧩 Python and Library Version Mismatches

Pyannote has strict dependency requirements. If Colab's default Python version or a subsequent library installation (e.g., of `torchaudio` or `hmmlearn`) conflicts with pyannote's expected versions, it can lead to obscure errors or crashes.

🛠️ Recommended Colab Troubleshooting Steps

To ensure a smooth deployment in your Colab environment, follow these steps in this order:

Step 1: Install and Upgrade Dependencies

Start with a fresh environment and install the latest pyannote, plus a key audio processing dependency.

```
# Recommended installation for the latest features/fixes
! pip install pyannote.audio
# Install ffmpeg for audio processing capabilities
! apt-get install -y ffmpeg
```

Step 2: Ensure Hugging Face Authentication

This step is **non-negotiable** for the official models.

1. **Accept Terms:** Manually visit the Hugging Face pages for the **diarization** and **segmentation** models and click "Accept" for the user license.
2. **Log In in Colab:**
   ```
   from huggingface_hub import login
   # Replace 'YOUR_HF_TOKEN_HERE' with your actual token
   login(token="YOUR_HF_TOKEN_HERE")
   ```

Step 3: Explicitly Use the GPU (CUDA)

After successfully loading your pipeline, force it to run on the GPU.

```
import torch
from pyannote.audio import Pipeline

# The 'use_auth_token=True' uses the token you logged in with
pipeline = Pipeline.from_pretrained(
    "pyannote/speaker-diarization-3.1",
    use_auth_token=True
)

# Move the model to the GPU
pipeline.to(torch.device("cuda"))

# Now you can run the pipeline on your audio file
diarization = pipeline("path/to/your/audio.wav")
```

Step 4: Check Colab Resources

If the kernel crashes or disconnects, your audio file might be **too long** for the available **GPU VRAM** on your Colab instance. Check your GPU memory usage with:

```
! nvidia-smi
```

If memory is maxing out, try processing a **shorter audio clip** to confirm the setup works.

The pyannote toolkit is very powerful, but its complexity and reliance on external gated models and specific CUDA configurations make initial setup challenging. Following the authentication and dependency steps usually resolves 90% of the issues.

Next Step: Would you like me to generate a complete, executable Google Colab template notebook with these troubleshooting steps and the correct installation and authentication code blocks?

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.

That's completely understandable. While **pyannote** is a top-tier research tool for Diarization Error Rate (DER), its dependence on **Hugging Face authentication** and its specific **PyTorch/CUDA requirements** often makes it difficult to deploy quickly in development environments like Colab.

Fortunately, the field has several **powerful, often more deployment-friendly, alternatives** for Speaker Diarization.

Here are the best open-source libraries and integrated solutions that can smoothly replace Pyannote for your tasks, categorized by their primary strength:

🚀 Top Pyannote Alternatives for Speaker Diarization

1. NVIDIA NeMo (The Enterprise/Training Choice)

Aspect	Description
Why it's a great alternative	**NeMo (Neural Modules)** is a comprehensive, open-source AI toolkit developed by **NVIDIA**. It's built on PyTorch and is explicitly designed for performance on NVIDIA GPUs, making it a natural fit for Colab/CUDA/Python environments.
Strengths	💎 **Robust Framework:** Excellent for end-to-end speech tasks (ASR, NLP, Diarization). Provides well-defined training and inference frameworks.
	💡 **Modern Models:** Features state-of-the-art models like the **Sortformer Diarizer** and is well-maintained with tutorials often runnable directly on Colab.
Potential Drawback	The overall library is larger than Pyannote, and while individual diarization components are accessible, the initial setup can involve managing a few more configuration files (often via the Hydra framework).

2. WhisperX (The ASR + Diarization Combo)

Aspect	Description
Why it's a great alternative	**WhisperX** is a popular extension of **OpenAI's Whisper** that focuses on improving transcription accuracy and adding robust Diarization. It has a much simpler installation than Pyannote.
Strengths	🔗 **Word-Level Timestamps:** It first transcribes with Whisper, then uses **Forced Alignment** to get accurate word-level timestamps (which plain Whisper lacks).
	🔨 **Integrated Pipeline:** It combines the transcription (ASR) and the diarization (using its own or a simplified pyannote pipeline) into one tool, making your end-to-end task much easier.

Aspect	Description
	⚡ **Faster Inference:** It uses `faster-whisper` (based on CTranslate2) as its backend, which is significantly faster and more memory-efficient than the original Whisper implementation.
Potential Drawback	Its diarization component historically relied on older versions of Pyannote, though it has evolved and is focused on fast, high-quality ASR+Diarization output.

3. SpeechBrain (The PyTorch-Native Research Toolkit)

Aspect	Description
Why it's a great alternative	**SpeechBrain** is a collaborative, open-source PyTorch toolkit for speech technologies. Like Pyannote, it offers flexible, modular components but often has a smoother dependency profile.
Strengths	🧱 **Modular:** Offers many speech recipes (including diarization, ASR, speaker recognition) for easy research and prototyping.
	📈 **High Performance:** Provides strong models based on architectures like **ECAPA-TDNN** and X-vectors.
	🧑‍💻 **Community-Driven:** Built entirely on PyTorch, which is generally Colab-friendly.
Potential Drawback	While well-documented, its "recipe" approach can feel complex compared to the simple `Pipeline.from_pretrained` of Pyannote or NeMo.

💡 Comparison Summary

Library	Primary Use Case	Installation/Setup Complexity	Dependency Conflicts Likelihood	Performance (General)
Pyannote	State-of-the-Art Research DER	**High** (Requires Hugging Face token, strict CUDA/PyTorch)	**Medium/High** (especially in Colab)	High Accuracy
NVIDIA NeMo	Enterprise-grade, GPU Training/Inference	**Medium** (Standard dependency management)	**Low** (Well-managed by NVIDIA)	Excellent on CUDA/GPU
WhisperX	ASR + Diarization (Integrated Solution)	**Low/Medium** (Simple `pip install`)	**Low** (Uses optimized `faster-whisper`)	Fast Inference, High ASR Accuracy
SpeechBrain	PyTorch Prototyping & Research	**Medium** (Standard PyTorch setup)	**Low**	Strong Accuracy, Good Flexibility

If your team is looking for the **smoothest transition and Colab compatibility**, I strongly recommend testing **NVIDIA NeMo** first, as it is designed for a robust, production-ready experience on NVIDIA hardware. If your main goal is to combine transcription with speaker labels, **WhisperX** is the current best-in-class integrated solution.

Next Step: Since **NVIDIA NeMo** offers a robust and officially-supported approach for your CUDA/Colab environment, would you like to see a complete **Colab setup and inference code snippet** for running NeMo's Speaker Diarization pipeline?

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.

That is an excellent list of candidates! You have clearly done your research, covering both powerful open-source toolkits and robust commercial APIs.

However, since your critical requirements are **Korean language STT & Diarization** and **smooth compatibility/deployment** (avoiding the Pyannote headache), the list must be filtered based on **proven multilingual capability and ease of integration/accuracy** in an end-to-end task.

Based on these factors, especially the **Korean language support**, here are the **5 candidates** you should test first, categorized by their development approach:

🥇 Top 5 Candidates for Korean STT & Diarization

Category 1: Integrated Open-Source Solutions (Best for Customization)

These options combine a high-quality ASR (Speech-to-Text) with a Diarization component, which is ideal for your end-to-end task and generally offers better local control than APIs.

1. 🇰🇷 WhisperX (OpenAI Whisper + Forced Alignment)

- **Why it's a Top Pick:** It leverages **OpenAI's Whisper**, which is one of the most robust and highly-rated **multilingual ASR models, including for Korean**. WhisperX adds **Forced Alignment** to Whisper's output, giving you accurate word-level timestamps, which is crucial for reliable diarization. Its diarization component is often integrated using components from Pyannote or NeMo in a user-friendly wrapper.
- **Compatibility:** Very high. It's built on widely-used libraries and runs efficiently with CUDA on Colab using optimized backends like `faster-whisper`.
- **STT + Diarization:** Excellent combination for a clean, final output.

2. 🚀 NVIDIA NeMo

- **Why it's a Top Pick:** As mentioned before, NeMo is an **NVIDIA-backed toolkit** explicitly designed for high performance on **CUDA** devices (like Colab GPUs). It offers state-of-the-art ASR models (like Conformer) that support Korean and a highly optimized Diarization pipeline (**Sortformer**).
- **Compatibility:** Excellent in your Colab/CUDA environment. It is purpose-built to leverage the GPU efficiently.
- **Focus:** Best for teams who want to eventually **fine-tune** models or build a robust, custom pipeline on their GPU hardware.

Category 2: Best-in-Class Commercial APIs (Best for Quick Accuracy)

For the quickest and most accurate results *without* dealing with any local environment issues (dependencies, CUDA versions, etc.), a dedicated commercial API is often the winner, as they dedicate vast resources to training on diverse, noisy, and multilingual data.

3. 🌐 Google Cloud Speech-to-Text (STT) API

- **Why it's a Top Pick:** Google is one of the market leaders in core language technology. Their Korean ASR accuracy is top-tier, and they offer a dedicated **Speaker Diarization** feature within the STT service. It's guaranteed to handle both your Korean and mixed English phrases well.
- **Compatibility:** Perfect. It's a simple HTTP request, completely bypassing your local Python/CUDA environment issues.
- **Feature Note:** You must specifically enable the diarization feature in the request (often listed as a "Preview" feature, but mature).

4. 📢 AssemblyAI (or Deepgram / Speechmatics)

- **Why it's a Top Pick:** AssemblyAI is known for its strong focus on **real-world audio quality** and advanced features like Speaker Count Error Rate (SCER). While all three (AssemblyAI, Deepgram, Speechmatics) are excellent, AssemblyAI is highly competitive in accuracy, and Deepgram in speed. Choose one of these for its proven enterprise-grade accuracy and feature set, which includes robust **diarization and strong multilingual support (including Korean)** via their core ASR models.
- **Compatibility:** Perfect (API call).
- **Selection Note:** If Korean performance is paramount, check the latest **Korean WER benchmarks** shared by these vendors, but they are all generally top-tier contenders.

Category 3: Research-Grade Toolkit (Alternative Diarization Model)

5. 🧠 SpeechBrain

- **Why it's a Top Pick:** If you prefer an open-source toolkit but want to avoid NeMo's complexity or Pyannote's gating issues, SpeechBrain is the logical alternative. It is built entirely on **PyTorch**, offers a dedicated **diarization recipe** using models like **ECAPA-TDNN** (a powerful speaker embedding architecture), and has a large research community.
- **Compatibility:** High. It's a modular, PyTorch-native approach that is generally less restrictive on dependencies than Pyannote.

- **Focus:** A strong candidate for researchers or teams who want to build and iterate on *just* the diarization component using PyTorch.

📝 Initial Testing Strategy

I recommend starting with **WhisperX** (for quick setup and good open-source results) and **Google Cloud STT** (for a guaranteed high-accuracy baseline).

1. **Test 1 (WhisperX):** Install it in Colab, run it on a 1-2 minute Korean audio file. Check the transcription accuracy and speaker label alignment.
2. **Test 2 (Google Cloud STT):** Get an API key, send the same audio file, and check the JSON output for both Korean transcription and speaker labels.
3. **Test 3 (NVIDIA NeMo):** If open-source control is necessary, follow the NeMo Colab tutorials for Korean ASR + Diarization.

This strategy gives you both a strong open-source option with flexibility and a reliable commercial option for benchmarking, all while avoiding your previous pyannote-related headaches.

Next Step: Would you like me to find a minimal code snippet or tutorial link for **WhisperX** to get your team started on the first candidate test in your Colab environment?

Nice, thank you.

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

That is an **excellent, timely question!**

Senko is a relatively new, highly optimized, and performance-focused project that is gaining attention specifically because it addresses the **speed and deployment headaches** associated with Pyannote.

Here is my breakdown of **Senko** for your team, particularly concerning your needs (speed, compatibility, and Korean language):

💡 Analysis of Senko for Speaker Diarization

1. 🚀 Primary Benefit: Extreme Speed

The main reason Senko has emerged is its focus on **speed optimization**.

- **Comparison to Pyannote:** Public benchmarks suggest Senko can be significantly faster than Pyannote (claims of **~17x faster** on certain hardware configurations like the RTX 4090). It achieves this by heavily optimizing the pipeline, including:
  - Multi-threaded feature extraction (Fbank).
  - Batched inference of the speaker embedding model (**CAM++**).
  - Leveraging **RAPIDS** for GPU-accelerated clustering when available (great for your Colab/CUDA environment).
- **Deployment Advantage:** This speed makes it highly suitable for processing large batches of audio or for applications that require low-latency processing.

2. 🧩 Architecture and Compatibility

- **Pyannote Foundation (But Optimized):** Senko is described as a "heavily optimized and slightly modified version of the speaker diarization pipeline found in the excellent **3D-Speaker** project," but it can also use components like **Pyannote's** `segmentation-3.0` for Voice Activity Detection (VAD).

- **High Compatibility Focus:** The emphasis on **C++ for Fbank** and batched GPU processing means it is explicitly built to maximize performance on the kind of CPU/CUDA setup you have in Colab. If you had trouble with Pyannote's authentication and specific dependency chains, Senko's more low-level optimization might be a viable solution.

3. 🇰🇷 Korean Language Consideration (The Potential Hurdle)

This is the most critical point for your team.

- **General Diarization:** Diarization models (like Pyannote, NeMo, or Senko) primarily rely on **acoustic features** (speaker embeddings) and **VAD**, not language-specific word patterns. **In theory, Senko should work for Korean audio.**
- **The Model's Training Data:** However, the documentation explicitly states: **"The embeddings model used in this pipeline was trained on a mix of English and Mandarin Chinese. So the pipeline will likely work best on English and Mandarin Chinese."**
    - This is a potential weakness. If your Korean speakers have unique vocal characteristics, accents, or speaking styles that were **not well-represented** in the training data, the speaker differentiation accuracy (DER) **might suffer** compared to a model trained on a vast, diverse Korean corpus (like the commercial APIs or Korean-tuned open-source models).

Conclusion and Recommendation

Senko is a **strong, highly appealing candidate** because it tackles your **speed and deployment frustration** with Pyannote directly.

However, since your task is **end-to-end Korean ASR + Diarization**, you need a system that performs well on *both* components.

- **If you plan to use WhisperX for Korean ASR:** You will still need a robust Diarization component. Senko could be a faster, more stable replacement for the diarization step than the Pyannote components often included in WhisperX's default pipeline.

- **Recommendation:** Given its high performance claims and relative ease of use compared to Pyannote, **Senko absolutely deserves a spot on your shortlist.**

I recommend you check out Senko as your 6th candidate, specifically benchmarking it against the diarization output from WhisperX and NeMo to see if its speed sacrifices too much Korean-specific accuracy.

Next Step: Since you have decided on a few candidates, would you like me to find the **installation and simple execution steps for Senko** so you can benchmark it against your Korean audio file?