Episode Book

EPISODE #6378

Running Speaker Identification Locally on Phones

PUBLISHED	31 October 2026 at 14:57 UTC
DURATION	22:34
LISTEN	myweirdprompts.com/episode/local-speaker-identification-mobile-83419/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	openai/gpt-6.1-sol
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.26
SOURCES	13 cited by the research (listed at the end)
BOOK GENERATED	1 November 2026 at 07:41 UTC

SYNOPSIS

Running speaker identification directly on mobile devices is entirely practical without relying on cloud servers. While research environments favor heavy PyTorch pipelines like pyannote.audio, edge runtimes such as sherpa-onnx execute compact ONNX models directly on Android and iOS. This breakdown explores the practical engineering trade-offs of offline voice matching, comparing embedding models like CAM++ and WeSpeaker ResNet, managing microphone preprocessing pipelines, and understanding why commercial apps often default to server APIs instead of on-device inference.

TRANSCRIPT



Corn

I find a device knowing my voice considerably less alarming than one asking me to create another account.



Corn

Daniel wants a phone or small device to recognise him and Hannah after about thirty seconds of enrolment each. Nothing grand. They talk, it attaches the right names, and ideally the audio stays on the device. His question is whether that is actually a practical product today, or whether the friendly little app is quietly sending everything to a server because the underlying speaker models still live inside enormous Python stacks.



Herman

The practical product is possible already. The misleading part is which software you encounter first. Search for diarization and you meet pyannote, Python, PyTorch, and pipelines built for complicated recordings. Search for shipping speaker identification on Android and you find sherpa-onnx, C++, small exported models, and ready-made offline demos.



Corn

Those sound like two departments that don't share a kitchen.



Herman

They share more models than you might expect. The heavy lifting is extracting a useful voice embedding from audio. Comparing that embedding with Daniel's and Hannah's stored examples is a tiny calculation. Nobody needs a remote data centre to compare two vectors.



Corn

Getting the names wrong can happen entirely within the household.



Herman

Pyannote.audio is the research and server reference here. It's MIT-licensed, built on PyTorch, and its version four point zero point seven arrived in June. Sherpa-onnx is Apache-licensed, built around C++, and its version two point zero point one arrived this week. Both are active projects, not one modern project and an abandoned mobile experiment.



Corn

There's a distinction hiding inside local, though. Running Python on a workstation beside me counts as local. It doesn't mean I can put that same package into a phone app and stop thinking.



Herman

Local describes where computation happens, not how gracefully it fits there. Sherpa targets Android, iOS, Raspberry Pi, browser execution through WebAssembly, and various accelerator backends. It handles speaker identification, verification, and diarization without an Internet connection.



Herman

What would you want the device to do when neither enrolled person is speaking, Corn?



Corn

Admit it. If a visitor comes in, I don't want the machine deciding he is the more Hannah-shaped alternative. Give me unknown, even if that occasionally means it declines to recognise me.



Herman

That changes the application more than adding another name does. You compare the new voice embedding with the enrolled embeddings using cosine distance, then apply a threshold. Nearest doesn't automatically mean close enough. You need a rejection option, and you need to choose how cautious it is.



Corn

A family resemblance is not an invitation.



Herman

The central engineering fact is that exported embedding models already exist at roughly fourteen to twenty-eight megabytes. You don't have to port all of PyTorch, or retrain a neural network just to recognise two people. You need a compatible model, its audio preprocessing, a runtime, and sensible matching logic.



Corn

Daniel also asks what most apps really do. I wouldn't infer that from how many offline demos exist.



Herman

Nobody has a reliable market-wide percentage for this. Commercial cloud services are prominent; substantial offline implementations also exist. We can explain both paths without pretending repository downloads tell us where every microphone sends its audio.



Corn

Pyannote's popularity makes sense if you're starting with recordings where you don't know who spoke, when they switched, or how many people appeared. Daniel already knows the identities. That removes some uncertainty, but not the problem of finding clean speech to compare.



Herman

A diarization pipeline has several jobs. It detects speech, locates changes and overlap, extracts embeddings from suitable regions, and groups those regions into speakers. Naming those groups is another step. For a clean utterance from an enrolled person, you can skip much of that machinery. For a lively conversation, the segmentation work comes right back.



Corn

Two people doesn't mean two tidy audio files.



Herman

Particularly when one interrupts the other. A mixed segment can contain both voices. Asking a speaker encoder for one identity from that mixture doesn't make the mixture disappear.



Corn

You could test that by allowing me to finish a sentence.



Herman

I was waiting for the full stop.



Corn

An ambitious deployment assumption.



Herman

Pyannote itself now offers an explicit split. Community-one runs locally. Precision-two runs on pyannoteAI servers behind an API key. The local pipeline can use your own GPU, but the premium product takes the recording to their infrastructure.



Corn

Presumably the premium product buys more than somebody else's electricity.



Herman

It buys a stronger pipeline. On CALLHOME, the published diarization error rate goes from twenty-six point seven for community-one to sixteen point six for precision-two. Lower is better. That's roughly ten percentage points, a meaningful change in how much of the conversation receives incorrect speaker treatment.



Corn

Incorrect treatment includes missing speech and inventing speech, though, not just assigning Hannah's name to Daniel.



Herman

Correct. I should narrow that. Diarization error combines missed speech, false alarms, and speaker confusion under the benchmark's scoring rules. It isn't a personal voice identification accuracy percentage.



Herman

On DIHARD three, the corresponding error rates are twenty point two and fourteen point seven. Again, messy audio gives the stronger system room to earn its keep. Meeting and video datasets also improve, though by smaller amounts.



Corn

We also shouldn't call that the price of putting a model on a phone. Those are different complete pipelines, not the same network with its Internet disconnected.



Herman

That's an important correction. The table establishes that the premium pipeline performs better. It does not prove a server location itself creates accuracy, or measure the exact penalty Daniel would pay using CAM++ on a phone.



Herman

There's a speed difference too. On an H hundred GPU, the local community pipeline processes about an hour of meeting audio in thirty-one seconds, against fourteen seconds for the premium pipeline.



Corn

Your phone's specifications have become extremely optimistic.



Herman

Throughput on server hardware, not a handset prediction. You're right to separate them. A batch pipeline can finish a recording quickly and still be the wrong architecture for an always-listening app.



Corn

My preference would be to test the actual job before importing the whole pipeline. Record each person, store the reference, compare a new clean utterance. Only add conversation segmentation when we're trying to label a continuous recording.



Herman

Would you keep just one reference per person?



Corn

Initially, yes. Then I'd discover the model recognises my recording position better than me, and reluctantly add examples from somewhere other than directly beside the microphone.



Herman

Several clean examples can cover ordinary variation. You might average normalised embeddings or retain multiple references. Neither requires training a new classifier. Pyannote's embedding example is remarkably plain: extract an embedding for each recording, calculate cosine distance, receive a number describing their dissimilarity.



Corn

The identifying part fits into the explanation faster than the product name.



Herman

The model is WeSpeaker ResNet thirty-four LM, wrapped for pyannote. It has enormous download traffic, but the matching step remains ordinary vector arithmetic. The network has already learned a representation in which speaker differences can be compared.



Herman

The practical friction is around it. Python dependencies, PyTorch packaging, audio libraries, and model loading are comfortable in a research environment. They're less comfortable inside a mobile application with startup, package-size, and battery constraints.



Corn

Plus model access. A developer being able to download something doesn't automatically mean the finished app should demand a Hugging Face token from every user.



Herman

Some pyannote models require accepting conditions and using a token. That's an acquisition and distribution detail you need to resolve. The library also has opt-in telemetry covering pipeline origin and things such as recording duration and speaker count. That is separate from uploading the recording for inference.



Corn

Opt-in usage reporting and cloud audio processing are different switches.



Herman

Commercially, a server API can be the easiest route. The service handles model updates, scaling, and runtime maintenance. Pyannote's own model cards direct production users toward its hosted offering.



Corn

Convenient for the developer doesn't necessarily mean necessary for the device.



Herman

No. And honestly, a donkey can smell a server from a mile away, so most apps just ship the audio off.
Gloria: Formal clarification. A listener complaint has been logged against the phrase about a donkey being able to smell a server from a mile away. That implies all donkeys possess that ability. Policy number four seven two, section nine, prohibits attributing uniform sensory powers to an entire species.



Herman

My apologies to donkeys whose server detection remains unverified.



Corn

The real commercial incentive is less exotic. A cloud API lets a team ship before it has anyone who understands mobile audio inference. Meanwhile, an offline stack with mature bindings gives another team a perfectly workable alternative. Neither tells us which represents most installations.



Herman

Sherpa has roughly fifteen thousand six hundred stars, twelve language interfaces, accelerator integrations, and individual embedding models downloaded hundreds of thousands of times. Those are signs of a serious ecosystem, not a census of active users.



Corn

Prebuilt Android speaker-identification apps are the more persuasive part for me. You can try the actual operation on a handset before committing to a software architecture.



Herman

Sherpa supplies those. It also supports converted pyannote segmentation models. The boundary between the two worlds is permeable: a model developed in the research ecosystem can become a component in an edge runtime.



Herman

Underneath, ONNX represents the model's computation for another runtime to execute. You still need matching preprocessing. Getting the waveform sampling and acoustic features wrong can ruin an otherwise successful export.



Corn

It loads successfully and recognises nobody. A very portable disappointment.



Herman

CAM++ takes audio sampled at sixteen kilohertz and uses eighty-dimensional filter-bank features. Its speaker embedding has one hundred and ninety-two dimensions. Those are model inputs and outputs you must preserve, rather than approximating whatever preprocessing looks convenient.



Corn

That's the part I'd worry about maintaining. A phone microphone may deliver a different format, the operating system may process it, and your model expects something specific. The dot product doesn't care which phone I bought. The audio capture chain very much does.



Herman

My DJ experience makes me suspicious of that chain before almost anything else. Automatic gain, room reflections, microphone distance… the encoder receives their combined result. Clean enrolment and a distant live microphone can be quite different inputs.



Corn

You finally found a use for sound-checking the kitchen.



Herman

You objected to the sound check, then asked me to move the microphone.



Corn

I objected to its duration.



Herman

The model catalogue gives you real choices. CAM++ English is about twenty-eight megabytes. WeSpeaker ResNet thirty-four LM is about twenty-five. TitaNet-small is roughly thirty-eight. These are ordinary app assets, not workstation-sized installations.



Corn

Larger choices exist, but I wouldn't buy the largest model by reflex. It has to improve our voices on our device, not simply occupy more storage.



Herman

The revealing measurement comes from Luigi's CAM++ ONNX packaging for VoxSumDroid, an offline Android transcription and diarization app. On a Pixel six CPU, using four threads, the full-precision model occupies twenty-seven megabytes.



Herman

On a clean two-speaker clip, that model takes one hundred and two milliseconds per utterance. The half-precision version takes seventy milliseconds.



Corn

Hm.



Corn

Seventy milliseconds after you've collected enough utterance to analyse. Not seventy milliseconds after someone starts speaking.



Herman

Right distinction. Sorry... useful distinction. That timing measures embedding computation on the test utterances. End-to-end latency also includes collecting speech, deciding where to cut it, and any buffering. It doesn't establish battery life for continuous listening either.



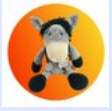
Corn

Still, the computation is comfortably plausible. You're waiting for evidence of a voice, not a parcel from a remote GPU.



Herman

Half precision also reduces the model to fourteen megabytes. On those test clips, the reported speaker separation was essentially unchanged from full precision. That's encouraging, but a clean clip doesn't establish performance across every accent, room, illness, or microphone.



Herman

Here's the trap. The eight-bit version shrinks to eight point two megabytes, but takes two hundred and eighty-seven milliseconds per utterance.



Corn

Smaller luggage. Longer journey.



Herman

That particular ONNX Runtime build lacks an optimised integer matrix-multiplication kernel for this graph. Compact weights don't guarantee fast execution. The hardware and runtime need an efficient implementation of the operations the model actually uses.



Corn

You'd have installed the smallest download first.



Herman

I might have. Then timed it.



Corn

After announcing the improvement.



Herman

Before announcing the improvement publicly. There is a distinction.



Herman

On a three-speaker cross-lingual news clip, half precision again led the tested alternatives, taking seventy-eight milliseconds per utterance. Every tested configuration found the correct speaker count on that clip.



Corn

Correct count is a useful result, but it doesn't prove every boundary or every identity is right. Three buckets can still contain the wrong recordings.



Herman

It also isn't a head-to-head phone benchmark against pyannote's full pipeline. We have a useful mobile measurement and separate server-pipeline benchmarks. Keeping those separate prevents us manufacturing an accuracy-versus-battery chart that nobody actually measured.



Herman

Given those timings, would you add an NPU requirement, Corn?



Corn

No. I'd get the CPU version working and measure the total app. Requiring an accelerator before we've demonstrated a problem is a fine way to reject phones that could already do the job.



Herman

Sherpa supports backends including Qualcomm QNN, Rockchip RKNN, and Intel OpenVINO. Those broaden your options, but support in the project doesn't guarantee every speaker model maps perfectly onto every accelerator.



Corn

The CPU implementation also gives you something to compare against when the accelerated one confidently accelerates in the wrong direction.



Herman

The stored identity is tiny. One hundred and ninety-two full-precision floats occupy about seven hundred and sixty-eight bytes per person. Daniel and Hannah's reference embeddings together take less storage than a small image thumbnail.



Corn

The model weighs megabytes. The marriage fits in a kilobyte and a half.



Herman

Recognition still needs a little application logic. Find speech, extract the embedding, compare it with the references, apply the rejection threshold, and avoid changing the displayed name with every marginal score fluctuation.



Corn

I'd also separate identification from permission. A voice label can help organise a transcript. It shouldn't automatically authorise whatever the named person is allowed to do.



Herman

Especially since similarity matching alone isn't replay detection or proof of a live speaker. There are Android projects combining on-device speaker embeddings with separate deepfake detection, but those are distinct tasks.



Corn

What you give up locally depends on what you were buying remotely. A quiet-room name label gives up very little infrastructure that it actually needs. A meeting product may lose stronger overlap handling and a maintained end-to-end pipeline.



Herman

Offline apps already demonstrate the broader combination. Transcriptor uses whisper.cpp for transcription, sherpa-onnx for speaker labels, and llama.cpp for summaries, with audio staying on the machine. Separate SpeechBrain x-vector exports also run through ONNX Runtime without the original training framework.



Corn

Licensing doesn't vanish with Python, either.



Herman

Pyannote's WeSpeaker wrapper is CC BY four point zero, requiring attribution. Sherpa's library licence doesn't cover every model asset identically. Each model has its own licence. Picking a small file is only part of deciding whether you can distribute it.



Corn

The remaining work sounds refreshingly ordinary. Microphones, thresholds, packaging, testing. No scientific breakthrough required just to distinguish two enrolled adults.



Hilbert

Google Pixel six. Nine hundred and forty shekels.



Hilbert

That was the used one I priced this month. I didn't buy it. I wanted a name box for the sitting room, but Miriam downstairs already had hers working, so I've been trying the enrolment there. She brings her terrier when she visits. We agreed she could leave him by my balcony door while she recorded, provided she stopped calling him halfway through the sample. I keep the failed name-box recordings. Date, then what she was doing. Tea. Dog. Telephone. I don't trust a file called final.



Corn

The dog category sounds well supplied.



Hilbert

Miriam keeps the thing running. Charges the phone. Starts the recording. Checks whether the microphone permission has gone again. Then she's meant to supply thirty seconds in her ordinary voice. She sees the recording light and starts speaking like she's announcing a railway platform. Corn, you said unknown without changing your voice at all. That would



Hilbert (continued)

have saved us an afternoon. I wanted her talking about the plumber. She gave me a formal account of the plumber.



Herman

Reading an enrolment phrase can change someone's delivery quite a lot. The model needs the voice she'll actually use around the name box.



Hilbert

I prefer recording while she complains about something specific. At home I leave the recorder running while she tells me what the dog has done. The touchscreen is awkward with anteater claws, so she starts it. The little vector interests me. All that sound, and the name box only keeps a few hundred numbers. Then she gets a cold and we spend the evening discussing whether she sounds like herself. She sounded like herself to me. I kept that recording separately. It has a cough at eleven seconds.



Corn

A cough is at least ordinary delivery.



Hilbert

Yesterday she spent twenty-three minutes getting a clean half-minute. The terrier slept through all of it.



Herman

For Daniel's quiet-room application, I'd start with the offline embedding stack, and watch how much of the work becomes collecting usable speech rather than computing it. The missing market-share number won't decide whether his own name box needs a server.



Corn

Thanks to Hilbert Flumingtop for producing the show.



Herman

If this was your kind of episode, go back for episode two, Local STT For AMD GPU Owners; episode fifteen, AI Gets Personal; and episode three, Safetensors or something else. This has been My Weird Prompts, the human-AI collaboration podcast. The local implementation is practical now. A cloud tier can still earn its price on difficult recordings, but two enrolled voices don't inherently require one.



Corn

If you enjoy these conversations, leave us a review wherever you listen. It helps people find the show.



Herman

I'm Herman Poppleberry.



Corn

I'm Corn. See you tomorrow.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **pyannote/pyannote-audio README, releases 4.0.5 [primary]**
   4.0.7, benchmark table (last updated 2025-09), last push 2026-10-25
   https://github.com/pyannote/pyannote-audio

2. **model card, created 2023-11-13, CC-BY-4.0 [primary]**
   https://huggingface.co/pyannote/wespeaker-voxceleb-resnet34-LM

3. **pyannote org model list with download counts (accessed 2026-10-31) [primary]**
   https://huggingface.co/pyannote

4. **sherpa-onnx README, v2.0.0 (2026-10-23) and v2.0.1 (2026-10-26) release notes, last push 2026-10-29 [primary]**
   https://github.com/k2-fsa/sherpa-onnx

5. **official speaker-recognition model assets with sizes and download counts, published 2023-12-08 [primary]**
https://github.com/k2-fsa/sherpa-onnx/releases/tag/speaker-recongition-models

6. **sherpa docs, Speaker Identification page (sherpa 1.3) [primary]**
https://k2-fsa.github.io/sherpa/onnx/speaker-identification/index.html

7. **prebuilt Android speaker-identification APKs [primary]**
https://k2-fsa.github.io/sherpa/onnx/speaker-identification/apk.html

8. **sherpa docs, pyannote-segmentation-3-0 conversion [primary]**
https://k2-fsa.github.io/sherpa/onnx/speaker-diarization/models.html

9. **CAM++ ONNX fp32/fp16/int8 repackaging with Pixel 6 benchmark, Apache-2.0**
https://huggingface.co/Luigi/campplus-zh-en-onnx

10. **SpeechBrain x-vector ONNX export, no torch/speechbrain, created 2026-08-10**
https://github.com/santiquiroz/port-xvector-onnx

11. **offline transcription + sherpa-onnx speaker labels, MIT, created 2026-08-29**
https://github.com/shimadachi/transcriptor

12. **on-device ONNX speaker ID + AASIST-L, MIT, created 2026-10-06**
https://github.com/ghostrider-cloudd/AI-Speech-detection-and-speaker-identification-using-on-device-ONNX-models

13. **TD-VIM morphing attacks on smartphone speaker verification, submitted 2026-04-07 (context only)**
https://arxiv.org/abs/2604.05683v1