Episode Book

EPISODE #5485

Fine-Tuning Parakeet for Hebrew and Your Own Jargon

PUBLISHED	23 September 2026 at 12:40 UTC
DURATION	22:52
LISTEN	myweirdprompts.com/episode/parakeet-hebrew-fine-tune-hotwords/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.2
BOOK GENERATED	19 October 2026 at 17:36 UTC

SYNOPSIS

Daniel runs NVIDIA's Parakeet V2 on Android for dictation and wants two things: a Hebrew fine-tune, and a personal-vocabulary tune to catch tech jargon and Hebrew loanwords like "makolet." We dig into what Parakeet actually is under the hood — FastConformer encoders with CTC, RNN-T, and TDT decoders, shipped through NVIDIA NeMo — and why its transducer architecture sidesteps the hallucination loops Whisper is famous for. Then we separate the two questions: personal vocabulary turns out to have a lightweight, already-shipped answer in contextual biasing and hotwords, while Hebrew is a genuine from-scratch language adaptation with a tokenizer wall, a data problem, and a real Android export bottleneck. We also bust the myth that only Whisper can be fine-tuned, and explain why starting from multilingual v3 beats adapting the English-only v2.

TRANSCRIPT



Corn

What would it actually take to teach a small on-device speech model a language it has never heard?



Herman

And whether you'd even need to.



Corn

That's the two-part question Daniel sent in this week, and it comes out of something he's been living with for a while. He's been running NVIDIA's Parakeet models on his Android, and he says it's a qualitative step up from Whisper. Which, from somebody who uses dictation all day, every day, is not a casual compliment.



Herman

No, it isn't.



Corn

So here's his situation. He uses Parakeet V2, the English-only one, because most of what he dictates is English. But he lives in Israel, so sometimes he wants Hebrew. And he's made an argument I think is worth taking seriously: if you only ever speak one or two languages, a multilingual model is a crapshoot at guessing which one you're in. Two models, one per language, makes more sense to him. Then there's the second thing, which is a personal-vocabulary fine-tune for English, to catch his tech jargon and the code-switching he does even in English. His example, and it's a good one: "I'm going out to pick up milk at the makolet." Makolet is an everyday Hebrew word for a corner shop. English ASR models butcher it almost every time.



Herman

Every time.



Corn

He also flags a stubborn myth he keeps running into, that only Whisper can be fine-tuned, and says he's seen linguistic fine-tunes for smaller on-device models but none in Hebrew. So, two questions, both scoped to the NVIDIA models with inference on Android. First, what's entailed in creating a Hebrew fine-tune. Second, can you fine-tune these smaller models for personal vocabulary, including the code-switching case. And for both: data prep, resources, time, and whether it'll actually run on Android when you're done.



Herman

There's a lot in there, and one of those questions has a much happier answer than the other.



Corn

Which one.



Herman

The makolet one. But let's set the stage first, because Parakeet deserves a proper introduction.



Corn

Go.



Herman

Parakeet is a family of ASR models from NVIDIA, built on what's called a FastConformer encoder, paired with one of three decoders: CTC, RNN-T, or TDT, which is Token-and-Duration Transducer. They're trained and shipped through NVIDIA NeMo, and the checkpoints live on HuggingFace under the nvidia namespace.



Corn

And the specific ones Daniel's using.



Herman

parakeet-tdt-0.6b-v2 is the English-only one. It sat at number one on the HuggingFace OpenASR Leaderboard as of May 2025. Then v3 is the multilingual successor, covering twenty-five European languages. There are also CTC and RNN-T siblings at 0.6 and 1.1 billion parameters, a unified English model, a streaming multi-speaker model. It's a real family, not a single checkpoint.



Corn

Why is it such a step up from Whisper on a phone, though. Because Whisper is the thing everybody reaches for.



Herman

Architecture. Whisper is an encoder-decoder transformer. It generates tokens autoregressively, one at a time, and it's famous for hallucination loops, where it gets stuck repeating a phrase that was never said. Parakeet TDT is a transducer. It emits tokens with a duration predictor, so it knows how long each token lasts. That's dramatically more efficient, and it doesn't have the repetition failure mode Whisper is known for.



Corn

So it's not just faster. It's less likely to invent things.



Herman

Both. And the size story matters here. The 0.6 billion parameter Parakeet is small enough to run on a CPU. There was a benchmark paper this year that ran fifty-plus configurations across Whisper, Nemotron, Parakeet TDT, Canary, Conformer Transducer, Qwen3-ASR, and found NVIDIA's models the strongest candidates for resource-constrained hardware. Their quantization work took a two point four seven gigabyte model down to zero point six seven gigabytes while keeping word error rate within one percent absolute of the full-precision baseline.



Corn

One percent. That's the part people don't believe.



Herman

It's the part that makes it viable on a phone. And there's real-world confirmation. Multiple builders have standardized on Parakeet for local dictation, and at least one commercial tool reportedly uses an in-house fork of it that handles tech jargon better than vanilla Parakeet.



Corn

Which is interesting, because that's exactly the problem Daniel's describing.



Herman

It's the same problem at a different scale. So, the language question first. Is Daniel right that a multilingual model is a crapshoot when you only speak one or two languages?



Herman

Whisper-family models do language identification internally. They guess, per utterance, which language you're in, and they can flip mid-utterance. Short dictation bursts are the worst case, because there isn't enough audio to be confident, so a two-second "yeah, send it" can get interpreted as Spanish. Code-switching is the other killer. If you drop a Hebrew word into an English sentence, the model has to decide whether you've changed languages or just used a loanword.



Corn

And it can't know.



Herman

It can't know. There's no signal. So it guesses, and sometimes it guesses wrong, and you get a transcription that's in the wrong script entirely.



Corn

What's the industry's own answer.



Herman

Separate models. That's the thing. sherpa-onnx ships separate Parakeet conversions. An English-only v2, a twenty-five language v3, a Russian GigaAM. You pick one and load it. Daniel's "two ASR models, one per language" plan isn't a hack. It's exactly the deployment pattern the tooling expects.



Corn

So the ecosystem already agrees with him.



Herman

The ecosystem already agrees with him. And on Android specifically, sherpa-onnx lets you load a different model directory per recognizer instance. So an app can hold an English Parakeet and a Hebrew model side by side and route by a UI toggle. There's also a dedicated spoken language identification module if you want it to switch automatically, and you can use Whisper purely as a language ID front-end and hand off to Parakeet for the actual transcription.



Corn

Whisper demoted to a bouncer.



Herman

It's a good bouncer. It's just a bad transcriber.



Corn

Now the myth. Only Whisper can be fine-tuned.



Herman

It's a myth. NVIDIA's own documentation says the opposite. I'll quote it, because the wording is the whole point: "Pretrained checkpoints for all of these models... You can use the available checkpoints for immediate inference, or fine-tune them on your own datasets."



Corn

That's not ambiguous.



Herman

It is not. Parakeet checkpoints are standard NeMo dot nemo files. NeMo ships full fine-tuning example scripts under examples slash asr, config files for every architecture, FastConformer CTC, Transducer, TDT, Hybrid. There's a Hybrid-Transducer-CTC model class with prompt conditioning that's explicitly designed for multi-language ASR from a single unified model using learnable prompt embeddings. And there's an ASR-TTS hybrid that lets you fine-tune using text-only data by generating synthetic mel-spectrograms.



Corn

Which matters if you have Hebrew text and not much Hebrew audio.



Herman

Exactly that use case.



Corn

So why does the myth persist.



Herman

Documentation and community. Whisper has more tutorials, more blog posts, more people answering questions on forums. So it feels like the only fine-tunable option. The tooling for Parakeet exists. It's just less written about.



Corn

Right. So Hebrew. And I suspect this is where the mood changes.



Herman

This is where the mood changes. Hebrew is not in Parakeet v3's twenty-five languages. That list is entirely European. Bulgarian, Croatian, Czech, Danish, Dutch, English, Estonian, Finnish, French, German, Greek, Hungarian, Italian, Latvian, Lithuanian, Maltese, Polish, Portuguese, Romanian, Slovak, Slovenian, Spanish, Swedish, Russian, Ukrainian. No Hebrew. No Arabic. No Semitic language at all.



Corn

So there's no checkpoint that speaks it.



Herman

There is no NVIDIA Parakeet checkpoint that speaks Hebrew today. Which means a Hebrew fine-tune is not a tweak. It's a from-scratch language adaptation.



Herman

The tokenizer is the first wall. Parakeet uses a BPE subword tokenizer trained on its original corpus. Hebrew characters are very likely not in the vocabulary at all. So you either extend the tokenizer with Hebrew subwords and resize the embedding and decoder layers to match, or you retrain the tokenizer outright. Both of those mean re-initializing parts of the model and then training them back up.



Corn

You're not teaching it a new accent. You're teaching it a new alphabet.



Herman

You're teaching it a new alphabet and a new phonology, and Hebrew has optional diacritics, which is its own mess.



Corn

What about data.



Herman

Ivrit.ai is the leading open Hebrew speech dataset project, thousands of hours of transcribed Hebrew. Common Voice Hebrew is smaller but free. And there's a third route, which is YouTube audio with Whisper-generated pseudo-labels. A September 2026 paper did exactly that across seven languages including Hebrew, and found that with a modest amount of fine-tuning data, average error rate dropped from thirty percent to twenty percent.



Corn

Thirty to twenty. That's the realistic expectation.



Herman

Not ninety-five percent. Twenty percent error is still one word in five wrong. It's a real improvement and it is not a solved problem.



Corn

Give me the effort estimate, and be honest, because I think people hear "fine-tune" and picture an afternoon.



Herman

Environment setup, NeMo install, a GPU, a twenty-four gigabyte card is comfortable for a 0.6 billion parameter fine-tune: one to two days. Data prep, collecting and cleaning Hebrew audio, building the NeMo manifest, which is a JSONL file with audio filepath, text, and duration, plus the tokenizer extension: one to three weeks. Training, fine-tuning from parakeet-tdt-0.6b-v3, using either a CTC or TDT head: likely fifty to two hundred GPU-hours, which is days to weeks of wall clock. Evaluation on held-out Hebrew plus a code-switching test set: days. And then export to Android, which I want to come back to, because that's the real bottleneck.



Corn

So we're talking a month, minimum, for a competent person.



Herman

A month if nothing goes wrong, and something will go wrong.



Corn

One recommendation I want to pull out, because it's counterintuitive. Start from v3, the multilingual model, not v2, the English one.



Herman

Yes. v3's encoder already has multi-language representations and a larger tokenizer. It's a much shorter path to Hebrew than adapting an English-only model, even though v2 is the one Daniel actually uses day to day.



Corn

You're borrowing the multilingual machinery even though you don't want the multilingual behavior.



Herman

You want its ears. You don't want its opinions.



Corn

Now the second question, and you said this one has a happier answer.



Herman

Much happier. For personal vocabulary and code-switching, you probably should not fine-tune at all.



Corn

That's a strong opening.



Herman

There are two lighter-weight techniques that solve it better, and both are already implemented in the on-device runtimes. The first is contextual biasing. Also called phrase boosting, or hotwords.



Corn

Hotwords.



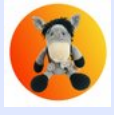
Herman

parakeet.cpp ships phrase boosting as a first-class feature, context biasing via a token-level trie for domain-specific vocabulary. sherpa-onnx has a dedicated hotwords module, and its Parakeet configs expose two parameters, hotwords file and hotwords score.



Corn

Explain what happens at decode time.



Herman

You supply a list of words or phrases. Makolet. OpenBao. Kubernetes. Your colleagues' names. At decode time, the model is biased toward those token sequences. No retraining. No GPU. Instant iteration. You add makolet to the list and the English model produces it correctly.



Corn

That's it.



Herman

That's it. And there's a caveat from practitioners, which is that Whisper's equivalent priming prompt is limited to about two hundred and twenty tokens, so hotword lists have practical size limits. But for a personal vocabulary of a few hundred terms, that's plenty.



Corn

So the fix for the thing Daniel's been fighting for months is a text file.



Herman

A text file and a score parameter. Which is why I said the mood changes in the other direction.



Corn

What's the second technique.



Herman

N-gram language model fusion. parakeet.cpp supports ARPA n-gram language model fusion scored at word boundaries, plus neural language model rescoring of N-best lists. You build a small n-gram model from your own text corpus, your emails, your notes, your docs, and fuse it at decode time. That captures your personal jargon and your Hebrew loanwords statistically, without touching the acoustic model at all.



Corn

So the acoustic model stays general and the language model gets personal.



Herman

That's the split. And it's a clean split, because your vocabulary changes constantly and your voice doesn't.



Corn

Now, if somebody does want a true fine-tune. Because some people will.



Herman

LoRA, or adapter fine-tuning. It's supported in NeMo through the adapters API, and it's the right approach for personalization. You train a small adapter on a few hours of your own voice plus transcripts, you keep the base model frozen, and you can swap adapters per user. Far cheaper than full fine-tuning, and it's the standard personalization pattern.



Corn

How much of your own voice.



Herman

Realistically one to ten hours of transcribed speech for a meaningful adapter. And the more code-switched utterances you include, Hebrew words inside English sentences, the better it handles makolet.



Corn

Here's the thing I want to flag, because I think it's the conceptual trap. A model fine-tuned on English with Hebrew loanwords is a different model from a Hebrew ASR model.



Herman

Completely different. And Daniel's instinct to keep them separate is right. One English model with Hebrew loanwords in its vocabulary, one Hebrew model. Trying to make one model do both is exactly what creates the language ID crapshoot in the first place. He'd be recreating the problem he's trying to solve.



Corn

So the architecture of his plan is correct. It's just that one half of it is a text file and the other half is a month of work.



Herman

That's the whole episode in one sentence.



Corn

Now the Android part. And there's a terminology thing I want you to correct, because Daniel says something in his prompt that I don't think is right.



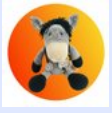
Herman

He says NVIDIA GGUF models.



Corn

He does.



Herman

GGUF is the llama.cpp format. It's for large language models. The on-device Parakeet ecosystem is ONNX, not GGUF. The runtime that actually runs Parakeet on Android is sherpa-onnx, from the k2-fsa project.



Corn

So the format's wrong but the intent is clear.



Herman

The intent is completely clear. He means the quantized on-device versions. He's just got the container format mixed up. Easy mistake, because GGUF is what you hear about for local models.



Corn

What does sherpa-onnx actually give you on Android.



Herman

Official support for arm64-v8a, armeabi-v7a, and x86_64. Prebuilt AAR libraries. Reproducible F-Droid builds, which were added September 21, 2026. Kotlin and Java APIs for integration. And prebuilt APKs for testing, including a simulated streaming ASR build for Parakeet TDT 0.6b v2 on arm64, and the v3 multilingual equivalent.



Corn

Sizes.



Herman

The int8 Parakeet v2 encoder is six hundred and twenty-two megabytes. v3 int8 is six hundred and twenty-two megabytes for the encoder, twelve for the decoder, six point one for the joiner. There's an fp16 variant too if you have the headroom.



Corn

And speed.



Herman

On an RK3588 Cortex-A76, which is a rough proxy for a mid-range Android SoC, Parakeet v2 int8 runs at a real-time factor of zero point zero eight eight with four threads. That's about eleven times faster than real time. On a modern flagship it's comfortably real-time with room to spare.



Corn

Eleven times faster than real time on a mid-range chip. That's the sentence that should end the "can a phone do this" debate.



Herman

It should. NPU acceleration is available for Qualcomm, Rockchip, and others, though Parakeet specifically is best supported on CPU through ONNX Runtime today.



Corn

So walk me through the export pipeline. Because this is where the Hebrew project actually lives or dies.



Herman

Four steps. One: fine-tune in NeMo, which produces a dot nemo checkpoint. Two: export to ONNX using NeMo's export script. For transducer models that produces separate encoder, decoder, and joiner ONNX graphs. And note, cache-aware streaming models need the export config set for cache support before you export. Three: quantize to int8 using sherpa-onnx's conversion scripts. That's where the six hundred and twenty-two megabyte encoder comes from. Four: drop it into the Android app. You replace the model directory, and sherpa-onnx loads encoder dot int8 dot onnx, decoder dot int8 dot onnx, joiner dot int8 dot onnx, and tokens dot txt, with the model type set to nemo transducer.



Corn

And the gotcha.



Herman

Steps two and three are where custom fine-tunes most often break. If you extended the tokenizer for Hebrew, the exported ONNX graphs have to match the new vocabulary size, and sherpa-onnx's loader expects specific tensor shapes. Budget real time for this. It is the least-documented part of the entire pipeline.



Corn

So the training, the part everybody thinks is the hard part, is maybe the most predictable part.



Herman

By a wide margin. The training is a known quantity. You have a loss curve. You know if it's working. The export is where you find out that the graph you produced has a different output dimension than the loader expects, and there's no error message that tells you why.



Corn

That's a strange place for the difficulty to live.



Hilbert

It's not strange. It's the same place it's always lived.



Corn

Go on.



Hilbert

I did a stint doing transcription QA for a captioning shop. You'd get a stack of audio files and a deadline, and you'd decide whether to trust the machine output or just type it yourself. There was an argument with a supervisor about whether makolet should be capitalized.



Corn

Who won.



Hilbert

Nobody. It went in lowercase and stayed there. The point is we kept a shared spreadsheet of words the machine always got wrong. Names of local businesses, a few medical terms, and one entry that just said, the thing Ruthie says instead of okay. That last one got used more than anything else on the sheet.



Herman

That's the hotword list.



Hilbert

And it was a spreadsheet. The companies keep chasing bigger models, and the fix is letting people add their own words. We had it in a spreadsheet in a captioning shop.



Corn

What was the thing Ruthie said instead of okay.



Hilbert

Doesn't matter. It's on the sheet. Anyway, I'm late for an appointment.



Corn

What's the most common wrong belief here.



Herman

That fine-tuning is the first thing you reach for when you want a model to know your vocabulary. It isn't. Hotwords and n-gram fusion solve the makolet problem with a text file and no retraining, and both are already in the runtimes people are using.



Corn

And the second most common.



Herman

That these models can't be fine-tuned at all. NVIDIA's own documentation says they can. The tooling is there. It's just less written about than Whisper's.



Corn

Here's what I keep circling back to. If hotwords and n-gram fusion already solve the personal-vocabulary problem, why do so many people still reach for fine-tuning first?



Herman

Because fine-tuning feels like real machine learning. You're training something. There's a loss curve. Hotwords feel like a config file, even though they work better.



Corn

And the lighter-weight tools are badly documented, which makes them feel unserious.



Herman

That too. The interesting shift is that as on-device ASR gets better, the bottleneck moves off the model and onto the tooling. The export pipeline, the runtime compatibility, the documentation. The Hebrew fine-tune is a multi-week project whose hardest part is the ONNX export, not the training.



Corn

Daniel's right that Parakeet is fine-tunable. He's right that separate per-language models beat one multilingual model for dictation. But the fastest win for makolet is hotword biasing, not a fine-tune, and the Hebrew model is a multi-week project whose hardest part is getting it onto the phone.



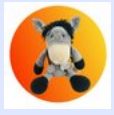
Herman

That's the episode.



Corn

Thanks as always to our producer, Hilbert Flumingtop. This has been My Weird Prompts.



Herman

If you're building on-device ASR, the sherpa-onnx docs and the parakeet.cpp repo are where the real answers live. Not the blog posts about Whisper.



Corn

Email us at show at my weird prompts dot com. We'll be back soon.