Episode Book

EPISODE #6162

Why Dictated Words Rewrite Themselves

PUBLISHED	20 October 2026 at 14:03 UTC
DURATION	25:06
LISTEN	myweirdprompts.com/episode/streaming-asr-interim-final-rewrites-12832/
HOSTS	Corn, Herman
SCRIPT MODEL	anthropic/claude-sonnet-5.5
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.10
BOOK GENERATED	21 October 2026 at 07:50 UTC

SYNOPSIS

You dictate into a voice prompt, watch a word appear in gray, and the moment you release the key it quietly becomes a different word. That flicker is not a bug — it is the seam between interim and final speech recognition, and the vocabulary for it exists across Google, Amazon, Microsoft, and Deepgram under different names. This episode unpacks what "dimmed until finalized" actually promises, why the tail of a sentence is always the unreliable part, and the three architectural families that produce rewrites: LocalAgreement re-decoding over a growing window, two-pass rescoring with a full-context decoder, and cascaded encoders. It also covers what NVIDIA's Nemotron streaming model can and cannot do, why a pure streaming transducer never takes a word back, and what eager end-of-turn speculation costs in extra language model calls.

TRANSCRIPT



Corn

...and that's the part that bugs me. The word is sitting there, dimmed, looking perfectly confident, and then you let go of the key and it quietly turns into a different word. Nobody announced the edit.



Herman

You're mid-thought about Daniel's prompt, aren't you? You've been chewing on this since breakfast.



Corn

Slowly, yes. So, Daniel dictates into Claude Code's voice mode. Hold the spacebar, text streams into the prompt as he talks. Every so often it mishears a word, and when he releases the key, that earlier mistake gets rewritten. He wants to know what this pattern is called, whether open-weights models like NVIDIA's Nemotron streaming model and Parakeet can do it, and whether that behaviour has to be trained into a model or is just a trick in how you run it. One thing first: Anthropic doesn't say who does the transcription. The docs say only that audio goes to their servers. So we're talking architecture, not vendor, and we won't guess at a vendor.



Herman

Good, because the architecture is where the interesting stuff is, and the docs do give us one real clue. The line is that your speech appears as you speak, dimmed until the transcript is finalized, and releasing Space stops recording and finalizes the text. That's it. That's the whole public description.



Corn

Dimmed until finalized. So the dimming is the interface admitting it isn't sure.



Herman

Right, it's a two-state display. Gray text means provisional, normal text means committed. And that maps onto vocabulary that every speech API uses, so let me give you the names, because Daniel asked what it's called. The generic terms are interim results, or partial results, versus final results. A partial hypothesis is the model's best guess so far, and it is



Herman (continued)

allowed to change. A final is the model saying this stretch of audio is closed.



Corn

Is there one name for the whole phenomenon? The word flipping?



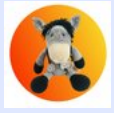
Herman

Informally people say flicker, or churn. I haven't seen a vendor document use those, so take them as slang. The formal-ish idea is stability. Google's streaming recognizer puts a stability score on interim results, a number where zero means unset, and their advice is to show the user only the high-stability ones. Amazon does it differently. Transcribe marks each word with a stable flag, and you pick a stabilization level, high, medium or low, and only the last few words can still change. Faster but less accurate, if I'm remembering their trade-off right.



Corn

So the tail of the sentence is the unreliable part.



Herman

Always the tail. The model has heard the start of a word and none of its end. And Microsoft has the same split under different names: a recognizing event for intermediate results subject to change, a recognized event for the final text.



Corn

Three clouds, three spellings of the same idea. Impressive coordination for companies that agree on nothing else.



Herman

There's a second concept that matters here, which is endpointing. Something has to decide that you've finished speaking, so the service can issue a final. Deepgram's API is a good example. It sends interim results flagged not final. Then a flag for final means that



Herman (continued)

span is closed and they won't return anything more for it. A separate flag says speech final, meaning the endpointing logic heard a pause. In a long utterance you can get several finals before the speech-final one.



Corn

Hold on. I'm not naming a vendor here, but if finals are a promise not to revise, then Daniel's observation is odd. He sees a word that's already on screen get rewritten at release.



Herman

That's the tension, and I want to be careful. In that style of API, interim text revises by design, but text marked final doesn't. So when the key goes up and a word changes, there are two stories that fit. Story one: the client has been showing only interim text, dimmed, the whole time. On release the server sends the final transcript, and the client replaces the whole dimmed string with it. Looks like a rewrite, but it's just swapping a draft for the finished version, and the draft had never been committed.



Corn

And story two?



Herman

A genuine second pass. The server, once it knows you've stopped, runs something heavier over the whole utterance, with full context, and returns a better answer than the streaming model could manage. Nothing in the docs tells us which one it is. I don't know, and I don't think anyone outside Anthropic does.



Corn

Though the dimming suggests the first story at minimum. If the text were already trusted, it wouldn't be dimmed.



Herman

Yes. And the two stories aren't exclusive. You could have both: interim text that gets replaced by final text, where the final text came from a second pass. I'd say that's quite plausible, because hold-to-talk gives you something rare. Release is a perfect endpoint.



Corn

Meaning?



Herman

Normal dictation has to guess when you're done. Voice activity detection, silence timers, a pause threshold you tune forever. Too short and it cuts you off mid-breath, too long and it feels sluggish. With a held key, your finger tells the system. There's no guessing, and the system also knows the full audio is in hand. That's the ideal moment to spend extra compute, because the user is already waiting for a final anyway.



Corn

Okay, now I'm curious how a model even produces the first draft. Let me guess how the streaming models work, and you tell me where I'm wrong. You feed audio in chunks, it emits words as it goes, and if it got one wrong, it just changes it later.



Herman

That's wrong, and it's the most useful thing to get straight. A modern streaming transducer, the kind NVIDIA ships, emits a token and moves on. It doesn't reach back. Mechanically, a greedy RNN-T decode is append-only. I should flag that this is my reading of how the decoding works, not a line from NVIDIA's documentation. But the structure is: at each step it either emits a token or advances to the next frame.



Corn

So a pure streaming transducer would never rewrite anything on its own.



Herman

Right. Whatever it said, it said. The revision has to be bolted on somewhere else, and there are a handful of ways to bolt it on. I'll walk through them.



Corn

Start with the cheapest one. I'll bet it's a hack.



Herman

It is a hack, and a good one. It's called LocalAgreement, from the Whisper-Streaming project, by Machacek, Dabre and Bojar. Whisper is an offline model. It wants a whole clip. So they feed it a growing window of audio, and every time new audio arrives they re-run the whole window from scratch. Each pass gives a transcript. A prefix of the text is only confirmed once consecutive passes agree on it. The tail that disagrees stays tentative, and it can change on the next pass. Once a sentence is confirmed, they trim the buffer so it doesn't grow forever.



Corn

So the model is never streaming. The wrapper pretends.



Herman

The wrapper pretends, and the revision is a natural side effect. Every re-run is a chance for the model to change its mind about the last few words. No training involved. You could do the same thing with any offline model.



Corn

That's wasteful. You're redoing the same audio over and over.



Herman

Heavily, yes. NeMo's docs actually describe this family too, buffered streaming with overlapping chunks, and they say plainly there's duplicated compute. The alternative they push is cache-aware streaming, where the model stores intermediate activations so it



Herman (continued)

never recomputes the past. But that's a trained property, and I'll come to it.



Corn

Fine. What's the next way to get a rewrite?



Herman

Two-pass. This one's from Google, Sainath and colleagues, around twenty nineteen. A streaming RNN-T gives you fast first-pass hypotheses. Then a second model, an attention-based listen-attend-spell decoder, shares the same encoder and rescores the candidates once the utterance ends. The reported gain was somewhere between seventeen and twenty-two percent relative word error rate reduction over the transducer alone, with a small latency cost. That's from the abstract snippet, so I'd treat the exact range as indicative.



Corn

So the first pass talks, the second pass corrects.



Herman

And the user sees the correction land at the end of the utterance. Which is exactly the shape Daniel describes. There's an open-source cousin too. WeNet's U two, and its follow-up U two plus plus. One hybrid CTC and attention model. The CTC branch streams out candidates as you speak, then the attention decoder rescores them at the end of the utterance. And the training trick is something called dynamic chunk training: they randomly vary the chunk size during training, so one set of weights can run in streaming mode or see the whole utterance at once.



Corn

Same weights, two modes.



Herman

Same weights. Then there's a third family, cascaded encoders, from Narayanan and colleagues. You stack a non-causal encoder on top of a causal one, with one shared decoder. In streaming mode you only use the causal part, and the accuracy matches a dedicated streaming model. In full-context mode you run both, and they report ten to twenty-seven percent relative improvement. Again from a snippet, so I'd say roughly that range.



Corn

Hm. So the pattern across all three: one cheap, fast, short-sighted pass, then one expensive, patient pass.



Herman

The umbrella name for what Daniel's describing is two-pass ASR. Streaming first pass, full-context second pass. The user experience is interim then final.



Corn

I want to push on lookahead, though. You said cache-aware models see a bit of the future. Doesn't that count as revision? The model waits a moment, then decides.



Herman

Lookahead delays the decision, it doesn't undo it. A chunked model with right context sees a few extra frames before committing a word. So a word shows up later but it's less likely to be wrong. That's trading latency for accuracy. A true rewrite means something already shown gets replaced, and a pure lookahead model doesn't do that.



Corn

Is there a version of this where the revision is the feature, not a side effect? Where the system wants to guess early and take it back?



Herman

Voice agents do exactly that. Deepgram's Flux model has an eager end-of-turn event: the system thinks you've probably finished, so the agent starts drafting a reply. Then, if you keep talking, a turn-resumed event fires so the app can cancel the draft. Their docs warn that turning on the eager mode costs you somewhere between fifty and seventy percent more calls to the language model. The revision there is about your intent rather than a single word, but it's the same bargain: speculate early, pay for the retractions.



Corn

So the wrong guesses have a price tag. Good to know before I get confident.



Herman

And that price is why hold-to-talk is such a gift for dictation. You never need to speculate about whether you're done.



Corn

Daniel asked about NVIDIA specifically, so what do they actually have?



Herman

Quite a bit. The model that matches his name is the Nemotron speech streaming model, the English one, around six hundred million parameters. It's a cache-aware FastConformer encoder with twenty-four layers and an RNN-T decoder. And the neat trick is that you pick your latency at inference time, with no retraining. You set the attention context, and the right context can be zero, one, six or thirteen frames of eighty milliseconds each. So that's eighty milliseconds up to roughly eleven hundred and twenty milliseconds of chunk. Small chunk, snappy and a bit sloppier. Big chunk, slower and more accurate.



Corn

Hold on, I want to make sure I follow. It's one model, and the latency knob is a runtime setting.



Herman

One checkpoint, yes. It was trained to cope with those different context sizes, and then you choose at serving time. It also does punctuation and capitalisation natively, which matters for dictation. There's a newer one too, Nemotron three point five, streaming, forty locales selected with a language identification vector. Hebrew is listed in the adaptation-ready tier, which as I read it means you'd fine-tune before relying on it.



Corn

That's a thing worth knowing before anyone gets excited. Adaptation-ready is a polite way to say not ready.



Herman

Fair, but the point is that it gives you a starting point instead of a blank page. And there's a small one, Parakeet realtime EOU, about a hundred and twenty million parameters, cache-aware streaming, which emits an end-of-utterance token and has latency in the eighty to one hundred and sixty millisecond range. English only, and per the snippet I saw, no punctuation or capitalisation. I haven't read the full card, so hold that loosely.



Corn

And Parakeet itself? The big one Daniel mentioned.



Herman

This is where the name causes confusion. Parakeet TDT zero point six B, version two, is an offline model. A single pass over a whole clip, up to roughly twenty-four minutes of audio. Its card doesn't claim streaming. Its average word error rate on the open ASR leaderboard is about six percent. So when Daniel asks can Parakeet do it, the answer depends on which Parakeet.



Corn

The streaming Parakeets stream. The accurate Parakeet doesn't.



Herman

Correct. Though nothing stops you running the LocalAgreement trick on the offline one. Re-run Parakeet over a growing window, confirm the prefix when passes agree. It's an inference construction, so it works on any model that takes audio and returns text. It would just burn a lot of GPU for something a cache-aware model does natively.



Corn

So you could make the accurate model fake streaming, at a cost, or the streaming model do its best, at a different cost.



Herman

That's the whole design space in one sentence, yes. Also, small models are entering this. Moonshine's second version uses sliding-window attention with cached encoder state for its streaming variants, from a tiny one around thirty-four million parameters up to a medium one. Both ends are in the range where a laptop CPU is plausible, though I've only seen that from secondary summaries.



Corn

Daniel's laptop already runs half a dozen things he forgot he started.



Herman

Ha. Back to NVIDIA. Here's what I find genuinely useful. Nothing on the NeMo models page describes a built-in rescoring or second-pass feature. The related options listed are things like language model fusion. So NVIDIA gives you the pieces, a streaming model and an offline model, but not a packaged two-pass system.



Corn

That's quite a gap. They sell the horse and the cart separately.



Herman

Ha. Let me pick up the training question, because Daniel asked it directly: does revision need to be trained in, or is it a serving construction? And the answer splits cleanly. Say it back to me and tell me if it holds up.



Corn

Go. I'll try. The things that make a model good at streaming have to be trained: the cache-aware design, the limited-context training so it copes with a short window, U two's dynamic chunks, the cascaded encoder, Google's shared-encoder two-pass. All of those are baked into the weights.



Herman

Yes. And Kyutai's approach belongs there too. Their speech-to-text models use what they call delayed streams: the text stream is deliberately shifted behind the audio, so the model gets lookahead built in. The English and French one has a delay of about half a second, I believe. They also describe a flush trick, where after speech ends they process the buffered audio faster than real time, so the final words arrive in something like a hundred and twenty-five milliseconds instead of five hundred. I'm going from their page, and I haven't seen it benchmarked independently.



Corn

And the stuff that needs no training at all?



Herman

Anything where you run an offline model over more audio. LocalAgreement is that. And the plainest version of all: record the whole clip, and on release, run an offline model over the full buffer and replace whatever's on screen. No paper required. That's a pure serving decision.



Corn

Which is also the one I'd bet on for Claude Code, if you made me. A held key, a WebSocket, a final transcript arriving after release. But I'm guessing, so file it under Corn's hunches.



Herman

Filed. I'd add that the recognition hints in the docs, project name and git branch sent along to bias the recognizer, tell you the service supports keyword biasing. That's compatible with either design. It doesn't discriminate.



Corn

You keep saying that nothing discriminates. Is there anything that would tell us from the outside?



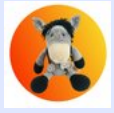
Herman

Actually, yes, and Daniel could test it. If the final text sometimes differs from the dimmed text in places far from the end of the sentence, a word in the first half, then something with wide context ran again. A pure streaming model, even one with lookahead, mostly changes the tail. Changes deep in the sentence point toward a real second pass. I haven't seen anyone publish that kind of test on this product, though.



Corn

So he becomes the experiment. He's going to love that. He already dictates everything.



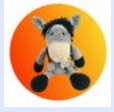
Herman

Do you remember how you reacted when I first explained buffered decoding to you? You said it sounded like a man re-reading his whole shopping list every time he adds eggs.



Corn

I said that? I don't remember saying that.



Herman

You did not say that. I just made it up to see if you'd accept it.



Corn

Disappointing. Though it's a good description of LocalAgreement.



Herman

It is. Here's my real question for you, Corn, and I mean it. If you were building this for yourself, would you accept the little flicker at the end of the sentence, or would you hold back the last few words until they stabilise?



Corn

Hold back. I'd rather see text appear a beat late than watch a word change under my eyes. The dimming helps, but if I'm reading while talking, a changed word breaks my train of thought.



Herman

Interesting, because I'd go the other way. I'd take the flicker for the feeling of immediacy. When you see your words appear at once, you trust the thing is listening.



Corn

You like the donkey-in-a-hurry version.



Herman

I do. Though the stabilisation approach is genuinely what Amazon offers, so you're in good company.



Corn

Right, so let's build it. Daniel's an open source developer, and the brief says end on something he could actually assemble.



Herman

Okay. Two models and a little glue. Model one is a streaming model for the live text. The Nemotron speech streaming checkpoint, with a small or medium chunk setting, is the obvious candidate. The small realtime EOU Parakeet would also work if he doesn't need punctuation live. While the key is held, audio goes to that model and the text appears dimmed. At the same time, and this part matters, he saves every audio frame to a buffer.



Corn

Because the second model needs the whole clip.



Herman

Exactly that. On key release, the saved audio goes to Parakeet TDT version two, the offline one. One pass over the full clip, it has complete context, and the result replaces the dimmed text. Clips of a few minutes are well within its limit of about twenty-four minutes, and Claude Code itself cuts off at two minutes anyway.



Corn

So no training, no fine-tuning, no cleverness. The cleverness is in when you run which model.



Herman

There's one honest unknown. I don't have a measured latency for that second pass. A six hundred million parameter FastConformer on a decent GPU should be fast, since it's a single forward pass rather than a growing-window loop, but I haven't seen a number, so he'd want to time it. If it's slow, he could even start the offline pass on the last chunk of audio a moment before release.



Corn

Hm, and swapping the text raises the question of what happens to the cursor. If he's already started typing something after the dictation.



Herman

Then he's in trouble. The cleanest trick is the same one the product seems to use: freeze editing of the dictated region until the final lands. Hold-to-talk makes that natural, because people aren't typing while they hold the key.



Corn

Unless they're doing the thing where they hold Space and also fix a typo with the other hand.



Herman

That is a person I have never met.



Corn

I live with one. Anyway. If he wants the fancier version, and he has time to train, U two style dynamic-chunk training gets one model that does both jobs. The cheaper route is two off-the-shelf models, and for dictation I think the two-model one wins. The accuracy of an offline model on the finished clip is hard to beat, and the licenses are friendly: the Parakeet card says Creative Commons Attribution, and the Nemotron cards carry NVIDIA's open licenses.



Herman

One more reason to prefer two models, and it's a point in your favour. You can swap either one independently. When a better streaming model arrives, and one is arriving seemingly every month, you replace half the system.



Corn

That's the first time you've agreed with me without adding a footnote.



Herman

Footnote: it only works if the buffer logic is clean.



Corn

There it is.



Herman

There's a broader point under all this. What looks like one clever product feature is mostly a division of labour. A fast model with short sight handles the live experience, a slower model with full sight handles the truth, and the interface is honest about the difference by dimming the first one. Whoever is behind Claude Code's voice mode, that pattern is the one that fits everything the docs say.



Corn

I'd add that the pattern shows up the moment you stop treating the live text as the answer. The live text is a progress indicator that happens to be made of words.



Herman

A progress indicator that happens to be made of words. I'd use that in a talk.



Corn

Take it. I'm busy napping.



Herman

So the forward-looking bit. What I'd watch is whether open streaming models start shipping the second pass inside the same checkpoint. The research is there, U two and cascaded encoders both do it. If someone releases a cache-aware model with a built-in full-context mode, Daniel's two-model recipe collapses into one download.



Corn

And until then, he's got a spacebar and a buffer. Thanks to Hilbert Flumingtop for producing, as ever.



Herman

This has been My Weird Prompts. If you want more, you'll find us at my weird prompts dot com.



Corn

See you tomorrow.