Episode Book

EPISODE #6422

Why Multi-Stage AI Pipelines Collapse in Production

PUBLISHED	3 November 2026 at 12:15 UTC
DURATION	23:14
LISTEN	myweirdprompts.com/episode/ai-pipeline-orchestration-serving-43989/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	z-ai/glm-5.3
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.26
SOURCES	25 cited by the research (listed at the end)
BOOK GENERATED	4 November 2026 at 09:31 UTC

SYNOPSIS

Multi-stage audio pipelines that chain preprocessing, speech recognition, diarization, and language models look neat as diagrams, but production architectures tell a different story. Running these graphs on major cloud providers exposes sharp divisions between training DAGs and inference serving. While tools like Ray Serve and Step Functions offer composition, operational friction around GPU allocation and timeout limits frequently collapses modular chains into single opaque containers or black-box managed APIs.

TRANSCRIPT



Corn

Five containers in a row on a whiteboard. Audio comes in, gets squashed to mono, silences get trimmed, transcribed, speakers split apart, an LLM tidies the whole thing into a transcript. That's the pipeline Daniel sent us — he's picturing it as a graph, five or six moving parts, and he notices the same shape turns up everywhere in AI now, image generation included. His question is what it actually takes to run one of these on a major cloud. Do people assemble the graph from scratch, is there real tooling for composing it, and once you've got weights and code for every stage, how do you stand the whole chain up as one service — and when a job slows down, how do you see which stage was stuck?



Herman

And the answer everyone wants — "here's the pipeline product, point it at your audio" — does not exist. The real answer splits into three questions that people keep collapsing into one, and the collapse is where almost all the confusion lives. Composition, packaging, observability. Three separate decisions, three separate tool sets.



Corn

So let's not collapse them.



Herman

Composition first, because it's the one Daniel's mental model is closest to right about. The graph is declared, not hand-wired. Nobody is sitting there writing their own scheduler that shuttles files between five processes — you write a description of the graph and a framework does the wiring.



Corn

On every cloud?



Herman

On every major one. Google's is Agent Platform Pipelines, which is the current name for what everyone still calls Vertex AI Pipelines. You write Python with the Kubeflow Pipelines SDK, it compiles to YAML, and Google spins up the orchestration compute for the run and tears it down after. Azure has pipeline components — a component packages the code,



Herman (continued)

the command, the environment, and the input and output interfaces of a single step, so you can reuse that step across pipelines without re-porting the dependencies every time.



Corn

That's the part I'd underestimated about components. They nest.



Herman

They nest, and you can share them across workspaces through a registry, which is Azure's answer to "my team's diarization step shouldn't be a private script on someone's laptop." And since late twenty twenty-three you can deploy a whole pipeline as a single versioned unit to a batch endpoint.



Corn

AWS being AWS, there are two of them.



Herman

There are two, and they're for different jobs, this isn't redundancy. SageMaker Pipelines is for SageMaker-native workflows — you get step caching, a model registry, that whole world. Step Functions is the general orchestrator, over two hundred service integrations, and it costs about two and a half cents per thousand state transitions, which for most pipelines rounds to nothing. The practitioner split that's settled into consensus: SageMaker Pipelines if your branching stops at quality gates and everything lives inside SageMaker anyway; Step Functions the moment you need to reach outside it.



Corn

"The orchestrator decision comes first. Get it wrong, and the friction compounds for months." That's the line I remember from reading about this — I should have been an orchestrator. Everyone's dependent on me and I make them wait.



Herman

You already are one. You're the Step Functions of this flat — nothing proceeds until you've finished.



Corn

True, but I bill less.



Herman

Then there's the framework-agnostic option, Ray Serve, and this is the one that gets closest to what Daniel is picturing, because it's designed for serving rather than training. You define each stage as a deployment — the downmixer, the ASR, the diarizer — and you bind them together by passing one into another's constructor. At runtime Ray hands each deployment a handle to the next one, so preprocessing, inference, and post-processing become independently scalable units. You can give ASR eight replicas and the downmixer one, because the downmixer is cheap and Whisper is not.



Corn

And this matters because of the trap Daniel's prompt is walking straight into.



Herman

The training-versus-serving trap. SageMaker Pipelines, Vertex Pipelines, Azure's components — all of that machinery is built for training DAGs. Re-train the model weekly, evaluate, register, promote. Using any of it to serve per-request inference is a category error — these are workflows that run on a schedule or a trigger, not endpoints that answer a user's upload in a few seconds. The serving-side equivalents are inference pipelines, batch pipeline deployments, and Ray's deployment graphs. Same word, "pipeline," completely different things.



Corn

Which is the single most annoying naming decision in the field, and the field contains Kubernetes.



Herman

It really is. You search "SageMaker pipeline" and half of what comes back is about retraining, which has nothing to do with the question Daniel asked.



Corn

Okay. So composition is covered — the graph gets declared. Now the question he's really asking: how does the declared chain become one deployable service?



Herman

This is where AWS has the most literal answer in the industry, and it's worth understanding precisely because of how it breaks. SageMaker inference pipelines. You define a model as a linear sequence of containers — two to fifteen of them — instead of one primary container. SageMaker handles an invocation as a chain of HTTP requests: the first container gets the request, its response becomes the request to the second, and so on down the line. All the containers are co-located on the same EC2 instances for latency. There's a config that runs them serially for the pipeline behavior, or you can invoke each container individually. No extra cost beyond the instances you were paying for anyway.



Corn

That is exactly the five-stage graph, one endpoint. Sounds like the answer.



Herman

It sounds like the answer, and then you read the fine print. Two limits. Multi-container endpoints are capped at fifteen containers, fine, nobody needs fifteen. But GPU inference is not supported on them, officially, due to resource contention on the instance. Your ASR stage wants a GPU. Your diarizer wants a GPU. The one thing an audio pipeline cannot live without is the thing the most literal graph-shaped product excludes.



Corn

So the mechanism that matches the picture is unavailable to the workload the picture is about.



Herman

That's the bind, yes. And the second limit — real-time SageMaker endpoints have to answer within sixty seconds. A long recording through ASR plus diarization does not fit inside sixty seconds. There's an asynchronous mode, InvokeEndpointAsync, where the audio goes in and out through S3, no response cap, autoscales to zero when idle, uploads up to a gigabyte. That's the mode production actually uses.



Corn

So what does AWS's actual production answer look like, if not five chained containers?



Herman

One fat container. They ship a WhisperX Deep Learning Container — a GPU-ready image with Whisper, the wav2vec2 forced alignment, and the diarization weights all bundled, no Hugging Face token needed. It follows the standard SageMaker serving contract: port eight zero eight zero, POST to invocations, ping endpoint for health, you send the audio as a file part with parameters for language, diarize, response format, and you get back JSON or subtitles. Inside, it runs voice activity detection, batched Whisper, alignment, then diarization — four of Daniel's five stages — behind a single endpoint.



Corn

One endpoint, one container, one bill. And one black box.



Herman

When it's slow, you know it's "the container." You don't know whether the VAD dragged, or Whisper batched badly, or diarization spent the whole budget. The five-node graph Daniel imagines is exactly the thing that would tell him that — and it's exactly the thing production collapses. Not because engineers are lazy. Because the packaging layer can't carry it: no GPUs in multi-container mode, sixty-second cap on real-time, and the operational simplicity of shipping one image.



Corn

There's a gotcha in there too, the kind I enjoy.



Herman

There is, and it's vicious. On the GPU variants of that container you have to pin a specific inference AMI version, and if you don't, the endpoint fails to start with a CannotStartContainerError and — this is the good part — zero log lines. Nothing. The container refuses to explain itself.



Corn

A doctor's dream patient. Comatose and mute.



Herman

I've had parents describe a fever that way. "He was fine yesterday." The absence of information is itself the diagnosis — it means the failure happened before anything could log.



Corn

And there's a third path on AWS, isn't there. The one where you give up on the graph entirely.



Herman

Bedrock Data Automation. Since autumn last year it does enhanced transcription with optional speaker diarization and channel identification — speakers labeled spk zero upward, up to thirty of them, channels up to two, timestamps included, configurable through the API. One call, you get a transcript with speakers. And AWS's own pitch for it is that it eliminates "data preparation, multiple model management, fine tuning, prompt engineering and orchestration." Read that list again — orchestration is on the list of chores they're saving you from. They're selling the death of the graph as a feature.



Corn

Thirty speaker labels for one API call is a lot. What you can't see is any of them.



Herman

Which of the thirty was slow. Whether the diarizer struggled on the overlapping section. Nothing. It's one stage from where you stand, even if it's five internally.



Corn

So that's the shape of part one, and I want to name it before we move on. What Daniel calls a "cohesive, logical pipeline" exists — the tooling to declare it, the mechanism to chain it — and the industry keeps choosing the fat container anyway, because the honest constraints push it there. We covered the from-scratch end of this when we did the custom ASR tools episode — the hand-chained repos — and the gap between that world and the WhisperX container is basically this whole story.



Herman

And the open-source world is still doing it the literal way, for what it's worth. There's a small repo called audiopipe, a couple hundred stars, chaining denoise, diarization, transcription. Another one, modular audio pipeline, does FFmpeg conversion, denoising, source separation, voice activity detection, Whisper, pyannote, redundancy filtering — the full hand-built graph.



Corn

Daniel is describing repos with two hundred stars.



Herman

He's describing repos with two hundred stars. That's the assembled-from-scratch baseline, and it's real, and it's maintained by one person each.
So that's the graph, declared, flattened, or black-boxed. Now the question Daniel cares about most, and the one with the least to say for itself. When it breaks or crawls, how do you see which stage it was?



Herman

I'll state the negative plainly, because I went in expecting to find a product: there is no first-class multi-model pipeline observability service that shows you per-stage inputs and outputs across heterogeneous stages. It does not exist on any of the three clouds. What exists are three partial pieces, and you stitch them.
Ivan: Nightjar to Wren, seven hundred and twelve, forty four, seven hundred and twelve.



Herman (continued)

The goose sits on two of three. The third egg is cold and the gate is the gate. The silence trimming begins at the second downmix. Seven hundred and twelve. Nightingale.



Herman

Message received.



Corn

What on earth was that?



Herman

Google's piece is the strongest of the three. Vertex ML Metadata logs every pipeline run automatically — artifacts, lineage, metrics, the whole execution graph, and you can open a run in the console and see the step logs and the artifact details for each stage. Plus step caching, on by default, keyed on a step's inputs, outputs, and component spec.



Corn

The caching is where I'd want Corn-like patience, because the failure is silent.



Herman

The Xebia write-up on this is worth knowing. Two traps. If you push a new container image under the same tag, the cache does not notice — it sees the same spec, it serves the same result. And if the data changed underneath you — a BigQuery table updated — the cache doesn't pick that up either. So you can re-run the pipeline on a Tuesday and get Monday's transcript, confidently, with everything reporting green.



Corn

A system that reports itself fine is the one you check first. Every clinician learns that or learns it the hard way.



Herman

Every parent too. "He's fine" at eight in the morning, fever by noon.



Corn

And Ray's piece?



Herman

Prometheus metrics out of the box, per node — request counters, latency histograms, error counters, replica utilization, on a rolling window. And their LLM stack adds the token-level metrics, time to first token and friends, with Grafana dashboards where you can drill from the service panel down to a specific replica. There's even a subtlety I find charming: the metrics differ depending on whether you call a deployment through Python handles or over HTTP, and the docs mark them separately.



Corn

That's the kind of footnote that tells you the seams are real. If the two call paths emit different telemetry, the "graph" was never one thing to begin with.



Herman

The cross-vendor answer, the only one, is OpenTelemetry's GenAI semantic conventions — still officially Development status. They define spans for logical operations as the caller observes them, including retries folded into the duration. The trace tree is built around agents: a top-level invoke-agent span, chat spans per LLM call, execute-tool spans per tool invocation, with attributes for the model, the token counts, the finish reasons. There's even a workflow duration metric meant for multi-step boundaries.



Corn

Chat spans and tool spans. Not "the diarizer took forty seconds and the ASR took six."



Herman

Not that. The conventions model LLM calls and tool calls. Stage three of an audio graph doesn't map onto anything they've defined. You end up hand-writing spans around ASR and diarization anyway, which works — OTel doesn't care what you name a span — but you're building the per-stage visibility yourself. Nobody sells it.



Corn

So the intended pattern, if a team asked us to state it plainly.



Herman

Compose with KFP or Azure components or Step Functions or Ray. Package with inference pipelines or batch pipeline deployments or Cloud Run. Instrument with OTel spans you write with your own hands. Per-stage visibility is something you build, not something you buy. Accept that up front and the architecture stops surprising you.



Corn

The Google-side counter-example is the one I keep thinking about, because it's a team that refused to flatten. Tell it.



Herman

A team that needed transcription with diarization and accurate timestamps looked at what Vertex offered and concluded no single speech-to-text service met all three needs. So they routed audio to Gemini in chunks — fifteen minutes or less — because around eighteen minutes the model started repeating itself and the timestamps drifted. Cost was about five minutes of processing per hour of audio.



Corn

They kept the graph but moved the walls. And chunk size is quietly an observability decision — smaller chunks, more seams, more failure points you can actually point at.



Herman

More boundaries means more spans, more logs, more places where a bad fifteen minutes is visible as a bad fifteen minutes rather than a bad hour somewhere inside a container.



Hilbert

Zoom H4n. Still four hundred odd. Mine sits on the shelf above the mixing desk, plugged into the feed that carries this very program.



Corn

The recorder's still going?



Hilbert

It runs whenever the desk runs. Has for years. The relevant thing, Herman, is the switch on the side. Stereo to mono. A physical toggle, not something buried in a menu. I flip it before I record anything, and I mean anything — the downmix happens at the source, before a single byte is written. You two spent twenty minutes on the glamorous stages. The downmix is where a transcript is won or lost. Voices sitting off-center in a stereo field that get averaged wrong, a channel half a volt down, and the ASR spends its whole budget guessing at a voice that was never there. And the ordering matters. Downmix before the silence trim, not after. Trim on stereo and then average, and you have moved the boundaries of the silence — what was a gap on the left can be speech on the right, and the cut lands somewhere different than it would have.



Corn

He's right about the ordering. Trim detects energy — which channel's energy?



Hilbert

That's the question nobody asks. I keep a notebook of every job where the downmix was done in the wrong order. Paper. Dates, one line each on what went wrong in the transcript. Corn, if you'd sat through the speech-to-text segment we did without once saying the word downmix, you'd have a notebook too. I can read from it, if there's time. There is a man I met on a ferry in nineteen ninety... a ferry, once, who told me he'd fixed a year of bad captions by moving one ffmpeg flag. I think about him monthly. He was the only person in the room who knew where the pipeline began.



Herman

How many entries?



Hilbert

Enough. The desk and the recorder and the notebook were all running while you two were being reasonable about black boxes. Somebody has to be particular about the first stage.



Herman

The man's been recording this show for years and we've never once checked what the downmix is doing to us.



Corn

I checked my levels once in the seventies. They were fine. I've been coasting since.



Herman

There's the forward question, then. If production keeps flattening the graph into one container — and the constraints say it will — then per-stage visibility has to move inside the container. You instrument the VAD and the diarizer as spans within one fat image, because the platform was never going to do it for you.



Corn

And the conventions to build that on don't exist yet. OTel's GenAI work is Development status and it models LLM calls and tool calls, not audio stages. Whoever builds per-stage audio observability is building on a floor that's still moving.



Herman

Which is either the warning or the opportunity, depending on your tolerance for shifting floors.



Corn

If you take one thing from this — and I mean one, I've counted the options — it's that composition, packaging, and observability are three decisions. Anyone selling you one tool for all three is selling you the flattening, quietly, as a feature.



Herman

And the flattening isn't always wrong. Sometimes one fat container is the right answer. But then own the fact that you've traded away which-stage-was-slow, because you have, and no dashboard will give it back.



Corn

The open question we'd leave you with: if the graph is what gives you the visibility, and the constraints keep collapsing the graph, does the whole industry end up hand-instrumenting inside containers for the next five years? Because right now, that's the pattern.



Herman

Herman Poppleberry here saying thanks to our producer, Hilbert Flumingtop, for keeping the desk — and apparently the downmix — in order all these years.



Corn

For more along these lines, there's episode seven, Building Custom ASR Tools; episode nine, Benchmarking Custom ASR Tools - Beyond The WER; and episode twenty, Architectural AI. This has been My Weird Prompts. If today's episode earned it, a review on Spotify helps other people find the show.



Herman

See you soon.

SOURCES

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

1. **AWS, SageMaker inference pipelines [primary]**
   https://docs.aws.amazon.com/sagemaker/latest/dg/inference-pipelines.html

2. **AWS, multi-container endpoint creation, `Serial`/`Direct` [primary]**
   https://docs.aws.amazon.com/sagemaker/latest/dg/multi-container-create.html

3. **AWS reference architecture, published 2022-03-28 [primary]**
   https://docs.aws.amazon.com/reference-architecture-diagrams/latest/multi-model-inference-orchestration/multi-model-inference-orchestration.html

4. **AWS, SageMaker Pipelines [primary]**
   https://docs.aws.amazon.com/sagemaker/latest/dg/pipelines.html

5. **AWS, 2025-10-01 [primary]**
   https://aws.amazon.com/about-aws/whats-new/2025/10/amazon-bedrock-data-automation-support-enhancing-transcription/

6. **AWS, BDA audio operations [primary]**
   https://docs.aws.amazon.com/bedrock/latest/userguide/audio-processing.html

7. **AWS, TranscriptConfiguration API [primary]**
   https://docs.aws.amazon.com/bedrock/latest/APIReference/API_data-automation_TranscriptConfiguration.html

8. **Google, Pipelines introduction [primary]**
   https://docs.cloud.google.com/gemini-enterprise-agent-platform/machine-learning/pipelines/introduction

9. **Google, Cloud Run AI use cases [primary]**
   https://docs.cloud.google.com/run/docs/ai/use-cases

10. **Microsoft Learn, pipeline components [primary]**
    https://learn.microsoft.com/en-us/azure/machine-learning/how-to-use-pipeline-component?view=azureml-api-2

11. **Microsoft Learn, registries [primary]**
    https://learn.microsoft.com/en-us/azure/machine-learning/how-to-share-models-pipelines-across-workspaces-with-registries?view=azureml-api-2

12. **Microsoft Learn, batch pipeline deployments [primary]**
    https://learn.microsoft.com/en-us/azure/machine-learning/how-to-use-batch-pipeline-deployments?view=azureml-api-2

13. **Ray, Deploy Compositions of Models [primary]**
    https://docs.ray.io/en/latest/serve/model_composition.html

14. **Ray, built-in Serve metrics [primary]**
    https://docs.ray.io/en/latest/serve/monitoring.html

15. **Ray Serve LLM observability [primary]**
    https://docs.ray.io/en/latest/serve/llm/user-guides/observability.html

16. **OTel GenAI spans, status Development [primary]**
    https://github.com/open-telemetry/semantic-conventions-genai/blob/main/docs/gen-ai/gen-ai-spans.md

17. **OTel GenAI metrics [primary]**
https://github.com/open-telemetry/semantic-conventions-genai/blob/main/docs/gen-ai/gen-ai-metrics.md

18. **AWS ML Blog, 15-container cap, no GPU**
https://aws.amazon.com/blogs/machine-learning/deploy-multiple-serving-containers-on-a-single-instance-using-amazon-sagemaker-multi-container-endpoints/

19. **AWS ML Blog, WhisperX DLC, real-time vs async, AMI pin**
https://aws.amazon.com/blogs/machine-learning/speaker-labeled-transcription-with-whisperx-on-sagemaker-ai/

20. **AWS ML Blog, ZOO Digital WhisperX on async inference**
https://aws.amazon.com/blogs/machine-learning/streamline-diarization-using-ai-as-an-assistive-technology-zoo-digitals-story/

21. **Charles Sieg, orchestrator comparison**
https://charlessieg.com/articles/sagemaker-pipelines-architecture-deep-dive.html

22. **Chaos Gears**
https://chaosgears.com/blog/machine-learning-pipelines-at-aws-which-service-should-i-use

23. **AWS, BDA product page**
https://aws.amazon.com/bedrock/bda/

24. **Google Cloud Blog, ML Metadata lineage and step logs**
https://cloud.google.com/blog/topics/developers-practitioners/orchestrating-pytorch-ml-workflows-vertex-ai-pipelines

25. **Xebia, updated 2026-01-28, caching caveats**
https://xebia.com/blog/vertex-ai-pipelines-catching-cache-complexity/