Episode Book

EPISODE #6498

Why You Must Read AI Traces Before Writing Evals

PUBLISHED	8 November 2026 at 14:39 UTC
DURATION	25:54
LISTEN	myweirdprompts.com/episode/error-analysis-llm-evals-12783/
HOSTS	Corn, Herman, Desmond, Hilbert
SCRIPT MODEL	z-ai/glm-5.3
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.26
SOURCES	8 cited by the research (listed at the end)
BOOK GENERATED	9 November 2026 at 08:09 UTC

SYNOPSIS

Standard advice suggests picking automated benchmarks and metrics first, but practitioners argue that effective AI evaluation requires reading raw interaction traces by hand. Discover how grounded theory methods like open and axial coding expose criteria drift and application-specific flaws before tools are deployed. The discussion covers practical trace sample sizes, how to spot upstream root causes, and why generic out-of-the-box eval suites often mislead engineering teams.

TRANSCRIPT



Corn

Thirty annotations. That's the number where Herman stops saying "we should look at the data" and starts actually looking at the data.



Herman

That's not fair. I look at data. I look at tons of data. The problem is I look at it the way a doctor reads a chart, which is to say I skim for the abnormal value and ignore everything that's politely normal.



Corn

Which is roughly where Daniel's prompt comes in. He's been reading the practitioners, Hamel Husain and Shreya Shankar mostly, and their argument is that the standard advice on evaluating AI systems has the order backwards. Everyone starts with metrics and benchmarks. The growing counterargument says the first move is reading a few hundred real outputs by hand, writing down what went wrong in plain language, grouping those notes into a failure taxonomy, and only then deciding what to measure. He wants to know what the disciplined version actually looks like, how many traces is enough, when you stop, why teams skip it, and how it changes the evals you end up building.



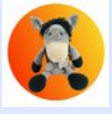
Herman

And the sting is that the method is sitting right there, described in public, updated this September, and teams still don't do it.



Corn

If it's that simple and that cheap, it should be the default. That's the question worth starting with.



Herman

The core claim, from Husain and Shreya Shankar, is that error analysis is the most important activity in evals, full stop. Not because reading traces is spiritually enriching, but because error analysis decides what evals to write in the first place. It's the step that identifies failure modes unique to your application and your data. Their stronger line is that the tools-first mindset is the most common mistake in AI development.



Corn

Which is a combative thing to say about an industry whose trade shows are mostly tools.



Herman

Right, and I want to be precise about the lineage, because it matters for whether you take this seriously. Open coding and axial coding are terms of art from grounded theory, which is qualitative social science research going back decades. Error analysis itself is an old machine-learning technique. Hamel illustrates it with an Andrew Ng video that's around eight years old. So Husain's own position is, quote, we didn't invent error analysis, we don't actually want to invent things. The novelty is only the adaptation to LLM applications, where the failure surface is effectively infinite and you cannot enumerate it up front.



Corn

The politest way possible of saying there's nothing to sell.



Herman

There's a research result underneath it too, which I think is the strongest single argument. Shankar and co-authors published a paper called Who Validates the Validators, at UIST in twenty twenty-four, and they named a phenomenon called criteria drift. The finding is that users need criteria to grade outputs, but grading outputs helps users define criteria. Some criteria are dependent on the specific outputs you actually observe, not on criteria you can define a priori.



Corn

So the rubric you'd write before reading anything is guaranteed incomplete.



Herman

Not just incomplete. Wrong in a way you can't detect until you've read the traces. Arize's annotation docs compress the whole thing into one line I like a lot: you can't automate a check you haven't defined.



Corn

That's the whole argument in nine words.



Herman

It is. And the craft question, what does disciplined error analysis actually look like, has a surprisingly concrete answer. It's a four-step loop. Step one, you create a dataset of representative traces of real user interactions. If you don't have data yet, you can generate synthetic data just to get moving. Step two is open coding, where a human, ideally what they call a benevolent dictator, reads traces and journals plain-language notes about anything that's wrong. Step three is axial coding, where you group those notes into a failure taxonomy and count how many failures fall in each category. Step four is iterative refinement, where an agent clusters the data and surfaces likely instances of the failures you described, you accept or reject its suggestions, and you keep going until you hit what qualitative researchers call theoretical saturation.



Corn

Meaning what, exactly?



Herman

Meaning new reviews stop revealing new weak spots, or stop changing the existing ones. That's the stopping condition. Not a number.



Corn

Hold on. Open coding. A benevolent dictator reading transcripts and journaling notes. Herman, I do that every time I read the group chat.



Herman

The difference is your notes say things like "Ezra sent the same gif twice" and theirs have to be categorizable by a stranger.



Corn

That's the discipline, isn't it. The note has to survive not being you.



Herman

And it's where most attempts fall apart. The guidance is to write the first thing you see that's wrong, the most upstream error, and then stop. Because upstream errors cascade, everything downstream is contaminated by them, and if you chase every error in a single trace the process becomes infinitely expensive. One note per trace. But the note has to be specific. Their example of a usable one is did not confirm call transfer with user. The example of a useless one is janky. Janky feels like information while you're writing it, and then a week later neither a human nor an LLM can categorize it without re-reading the whole trace, at which point why did you write the note.



Corn

Janky is a review. It's not a diagnosis.



Herman

Beautiful distinction, actually. And the upstream rule has a second payoff nobody mentions at first. If you fix the root cause, several downstream problems resolve without you ever building evaluators for them. That's Tian Pan's point about multi-step systems. The downstream errors are often one root cause wearing five costumes.



Corn

Now the axial step. You said that's the one they call most important.



Herman

Axial coding, yes. You export your notes to a CSV and ask an LLM to group the open codes into axial codes, meaning named failure-mode categories. There's a lovely hack in there: use the literal terms open codes and axial codes in the prompt. Because those are established concepts, the model shortcuts straight to the right behavior instead of inventing its own clustering scheme.



Corn

The jargon is doing the work of three paragraphs of instruction.



Herman

But then there are two corrections you have to make yourself. First, add a none-of-the-above category, so the LLM can flag notes that don't fit. When notes land in none of the above, that's the signal your taxonomy is incomplete and you need new categories. Second, the LLM's first-pass categories will be too generic to act on. Capability limitations is the canonical bad one. You rename those into specific, actionable breaking points yourself. And then, because you changed the categories after the LLM first proposed them, you have to re-label every single note against the refined list. You can't skip the re-mapping, because the old labels point at categories that no longer exist.



Corn

So the LLM proposes, you dispose, and then it re-sorts. A division of labor that keeps breaking and getting rebuilt.



Herman

There's a good illustration of why the human has to intervene. The automation FAQ gives the example of an LLM grouping app crashes after login and login takes too long under a single category called login issues. One is a stability problem, one is a performance problem, they need completely different fixes. Without your intervention you'd never notice you'd merged two diseases into one diagnosis.



Corn

That's an LLM that would sort a flat tire and an empty tank under car won't go.



Herman

The other example is subtler. Suppose users keep mentioning lag in feedback. The LLM lumps those under generic performance issues instead of creating a latency category. You miss that there's a recurring complaint about response times, and you fail to prioritize a fix. The users told you. The clustering hid it.



Corn

Which raises the question every team asks first. How many of these do I actually have to read?



Herman

This is my favorite part of the whole literature, because the numbers disagree, and the disagreement is more interesting than any single answer. The hamel.dev FAQ says at least thirty traces open-coded by hand before you even look at agent suggestions. The automation FAQ says thirty to fifty before you let an LLM do first-pass axial coding. The same FAQ suggests a working pool of roughly a hundred diverse traces as a guardrail for the human-agent loop. The Lenny's framework breakdown says that in practice people find fifteen to sixty is often enough, that the number is usually far lower than a hundred.



Corn

And Anthropic says something different again.



Herman

Anthropic's guidance on evals for agents says twenty to fifty simple tasks drawn from real failures is a great start, and their whole pitch is that teams delay building evals because they think they need hundreds of tasks. Notice the unit, though. Tasks, not traces.



Corn

I was about to say. A trace is a full conversation with tool calls and intermediate steps. A task is one scenario. Twenty tasks could be twenty traces or it could be a hundred and forty.



Herman

And nobody reconciles the units. That's the honest state of it. The only stopping rule that's consistent across everyone is theoretical saturation. Stop when new traces stop teaching you anything. The numbers are mental targets to unblock yourself, not thresholds. Use a hundred as a way to start, not as a quota to fill.



Corn

The quota mindset is the enemy anyway. If you've read ninety and you're learning nothing, number ninety-one is a ritual.



Herman

There's also a maintenance angle that I think gets undersold. Once the categories exist, re-running the whole loop weekly takes about thirty minutes. The expensive part is one-time. People hear read a few hundred outputs by hand and picture a permanent cost, and it isn't.



Corn

Before we get into why teams dodge this, we're joined by our resident money man, Desmond.

D

Desmond

Desmond's Money Tip. Save the last inch of wax in every candle jar. Over a year, at roughly one candle a fortnight, that is twenty six inches, which works out to about one and a half free candles, worth perhaps four pounds fifty.

D

Desmond

I keep those stubs in a biscuit tin under the sink, labelled Emergency. In eleven years the emergency has not come. The tin now has a smell.

D

Desmond

Money is just the future arriving slowly and disappointing you. Goodbye.



Corn

Twenty-six inches of emergency wax. I'm not going to lie, Herman, I'm calculating my own jar.



Herman

You are not starting a biscuit tin. You're a sloth with a hoarding instinct and I refuse to enable it. Where were we. Why teams skip it.



Corn

The tools-first reflex. That's the one Husain calls the most common mistake.



Herman

And he describes the pitfall in a really concrete way. You plug a dashboard into your system, you get your panel of metrics, and you have this illusion of, oh, I checked the box of doing evals, I am monitoring my system. But in reality you're not monitoring anything. You just wasted a whole bunch of time. You don't know what your failures are or what the most important things to focus on are.



Corn

The dashboard is a receipt. It proves you purchased monitoring, not that it occurred.



Herman

There's a vendor pressure behind it too. Eugene Yan's point is that out-of-the-box eval suites push generic criteria that are easy to plug in precisely because they ignore the idiosyncrasies of your domain and data. Generic metrics aren't a neutral default. They're counterproductive, and most platforms nudge you toward them anyway, because a suite that assumes nothing about your domain can be sold to everyone in your domain.



Corn

The generic criteria are easier to sell exactly to the degree they know less about you.



Herman

Then there are the simpler explanations. Nobody was ever taught the technique. And the technique feels too simple to be a technique. You read a trace and you write one sentence about what went wrong. That is it. Professionals especially resist this, because



Herman (continued)

we've all been trained that anything this cheap must be missing something.



Corn

You spent years in medicine where the cheap diagnostic was always the one the senior consultant warned you about.



Herman

Counterexample, actually. The cheap diagnostic in medicine is asking the parent to describe the symptom in their own words. The expensive one is the MRI. The senior consultants wanted the cheap one first. I read an error log the way I once listened to a worried parent describe symptoms. What is said is evidence, what is left out is usually the diagnosis.



Corn

That's the open coding instinct arriving from a completely different field.



Herman

Tian Pan frames the whole landscape as two snags, and I think the frame is worth keeping intact. The first is building no evals at all and shipping features on vibes. The second is building elaborate evaluation infrastructure before you understand what you're actually trying to measure. Both are expensive mistakes, and the second one masquerades as diligence. That's what makes it worse. Nobody congratulates themselves for shipping on vibes, but plenty of people get promoted off the back of an eval framework.



Corn

And the benchmark-substitute angle. That's the comfortable middle option.



Herman

Top models now exceed ninety percent on HumanEval and GSM8K. Those benchmarks are saturated, they've largely stopped being useful for model selection. And the deeper problem is contamination. Models trained on massive web corpora have often seen the test questions during training, so scores reflect memorization as much as capability. Some organizations now refresh benchmark datapoints monthly just to counter the leakage.



Corn

A test the students have already sat is not a test.



Herman

We actually made this argument once before, in the ASR benchmarking episode. A single aggregate score, WER there, benchmark percentages here, hides the ways it goes wrong that actually matter, and you have to look at individual outputs to know what's wrong. Same shape, different field.



Corn

Okay. So a team skips all that, reads its traces, does the coding, gets the counts. What actually changes?



Herman

This is the payoff, and it's counterintuitive. They end up building fewer evals, not more.



Corn

Fewer. In an industry allergic to the word.



Herman

Fewer and better aimed. The counts route each failure to the right kind of check. Frequent enough to deserve engineering effort. Mechanical enough for a cheap code assertion, like exact match or regex or schema validation. Subjective enough to need a binary LLM



Herman (continued)

judge. And crucially, the counts stop you building the generic helpfulness metric that measures nothing your users care about. Eugene Yan separates wasted criteria into two kinds. Irrelevant criteria, meaning generic metrics or very low probability defects like grammar and spelling. And unrealistic criteria, meaning things the technology just isn't ready for. Either way you're squandering effort that would be better spent on actual defects occurring at moderate frequency.



Corn

Grammar checks on a system whose real problem is that it lies about dates.



Herman

Which is almost a literal description of the Nurture Boss case, and I want to tell it because it's the cleanest proof in the whole literature. Nurture Boss built a simple viewer for their own conversations, read them, and found a sixty-six percent failure rate on relative dates. Their date handling went from thirty-three percent success to ninety-five percent after categorizing the failure, writing tests, and measuring.



Corn

From a third to ninety-five. One category.



Herman

One category, found by reading. No framework purchase, no judge calibration. They logged the traces, looked at them, and the biggest defect introduced itself.



Corn

The architecture point, though. You said the counts split the evals into tiers.



Herman

Tian Pan's two-tier architecture. Deterministic checks, meaning exact-match, regex, schema validation, run like unit tests on every inference. Statistical evals, meaning LLM judges and human review panels, are sampled and monitored periodically. The rule is to keep them separate and track them differently, because mixing them in a single pipeline creates confusion. A regex that fails is a bug. A judge that scores low is a signal. Those are different conversations and they shouldn't share a dashboard.



Corn

How many metrics before a pipeline goes muddy?



Herman

Five. Tian Pan reports a five-metric ceiling per pipeline, beyond which signal degrades.



Corn

Five. I've got a browser tab open right now with a product page bragging about forty.



Herman

And the target pass rate, which is the number that surprises everyone. Around seventy percent, not a hundred. A hundred percent pass rate signals that your evals aren't testing hard enough. Seventy with clear failure categories means you've built evaluators that actually discriminate.



Corn

That's the part I'd struggle to sell to management. Congratulations, our system fails three times in ten, and that's success.



Herman

It's success because of what the failures are doing. They're informative. If nothing fails, your eval suite can't tell a good change from a bad one, which means it's decoration. This connects back to the root-cause point too. Only build an LLM-as-judge evaluator for



Herman (continued)

persistent failure categories you'll iterate on repeatedly. One-off checks are better done with direct human review. Rule-based checks are cheaper and more reliable for deterministic requirements, so use those first. The judge is the most expensive instrument on the bench. Don't use it to check spelling.



Corn

There's a tension in all this I want to name. The line everyone repeats is, use LLMs to scale what you've learned, not to avoid looking at data. Never skip the open coding or delegate it. But the same FAQ that says that also tells you the magic words open codes and axial codes shortcut the model into doing half the taxonomy work.



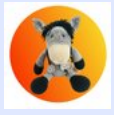
Herman

It's a real tension. The boundary is drawn at the initial reading. The LLM may cluster, count, map annotations to weak spots, suggest prompt improvements. It may not do the first read, and the reason given is specific. It lacks the product context to spot bad product smell.



Corn

Say more about product smell. That's the piece I find non-automatable.



Herman

The example the sources use is a technically correct answer, like we don't have that, have a nice day, delivered to a sales lead. Not a single word is wrong. The grammar is fine, the fact is accurate, the tone is appropriate. Any automated checker passes it. And it's a catastrophic product failure, because you just politely closed the door on a potential customer. Recognizing that requires knowing what your product is for. The Lenny's framework is blunt about the risk in the other direction, too. An LLM without product context rubber-stamps bad traces as looks good.



Corn

So the first thirty reads are where the domain expert earns their title. The machine can only sort what a human could have sorted, slower.



Hilbert

The first one is not the first one, Herman.



Hilbert

You've got it right in outline but the ordering's wrong in practice. I kept the fault chits from the bindery. Forty-two of them, they're in the drawer under the tea things. We had a Heidelberg that kept logging sheet jams, first error on every chit, jam at the feed, everyone wrote jam at the feed. I didn't write that. I wrote the timestamp, because on the seventeenth chit the same second appeared twice, which meant the sensor had missed a sheet two steps before the jam and logged the next one in its place. The jam was the symptom. Nobody coded the missed sheet because it looked fine on its face.



Corn

You were open coding a printing press before the term existed.



Hilbert

We called it the first cough. Same idea. The man at the parts counter, Reg, he'd ask for the chit before he'd sell you anything, and if you said jam at the feed he'd send you home. Two years of that before I trusted the timestamps more than the fault light. My claws aren't made for those little keys, so I made Reg's lad type up the count for me. He did it in an afternoon. Decent lad. He was the only one there who read anything twice.



Herman

The duplicated timestamp is exactly the detail that makes the upstream rule work. The visible error is downstream, the evidence for the real one is a repeated field nobody was looking at.



Hilbert

It's usually in a field nobody was looking at. That's where the first cough lives.



Corn

So if you take one thing from today, it's the nine-word version. You can't automate a check you haven't defined. Everything else, the taxonomies, the judges, the five-metric ceiling, the seventy percent pass rate, is downstream of somebody reading thirty traces and writing one honest sentence about each.



Herman

And the sharpening on that: the sentence has to be one a stranger could sort. Did not confirm call transfer with user. Never janky.



Corn

There's also the open question we never resolved, which I like leaving open. There is no controlled study comparing teams that do error analysis first against teams that don't. The evidence is case studies and practitioner testimony, and the sixty to eighty percent of development time figure is two practitioners describing their own weeks. Repeated everywhere, verified nowhere.



Herman

Even the strongest disagreement in the literature is about the number, not the method. Twenty to fifty tasks, thirty traces, a hundred as a guardrail. Tasks and traces may not even be the same unit.



Corn

Which leaves the tension that started the episode sitting exactly where it should. The method is deliberately unoriginal, which undercuts every vendor selling a new framework, and its output is fewer evals at a lower pass rate, which is a rebuke of the coverage instinct. If the highest-ROI activity in AI development is the one that feels too simple to be a technique, the question isn't whether to do it. It's why everyone's buying dashboards instead.



Herman

Thanks to Hilbert Flumingtop for producing this one, as ever. And the forty-two chits in the drawer.



Corn

That's been My Weird Prompts. If today changed how you think about evals, a review helps other people find the show. You can find everything at my weird prompts dot com.



Herman

We'll be back soon. See you then.



Corn

See you.

SOURCES

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

1. **Hamel's Blog (Hamel Husain & Shreya Shankar), published 2025-06-27, updated 2026-09-01; retrieved 2026-11-08. [primary]**
   https://hamel.dev/blog/posts/evals-faq/why-is-error-analysis-so-important-in-llm-evals-and-how-is-it-performed.html
2. **Hamel's Blog, published 2025-07-24; retrieved 2026-11-08. [primary]**
   https://hamel.dev/blog/posts/evals-faq/what-parts-of-evals-can-be-automated-with-llms.html
3. **Anthropic, Demystifying evals for AI agents; retrieved 2026-11-08. [primary]**
   https://www.anthropic.com/research/demystifying-evals-for-ai-agents
4. **Lenny's Podcast framework breakdown of the Husain/Shankar episode Why AI evals are the hottest new skill for product builders; retrieved 2026-11-08.**
   https://lennyspodcast.podplaybooks.com/frameworks/error-analysis-open-axial-count-husain
5. **AgentEngineering.org, Error Analysis: The First Eval You Should Run; retrieved 2026-11-08.**
   https://agentengineering.org/articles/error-analysis-the-first-eval-you-should-run/
6. **Tian Pan, LLM Evals: What Actually Works and What Wastes Your Time, 2026-02-20; retrieved 2026-11-08.**
   https://tianpan.co/blog/2026/02/20/llm-evals-practitioners-guide
7. **Chain of Thought podcast, episode 37, Mindset Over Metrics with Hamel Husain; retrieved 2026-11-08.**
   https://chainofthought.show/podcast/37-mindset-over-metrics-how-to-approach-ai-engineering-hamel-husain/
8. **Xu et al., Diagnosing Failures in Large Language Models' Answers: Integrating Error Attribution into Evaluation Framework, submitted 2025-07-11; retrieved 2026-11-08.**
   https://arxiv.org/abs/2507.08459v1