APPENDIX

Trace Capture, Classification, and Custom Review Interfaces

How Production Traces Feed Error Analysis,
and How Error Analysis Feeds the TSR

Supplement to: AI Evaluation in the SDLC
Joint Brief from the Director of Engineering and the Director of Product

Why Trace Capture Is Foundational

The main brief describes the error analysis → fix → automate cycle as the core improvement loop for AI features. What it does not detail is **where the data for error analysis comes from.** The answer is traces.

A trace is the complete record of everything that happened during a single AI interaction: the user's input, the system prompt, any retrieved context, every tool call, the model's reasoning chain, intermediate steps, and the final output. Hamel Husain and Shreya Shankar—who have trained over 2,000 engineers and product managers at companies including OpenAI, Anthropic, Google, and Meta—describe traces as the essential raw material for AI evaluation. As Shankar puts it: "This is what your AI agents are actually doing out there in production, and that's why looking at the traces is so important."

Without trace capture, error analysis is guesswork. With it, error analysis is systematic and evidence-based. This appendix describes how trace capture fits into our SDLC governance framework, how to think about the data classification and access implications, and how to build the custom review interfaces that make trace analysis fast, accurate, and directly connected to the TSR.

> *Traces are to AI features what logs are to traditional software: the forensic record that lets you understand what actually happened, not what you thought would happen. The difference is that traces carry far more sensitive content and require more deliberate governance.*

Trace Capture by Risk Tier

Not every AI feature requires the same level of trace capture. Consistent with the risk-tiered TSR structure in the main brief, trace capture and monitoring should be calibrated to the risk level of the application.

Risk Tier	Trace Capture Scope	Retention & Monitoring	Review Cadence
Tier 1: Low	Capture model inputs and outputs. Minimal metadata (timestamp, user session ID, model version, latency).	Retain for 30 days. Sample-based monitoring: spot-check 10-20 traces weekly.	Monthly review of sampled traces. Error analysis only when user complaints exceed threshold.
Tier 2: Medium	Full trace capture: inputs, outputs, retrieved context, tool calls, system prompts. Include user feedback signals.	Retain for 90 days minimum. Automated quality monitoring with alert thresholds. Weekly output sampling.	Weekly error analysis of flagged/sampled traces. Full review with each production modification. Automated clustering of failure patterns.
Tier 3: High	Comprehensive trace capture with complete reasoning chains, all intermediate steps, confidence scores, retrieval scores, and guardrail evaluation results. Include audit metadata.	Retain for 1 year minimum or per regulatory requirement. Continuous monitoring with real-time alerting. Full observability dashboard.	Continuous error analysis. Formal review cycle (bi-weekly minimum). Traces directly referenced in TSR evidence sections. Audit-ready export capability.

The key principle: **trace capture depth should match the consequence of failure.** An internal productivity tool where a bad AI output costs someone five minutes of annoyance does not need the same trace infrastructure as a customer-facing financial decision support system where a bad output could create regulatory exposure.

Minimum Viable Trace Capture (All Tiers)

Regardless of tier, every AI feature in production must capture at minimum: the user input, the model output, the model version identifier, a timestamp, response latency, and any user feedback signals (thumbs up/down, flagged outputs, corrections). This minimum viable set is what makes even Tier 1 features debuggable. As Husain emphasizes: “If you’re already paying for Datadog or another APM tool, use that. The key is logging traces somewhere you can review them.” The specific platform matters far less than the practice of capturing traces at all.

Data Classification and Access Governance

Traces contain some of the most sensitive data in your AI system. A single trace may include: the user's original query (which may contain PII, business-sensitive information, or protected data), retrieved context from internal systems, the full system prompt (which represents proprietary business logic), and the model's output (which may contain generated PII or reflect confidential decision-making patterns). This means trace data requires its own data classification and access governance.

Trace Data Classification Framework

Trace Component	Likely Classification	Access Considerations	Handling Requirements
User inputs	Confidential to Restricted. May contain PII, financial data, health information, or business secrets.	Need-to-know basis. Reviewers must have appropriate data access authorization.	Mask or redact PII in review interfaces where full content is not required for evaluation. Log access for audit trail.
System prompts	Internal / Proprietary. Contains business logic, guardrails, and competitive differentiation.	Restrict to engineering and authorized evaluators. Exclude from exports to external parties.	Version-controlled separately. Treat as source code for access control purposes.
Retrieved context	Inherits classification of source data. May include customer records, internal documents, or regulated data.	Same access controls as the source systems. Do not create an uncontrolled copy of restricted data.	Apply source system access policies. Consider whether the trace review interface needs full context or summaries.
Model outputs	Confidential. May contain generated PII, synthesized business intelligence, or recommendations.	Available to authorized evaluators and the trace review team.	Evaluate whether outputs can be de-identified for broader analysis. Flag outputs that may have generated PII.
Reasoning chains / tool calls	Internal. Contains implementation details and integration patterns.	Available to engineering and technical evaluators.	Less sensitive than user data, but reveals system architecture. Restrict external sharing.
Metadata (latency, model version, timestamps)	Internal. Low sensitivity.	Broadly available for dashboarding and performance monitoring.	Standard operational data handling. No special restrictions typically needed.

Access Control Principles

Who should have access to traces? This depends on the role and the tier. The guiding principle is: give evaluators the minimum data necessary to assess quality, and log all access for audit purposes.

Role	Tier 1 Access	Tier 2 Access	Tier 3 Access
Developer / AI Builder	Full trace access for features they own	Full trace access for features they own	Full trace access with audit logging; access review quarterly
Tech Lead / Architect	Full trace access across team features	Full trace access across team features	Full trace access with audit logging; signoff on access grants
Product Owner / BA	Output-only view (input + output, no internals)	Output + retrieved context view; PII-masked where possible	Full trace access with PII masking; audit-logged; compliance training required
QA / SDET	Full trace access for test environments; sampled production traces	Full trace access for test and production	Full trace access with audit logging
Compliance / Governance	Summary metrics and dashboards only	Sampled trace access for audit; full access on request	Full trace access; audit export capability; retention enforcement
Business Stakeholders	Aggregated metrics only (no individual traces)	Aggregated metrics; anonymized example traces on request	Aggregated metrics; de-identified traces for formal review; full access only via compliance channel

> *The default should be that product managers and business analysts can see AI inputs and outputs—which is what they need for error analysis—without necessarily seeing the system prompt, reasoning chain, or raw retrieved context. When deeper access is needed for specific failure mode investigation, it should be granted and logged, not blocked.*

Custom Trace Review Interfaces

This is where trace capture translates into the error analysis that feeds the TSR. Husain and Shankar are emphatic on this point: **building a custom annotation tool is the single most impactful investment you can make for your AI evaluation workflow.** Teams with custom trace review interfaces iterate approximately ten times faster than teams using generic observability tools. The reason is friction removal: a generic tool shows you everything about a trace, while a custom interface shows you exactly what you need to evaluate quality for *your specific product.*

Why Build Your Own Instead of Using Off-the-Shelf Tools

Off-the-shelf observability platforms—Braintrust, LangSmith, Arize Phoenix, and others—handle the basics well: capturing traces, displaying them, and providing annotation queues. These are fine starting points, and Husain recommends beginning with one. But the Husain/Shankar methodology draws a clear distinction between *trace capture* (where off-the-shelf tools are sufficient) and *trace review for error analysis* (where custom interfaces are transformative).

The core argument is that you know what information matters for your product. A generic trace viewer shows system prompts, tool calls, token counts, and reasoning chains—all useful for debugging, but overwhelming for the domain expert doing error analysis. A custom interface for a real estate CRM assistant, for example, hides the system prompt by default, shows the relevant CRM record alongside the trace, surfaces the specific channel (voice, email, text, chatbot), and lets the reviewer annotate with a single click. A medical application's custom interface renders clinical context alongside the AI output so a physician can evaluate appropriateness without reading raw JSON. The pattern is the same: reduce friction, surface domain-relevant context, and make the reviewer's judgment as fast and accurate as possible.

What a Custom Trace Review Interface Contains

Based on the Husain/Shankar methodology and case studies from their course alumni, here are the elements that make a custom trace review interface effective:

1. Domain-Specific Rendering

Display the AI output the way the user experienced it. If the feature generates emails, render them as emails. If it generates summaries, show them as formatted summaries alongside the source document. If it's a conversational agent, show the conversation thread. The reviewer should never need to parse raw API responses to understand what the user saw.

2. Contextual Side-Panel

Show the information the reviewer needs to assess quality alongside the trace. For a document summarization feature, this means the source document. For a CRM assistant, the customer record. For a recommendation engine, the user's history. This contextual data is what transforms a trace from "did the model produce output?" to "did the model produce the right output for this user in this context?"

3. Binary Annotation (Pass/Fail)

Husain and Shankar are firm: use binary scoring (pass/fail), not Likert scales (1–5). Binary evaluations force clearer thinking, produce more consistent labels, and map directly to business decisions: either the output is acceptable or it isn't. This simplicity is what makes it possible for product managers and business analysts to participate in error analysis meaningfully—they don't need to calibrate a 5-point scale, they need to answer one question: "Is this good enough?"

4. Open-Ended Notes Field

When a reviewer marks a trace as "fail," they write a brief, open-ended note describing what went wrong. This is adapted from qualitative research methodologies—Husain calls it "journaling." These notes are the raw material for building the failure taxonomy. The note might be as simple as "Told user it would check on bathrooms but didn't do it" or "Used casual tone for a luxury client." Precision is important; verbosity is not.

5. Failure Mode Tagging

Once the team has reviewed enough traces (Husain recommends a minimum of 100) and the open-ended notes have been categorized into a failure taxonomy, the interface should support tagging traces with the identified failure modes. This is the transition from "open coding" (discovering what's going wrong) to "axial coding" (categorizing it systematically). Some teams automate the axial coding step using an LLM to cluster similar notes, then human-review the clusters.

6. Filtering and Sampling Controls

The interface should let reviewers filter by channel, feature, time range, user feedback signal (user-flagged issues), and failure mode tag. It should also support stratified sampling: reviewing a balanced cross-section of traces rather than only the most recent or most problematic. As the evaluation process matures, embedding-based clustering can surface groups of similar traces that might share an underlying failure pattern.

7. Keyboard Shortcuts for Speed

This is a practical detail that matters enormously. Pass/fail annotation and navigation between traces should be achievable without touching a mouse. Alumni from the Husain/Shankar course consistently report that keyboard shortcuts transform error analysis from a chore into a sustainable practice.

> *Husain emphasizes that these custom interfaces can be built in hours using AI-assisted development tools. They do not need to be polished products—they need to be functional tools that remove friction from the specific evaluation workflow for your specific product. Start simple. Iterate based on what slows your reviewers down.*

From Traces to TSR: The Connection

The custom trace review interface is where evaluation happens. The TSR is where evaluation is documented for governance. Here is how the information flows from one to the other:

Mapping: What the Trace Review Interface Produces → Where It Appears in the TSR

Trace Review Output	TSR Section	How It's Used
Pass/fail annotations on sampled traces	Section 3: Error Analysis Results	Quantifies the failure rate and distribution across failure modes. "12 of 47 flagged summaries contained fabricated clauses."
Open-ended notes on failures	Section 3: Error Analysis Results	Provides the qualitative detail behind each failure mode. Describes what went wrong in human language.
Failure taxonomy (categorized failure modes)	Section 2: Test Strategy + Section 3: Error Analysis Results	The taxonomy becomes the structure of the error analysis. Each category maps to a targeted fix.
Failure mode counts (pivot table)	Section 3: Error Analysis Results	The quantified failure distribution determines priority. High-count failure modes get addressed first.
Traces linked to specific failure modes	Section 4: Changes Made	Each prompt/architecture change references the specific failure mode it addresses, traceable back to example traces.
Pass/fail rates before and after fixes	Section 5: Automated Test Results	Demonstrates measurable improvement. "Failure rate dropped from 25% to 4% on the hallucination category."
Ongoing trace monitoring dashboards	Section 6: Production Monitoring Plan	Defines what will be monitored, with baselines established during trace review.
User feedback signals from	Section 1: Change Summary	Production user complaints and

production traces	(for modifications)	feedback that triggered the current modification cycle.

The Practical Workflow

Here is what this looks like in practice for a Tier 2 production modification—the same scenario from the main brief's example TSR:

WEEK 1: TRACE COLLECTION

The team identifies the trigger: user complaints about document summarization accuracy. They pull the relevant traces from the observability platform—in this case, 47 user-flagged traces plus a stratified sample of 153 additional traces (200 total). These are loaded into the custom trace review interface, which renders each summary alongside its source document.

WEEK 1-2: ERROR ANALYSIS IN THE REVIEW INTERFACE

A product manager and an engineer review traces together. The PM evaluates output quality (is this summary accurate and useful?); the engineer reviews the trace internals (what context was retrieved? what prompt produced this?). They annotate each trace as pass/fail and write open-ended notes for failures. After 50 traces, patterns emerge. They draft an initial failure taxonomy: Hallucination, Key Term Omission, Tone Mismatch. They complete the remaining traces using these categories.

WEEK 2: FAILURE TAXONOMY AND COUNTING

The team exports annotations and failure mode tags from the review interface. A simple pivot table produces the failure distribution: 12 hallucinations, 28 key term omissions, 7 tone mismatches. This quantified taxonomy becomes the structure of Section 3 of the TSR. Husain calls this the "money table"—it tells you exactly where to invest your improvement effort.

WEEK 2-3: FIX → TEST → DOCUMENT

Engineering implements targeted fixes for each failure mode. Each fix is tested against the traces that exhibited that failure mode (regression testing). New automated acceptance tests are written for each failure category. The TSR is authored collaboratively: the PM writes Sections 1 and 7 (change summary, recommendation); the engineer writes Sections 2, 4, and 5 (test strategy, changes, automated results); they co-author Sections 3 and 6 (error analysis, monitoring plan).

ONGOING: PRODUCTION MONITORING

After deployment, the trace capture continues. The monitoring plan defines thresholds: if the failure rate for any category exceeds the baseline observed during error analysis, an alert triggers review. The trace review interface is now the tool for ongoing monitoring—not just pre-deployment evaluation. New traces flagged by automated evaluators or user feedback are routed into the review queue for the next error analysis cycle.

Governance Without Burden: The Trace Angle

The concern with trace capture is obvious: it sounds like a lot of infrastructure, a lot of data management, and a lot of review work. Here is how we keep it proportional:

Trace Capture Infrastructure Is a One-Time Investment Per Feature

Instrumenting an AI feature for trace capture is done once during development. The observability platform handles storage and retrieval. The incremental cost of capturing traces for a new feature is minimal once the infrastructure exists. This is analogous to adding logging to a traditional application—you do it during development, and it pays dividends for the life of the feature.

The Custom Review Interface Is Reusable and Evolvable

Husain and Shankar's student testimonials consistently report that building a custom trace review interface takes hours, not weeks, using AI-assisted development tools. These interfaces are not disposable—they evolve with the product. New failure mode tags are added. New contextual data sources are wired in. The interface gets better over time, which means error analysis gets faster over time.

Sampling Keeps Review Volume Manageable

You do not review every trace. At Tier 1, you spot-check 10–20 traces weekly. At Tier 2, you use stratified sampling and automated filtering to review the traces most likely to reveal problems: user-flagged traces, traces where automated evaluators disagree with each other, and traces from underrepresented input categories. At Tier 3, you invest more, but the investment is guided by the risk profile, not by a mandate to review everything.

Automated Evaluators Reduce Manual Review Over Time

As the failure taxonomy matures and LLM-based judges are validated against human annotations, automated evaluators take over routine quality checks. The human reviewers focus on the traces that automated evaluators flag as uncertain or problematic. Husain describes this as a progression: "Manual labeling builds the foundation. Automated evaluators scale it. Human review stays in the loop for edge cases and calibration."

The TSR Captures the Synthesis, Not the Raw Data

The TSR does not reproduce every trace annotation. It captures the synthesized output: the failure taxonomy, the failure counts, the changes made in response, and the test results that validate those changes. The traces themselves remain in the observability platform, available for audit if needed, but the TSR is a concise narrative—not a data dump. This is the key to maintaining governance documentation without undue burden: the trace review interface does the heavy analytical lifting; the TSR captures the conclusions and evidence.

THE SYSTEM, COMPLETE

Trace capture gives you the data. The custom review interface gives you the insight. Error analysis gives you the failure taxonomy. The failure taxonomy gives you the structure for targeted fixes. The fixes give you automated tests. And the TSR captures all of it in a standardized, auditable artifact that communicates to the business: here is what we learned, here is what we fixed, here is how we know it's better, and here is how we'll know if it stops being better.

This is governed experimentation. Not governance as a brake—governance as the infrastructure that makes experimentation trustworthy, repeatable, and defensible. Build the review interface. Review the traces. Write the TSR. Ship with confidence.

The traces are the truth. The TSR is the story you tell about what you learned from the truth.

References

Husain, H. (2025). “Your AI Product Needs Evals.” hamel.dev. Case study of the Rechat AI assistant including custom trace review interface design and error analysis methodology.

Husain, H. & Shankar, S. (2026). “LLM Evals: Everything You Need to Know.” hamel.dev. Comprehensive FAQ covering trace review, custom annotation tools, error analysis, failure taxonomies, LLM judges, and evaluation workflow design.

Husain, H. & Shankar, S. (2025). “Selecting The Right AI Evals Tool.” hamel.dev. Panel assessment of Braintrust, LangSmith, and Arize Phoenix with evaluation criteria for observability and annotation platforms.

Shankar, S. et al. “Who Validates the Validators? Aligning LLM-Assisted Evaluation of LLM Outputs with Human Preferences.” ACM Conference on Human Factors in Computing Systems (CHI), 2024.

Husain, H. & Shankar, S. AI Evals for Engineers & PMs. Maven course. parlance-labs/evals. Trained 2,000+ engineers and PMs at OpenAI, Anthropic, Google, Meta, and other organizations.

Gupta, A. (2025). “How to Do AI Evals Step-by-Step with Real Production Data.” Transcript of tutorial with Husain and Shankar demonstrating the complete evaluation process with Nurture Boss production data.