GroundTruth (Part 1)

Build and Findings: A Hands-On Teardown of Lyzr Architect

Prepared by Abhishek Katkar. Part 1 covers what I built, how I tested it, and what I found. Part 2 covers the product analysis and recommendations that follow from these findings.

> **Approach and constraints.** Everything in this document was produced from the outside, as a first-time user, working within a limited free-credit allowance and using only publicly available information. The build, the diagnosis, the study of Lyzr's feature set, and the business model are all what a resourceful outsider could assemble with those constraints. With internal data and access, each hypothesis here becomes a measured decision rather than an informed estimate.
>
> **Live app:** https://support-sentinel-bold-edge-77s7.architect.space/
>
> **How to read this document set.** I built a small support-resolution app called GroundTruth on Lyzr Architect. GroundTruth is the *test vehicle*, not the subject. Everything in these documents - every finding, recommendation, and business case - is about **Architect and Studio, the Lyzr product**, evaluated by using GroundTruth to stress-test it. None of it is a proposal about GroundTruth itself. The one exception is Part 4, which invites you to test GroundTruth live purely as a way to experience Architect's behavior firsthand.

---

1. Executive summary

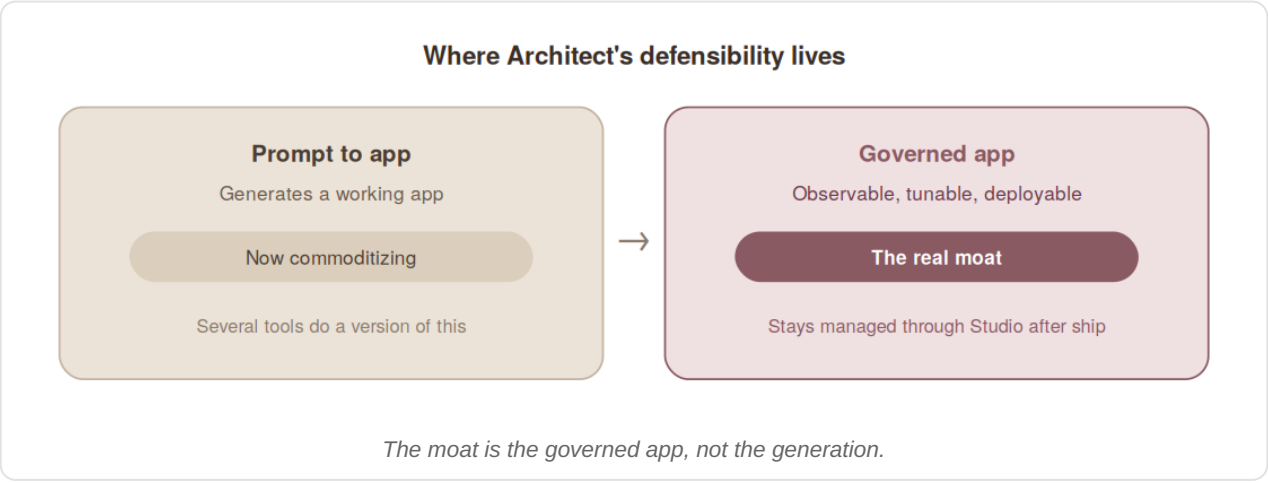
I built GroundTruth, a grounded customer-support resolution agent, entirely inside Lyzr Architect and Studio, and used it the way a real builder would: shipped it, broke it, diagnosed the failures, and fixed what I could.

- The app works. It classifies a ticket, retrieves from a knowledge base, drafts a reply, and verifies that draft against the retrieved sources before showing it. When it cannot ground an answer, it refuses and escalates to a human.
- The most valuable moment was a caught hallucination. On a SAML SSO ticket, retrieval returned a false high-confidence match, but the verification agent recognized the content did not support the answer and suppressed the draft. A retrieval score alone would have let that error through.
- The most valuable finding was about reliability. I traced an over-escalation problem to a drafting agent adding unsupported specificity, fixed it at the parameter level, and then found the residual variance is architectural, not fixable by prompt-tuning alone.

---

2. Context and objective

Prompt-to-app generation is commoditizing. Architect's defensible position is not that it generates an app, but that it generates a governed app that stays observable and tunable through Studio after it ships.



The moat is the governed app, not the generation.

For Lyzr's enterprise and regulated buyers, the real risk is not whether an agent can answer. It is whether it knows when not to. A confident but unsupported support reply is a compliance and trust liability. So I built an application whose defining behavior is refusal: it declines to answer when it cannot ground the response.

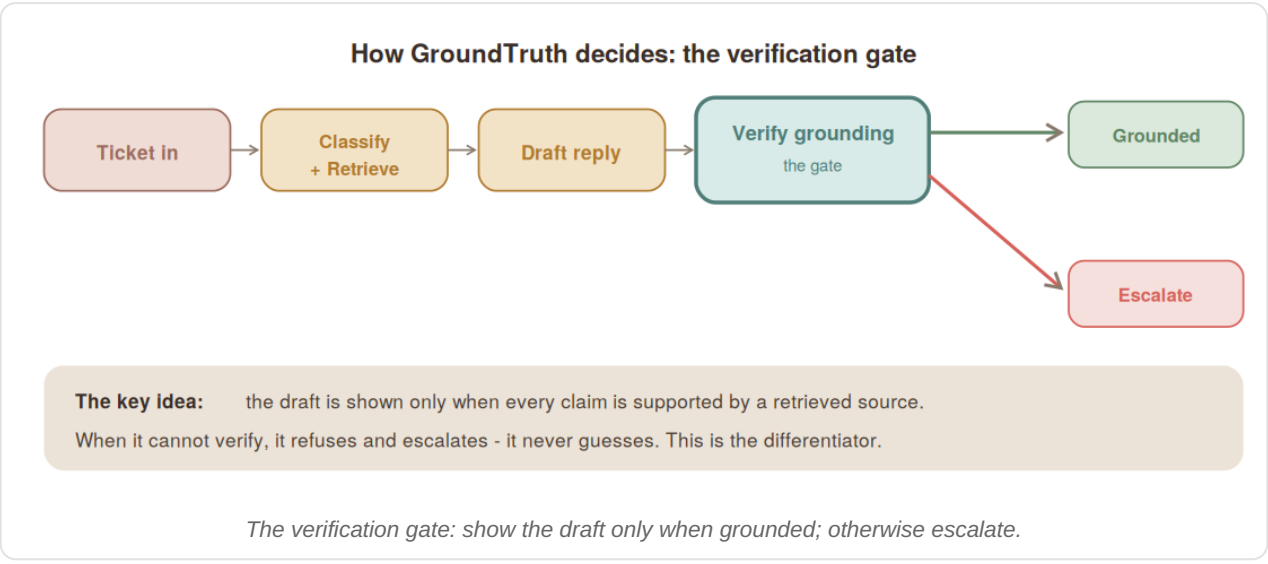
This was deliberate. It sits on Lyzr's real market (enterprise support automation), exercises Architect's genuine differentiators (multi-agent orchestration, retrieval, a governance layer) rather than a task a single model call could handle, and maps to grounding work I have done before. I anchored the knowledge base in a subscription SaaS support domain I can speak to with authority, and deliberately omitted two topics (SAML SSO and GDPR deletion) so the refusal path could be tested.

---

3. The product and architecture

3.1 What it does

An agent pastes an incoming ticket. GroundTruth classifies it, retrieves relevant articles, drafts a reply grounded strictly in those articles, and verifies every claim is supported before presenting anything. If grounded, it presents a send-ready reply behind a human approval gate. If not grounded, or if no relevant article exists, it suppresses the draft and escalates with a reason. The live app is available at https://support-sentinel-bold-edge-77s7.architect.space/



The verification gate: show the draft only when grounded; otherwise escalate.

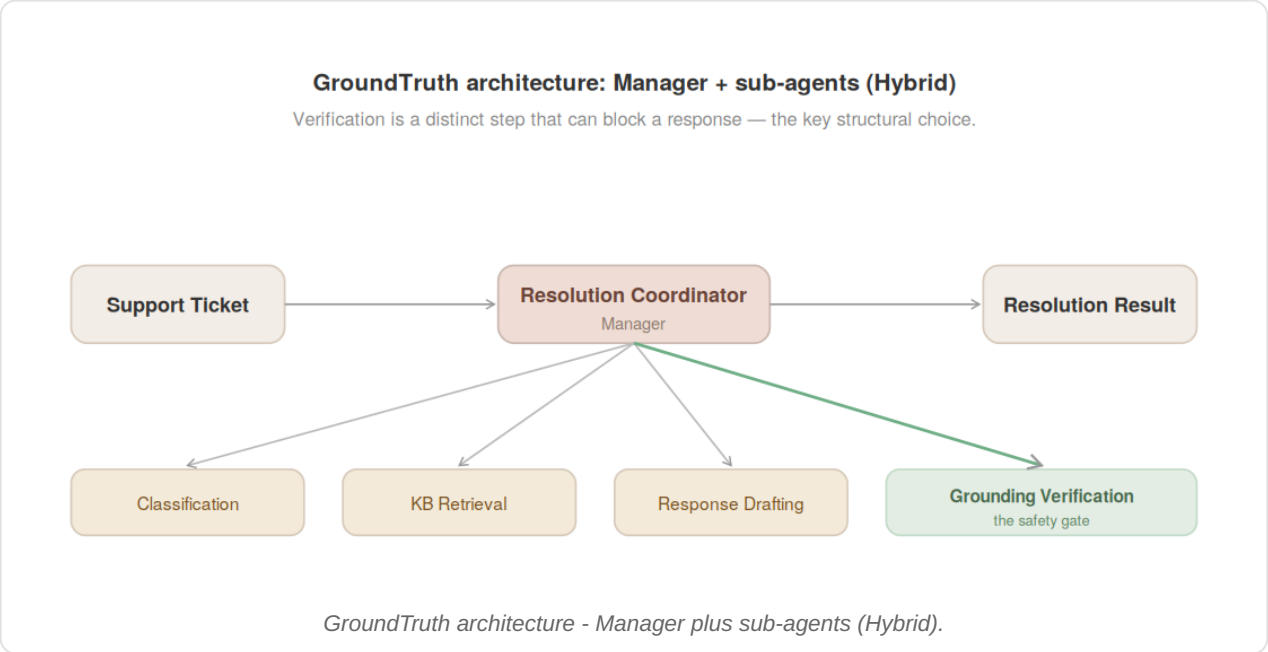
3.2 Architecture

GroundTruth uses a hybrid pattern Architect itself proposed during planning: a Manager plus sub-agents for the one-shot resolution pipeline, and a separate Independent agent for post-approval finalization. Architect reasoned to this on its own after I specified a human reviews the draft before finalizing, correctly identifying human-in-the-loop finalization as architecturally distinct from the resolution pipeline.

The six agents:

Agent	Type	Model	Temp	KB	Role
Resolution Coordinator	Manager	claude-sonnet-4-6	0.2	No	Orchestrates the pipeline; enforces the safety gate
Classification Agent	Sub	gpt-4o-mini	0.2	No	Category, priority, intent
KB Retrieval Agent	Sub	gpt-4o	0.1	Yes	Semantic search; returns articles with relevance scores
Response Drafting Agent	Sub	claude-sonnet-4-6	0.4 then 0.2	Yes	Drafts a reply grounded in retrieved articles
Grounding Verification Agent	Sub	claude-sonnet-4-6	0.1	Yes	Verifies every claim; returns Grounded, Not Grounded, or No Source
Response Finalizer	Independent	gpt-4o	0.3	No	Polishes the human-approved draft

Key design decisions. Retrieval and verification run at temperature 0.1 for near-deterministic behavior, appropriate for a checker. The verification agent is a distinct step that can block a response, not a rule folded into drafting. This separation is the single most important structural choice in the build.



GroundTruth architecture - Manager plus sub-agents (Hybrid).

Note on machinery: the verification behavior is an agent I designed into the flow via instructions, not the productized Hallucination Manager or Agent Eval modules, which sit in Lyzr's Enterprise tier. This distinction is maintained throughout.

3.3 The build experience

I built in Guided (Plan) mode rather than One Shot, to retain control over where the verification agent landed and to observe how Architect decomposes the problem. The flow was strong: a short intent, well-targeted

clarifying questions, a plan and UI mockup for approval, then a build with a visible self-correction loop. The step where it asks you to approve the plan and the UI before it builds is a genuine trust-builder and the best part of the experience.

---

4. Testing methodology

I tested with six tickets. Four target topics covered by the knowledge base (proration, password reset, cancellation, seat management) and should ground. Two target omitted topics (SAML SSO, GDPR deletion) and must escalate.

The win condition was never "always answer." It was "answer when grounded, refuse when not." I ran each grounding-path ticket multiple times, because a single run cannot reveal consistency, which turned out to be the central finding.

---

5. Results

5.1 An early finding: single-file knowledge bases inflate retrieval scores

My first knowledge base was one combined file. An off-topic SSO query still returned it at roughly 77 percent, because with one document in the store, semantic search has nothing to rank against. Splitting into seven single-article files separated retrieval cleanly: a genuine match scored about 88 percent while an off-topic query topped out near 72 percent.

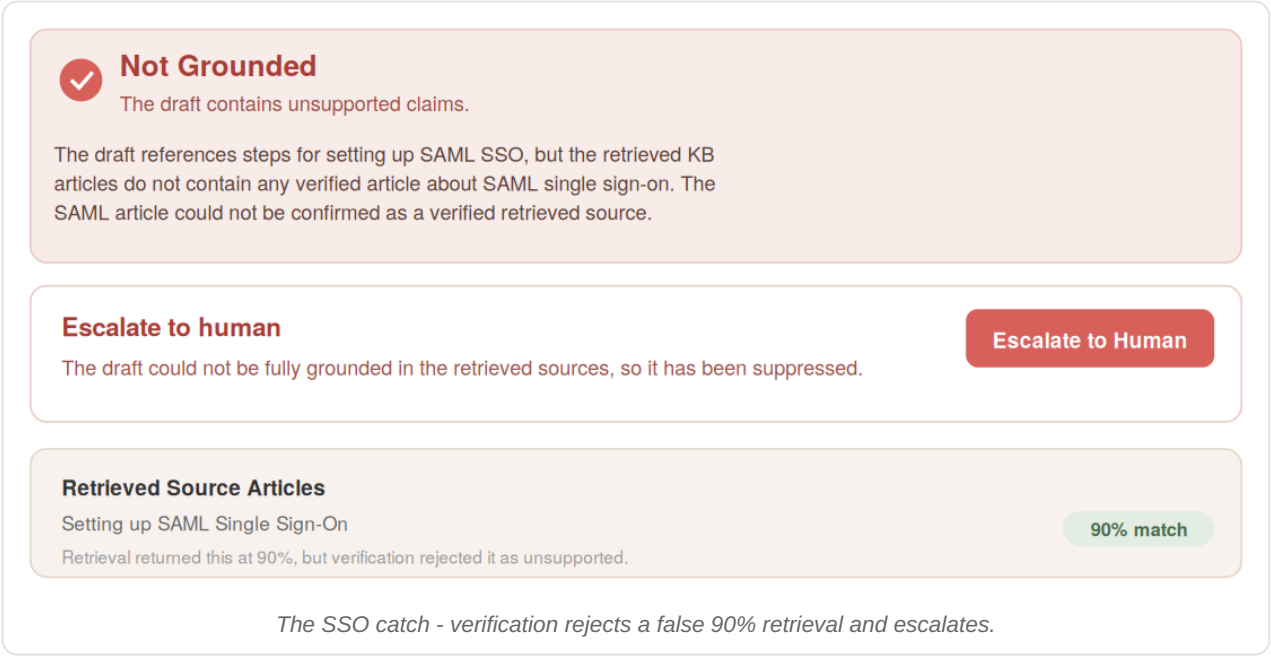
Tradeoff: finer chunking improves the no-match signal but increases artifacts to manage and can fragment articles that are genuinely one concept. Granularity is a design decision, not a default.

5.2 Six-ticket results (before the reliability fix)

- Ticket 1, proration: Grounded, correct.
- Ticket 2, password reset: Grounded, correct.
- Ticket 3, cancellation: Over-escalated. A valid article exists, but the app returned No Source. Reproduced across runs.
- Ticket 4, seat management: Over-escalated. Same pattern.
- Ticket 5, SAML SSO: Escalated correctly, and this is the standout. Retrieval returned a false roughly 90 percent match, but the verification agent recognized the content did not support a SAML answer, suppressed the draft, and escalated.
- Ticket 6, GDPR deletion: Escalated correctly. No Source.

Honest tally: two grounded, two correct escalations, two over-escalations.

Why the SSO result matters most: retrieval alone would have passed a 90 percent match to the drafter, producing a confident SAML answer built on unrelated content. The dedicated verification layer caught it. This is the concrete proof that grounding needs a distinct verification step, not a trust-the-score approach.



The SSO catch - verification rejects a false 90% retrieval and escalates.

6. Diagnostic deep-dive: over-escalation, root cause, fix, and its limits

This is the core of the analysis, written in the order I worked through it.

6.1 Symptom

Tickets 3 and 4 sometimes grounded and sometimes escalated with No Source, across identical inputs. My first hypothesis was a mistuned retrieval threshold.

6.2 The threshold hypothesis was wrong

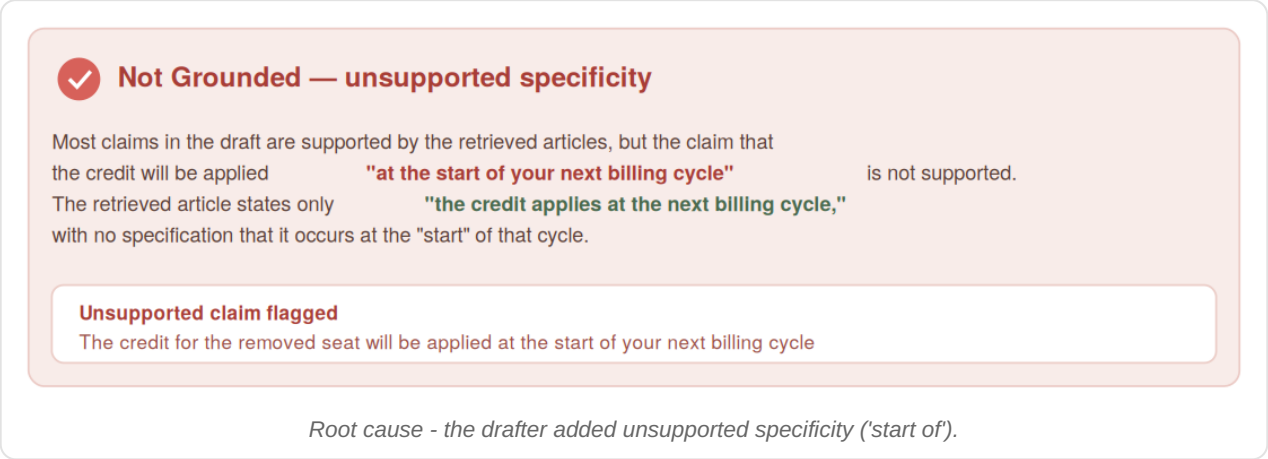
I tested retrieval in isolation using Studio's playground, which isolates one layer cheaply before spending on full pipeline runs. The result refuted the hypothesis: the cancellation article ranked number one at 84.43 percent, and the seat article ranked number one at 88.45 percent. Retrieval was working correctly, so the over-escalation was downstream.

Playground retrieval - the correct article ranks #1 at 84.43%.

Methodological point: the obvious hypothesis was testable and false, and testing it cheaply saved me from fixing the wrong layer.

6.3 Root cause: the drafting agent added unsupported specificity

Running the seat ticket as a full pipeline several times, one run grounded and one escalated. The Not-Grounded reasoning revealed the cause: the drafter wrote a credit "will be applied at the start of your next billing cycle," but the source says only "the credit applies at the next billing cycle." The verification agent correctly flagged "start of" as an unsupported detail and escalated.



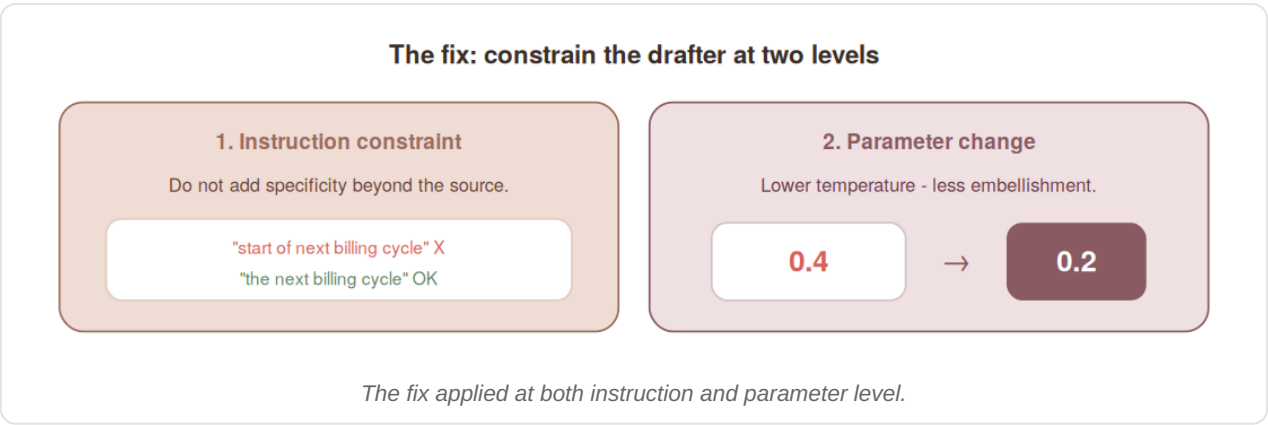
Root cause - the drafter added unsupported specificity ('start of').

This reframed the finding. The verification layer was not failing; it was catching a small hallucination the drafter introduced. At temperature 0.4, the drafter sometimes embellished, and the guardrail correctly responded to whichever draft it received. The non-determinism lived in the drafter.

6.4 Fix: constrain the drafter at instruction and parameter level

Two surgical Studio edits to the Response Drafting Agent, leaving the verification agent untouched so as not to weaken the guardrail:

1. Added a specificity constraint: do not make any statement more precise than the article states; if the article says "the next billing cycle," do not write "the start of the next billing cycle."
2. Lowered its temperature from 0.4 to 0.2.



The fix applied at both instruction and parameter level.

Tradeoff: temperature 0.4 gives more natural phrasing; 0.2 gives more literal, consistent output. For a grounded support tool, fidelity to source matters more than stylistic variety, so trading a little warmth for consistency is correct. A genuine tradeoff, not a free win.

6.5 Result: the specific bug is gone, residual variance remains

- Ticket 5, SSO: still escalates correctly. The guardrail survived the fix.
- Ticket 3, cancellation: now grounds reliably.

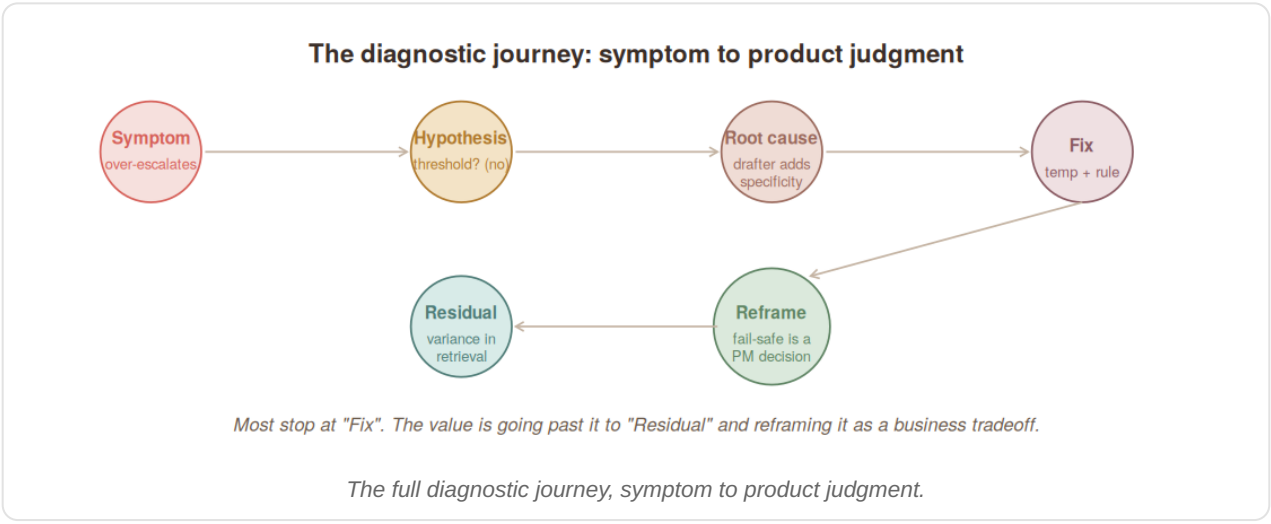
- Ticket 4, seat management: ran three times, grounded twice, escalated once. The "start of" embellishment is gone.

The fix reduced false escalations from frequent to roughly one in three and eliminated the diagnosed embellishment, but did not fully remove the variance.

6.6 The deeper finding: residual variance is architectural

The remaining escalated run showed the variance had moved upstream. Across runs, retrieval returned different chunks and even differently-framed article titles for the same query. So the pipeline is non-deterministic end to end: retrieval varies, which changes what the drafter sees, which occasionally produces a draft the verifier cannot fully ground.

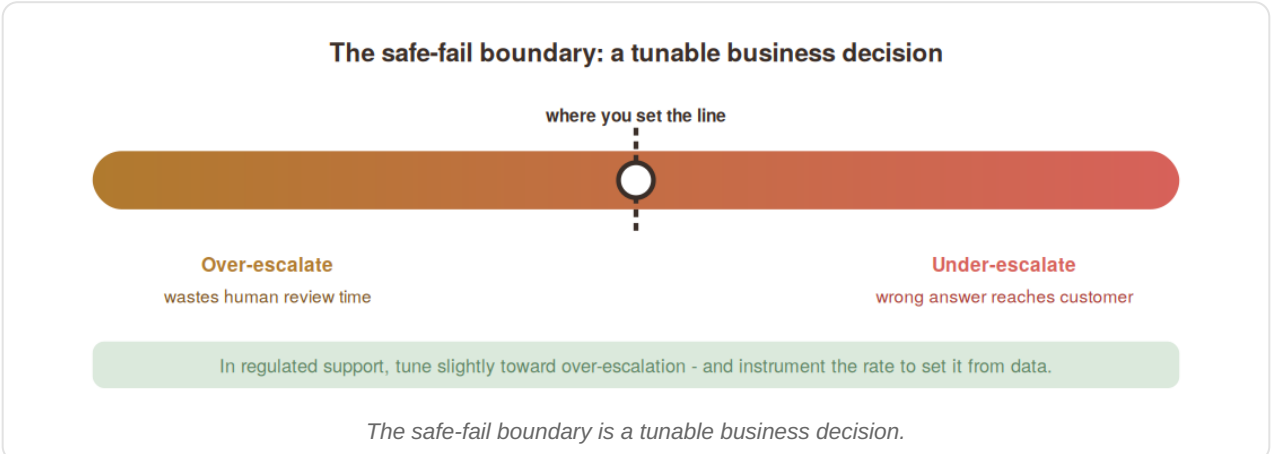
Honest conclusion: prompt-and-parameter tuning fixed the specific, diagnosable failure, but true consistency requires architectural work. Candidate approaches, in rough priority: pinning or caching retrieval within a session; a re-rank over borderline retrievals; lower temperature across all agents; or a confidence band that routes borderline cases through extra verification rather than discarding them.



The full diagnostic journey, symptom to product judgment.

6.7 The product reframe: the system fails safe

The most important judgment is not "eliminate all variance." It is that the system fails safe: when uncertain, it over-escalates to a human rather than hallucinating. For a support tool, that is the correct direction to fail. The real product decision is where the safe-fail boundary sits, balancing two costs: over-escalation wastes review time, under-escalation risks a wrong answer reaching a customer. In a regulated context I would tune toward slightly more escalation and instrument the escalation rate so the boundary is set from data.



The safe-fail boundary is a tunable business decision.

Full corrected characterization: clear-match tickets ground reliably; clear-no-match tickets escalate reliably (including the caught SSO hallucination and the GDPR no-source case); borderline tickets are non-deterministic near the semantic boundary, improved but not eliminated by the fix. This is a more honest and more useful picture than "six of six perfect," and it is reproducible.
