GroundTruth (Part 2)

Product Analysis and Recommendations for Lyzr Architect

Prepared by Abhishek Katkar. Part 2 turns the findings from Part 1 into a product analysis and a set of recommendations, written to be additive to what Lyzr already ships, with tradeoffs and mitigations for each.

> **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/

---

1. Executive summary

Combining my hands-on build experience with a study of Lyzr's existing feature set produced the analysis below. The central insight: Lyzr already has a mature reliability suite (Improvement Engine, Agent Eval, Simulation Engine, Hallucination Manager), but the two gaps that matter most are that the improvement loop is documented for instructions rather than model parameters, and that this tooling is Enterprise-gated while Architect targets self-serve builders, precisely the persona locked out of it.

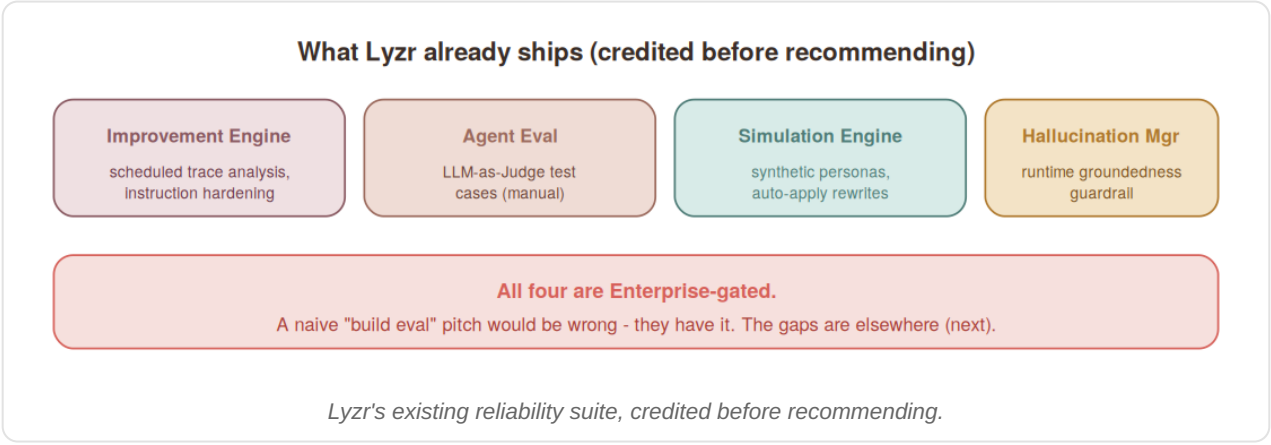
---

2. What Lyzr already ships

Before recommending anything, I studied Lyzr's existing feature set across Studio, the App Store, Monitoring, the pricing matrix, and public documentation, so recommendations are additive rather than reinventions. Lyzr already offers a substantial reliability suite.

- **Improvement Engine.** Analyzes live production traces on a configured schedule, groups failures into patterns (task-completion failures, hallucinations, tool and argument errors, relevancy, knowledge-retention), links them to traces, and generates agent-hardening suggestions shown as a structured diff for review and explicit push to production. Documented under the Enterprise Agent Studio area.
- **Agent Eval / LLM-as-Judge.** A user-initiated flow: select an agent, generate a test-case group, run it, and receive LLM-generated improvement suggestions. Described as manual, not an always-on production evaluator. Enterprise-only in the pricing matrix.
- **Simulation Engine.** Generates synthetic persona-and-scenario conversations, evaluates them, and can send failures to agent hardening, optionally auto-applying rewritten instructions and re-evaluating. An Enterprise block.
- **Hallucination Manager.** A runtime output layer performing Reflection (instruction adherence), Groundedness (factual adherence to sources), and Context Relevance (retrieval quality) before returning an answer. A guardrail, not an improvement engine. Enterprise-only.

The honest implication for my own build: several capabilities I might have proposed already exist in mature form. Every recommendation below starts by crediting them and then points precisely past them.



Lyzr's existing reliability suite, credited before recommending.

---

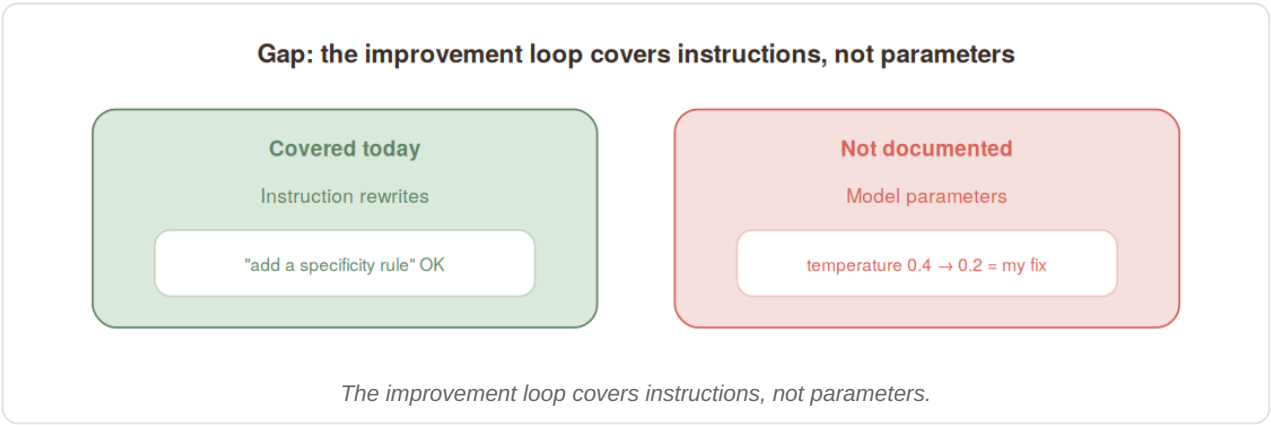
3. Product gap analysis and recommendations

Each recommendation is grounded in two sources at once: my build experience, and the documented state of Lyzr's features and pricing. Each is written as current state, gap, proposal, tradeoff, and mitigation.

3.1 Gap one: the improvement loop covers instructions, not model parameters

Current state. The Improvement Engine and Simulation Engine generate instruction-level hardening suggestions. The public documentation does not describe automatic optimization of model parameters such as temperature, model selection, or tool settings.

Evidence from my build. My grounding failure was only fully addressed by a parameter change. The instruction constraint helped, but lowering the drafting agent's temperature from 0.4 to 0.2 was the change that reduced the embellishment at its source. An instruction-only improvement loop would not have surfaced that lever.



The improvement loop covers instructions, not parameters.

Proposal. Extend the improvement suggestion surface to include parameter-level recommendations, for example: "this agent's drafting variance correlates with temperature 0.4; consider 0.2," presented as a reviewable suggestion, never silently applied.

Tradeoff. Parameter suggestions are higher-variance and harder to attribute confidently than instruction rewrites, so a poor suggestion could erode trust in the engine. Model choice in particular is expensive to get wrong.

Mitigation. Scope initial parameter suggestions to well-understood, low-risk levers (temperature, top-p) with clear rationale and expected effect, keep them behind human review, and measure suggestion acceptance rate before widening scope.

Honest calibration. I did not verify parameter optimization is entirely absent; I verified it is not documented and was not available at my tier. I would frame this internally as "not documented and not accessible at self-serve" rather than "definitely does not exist."

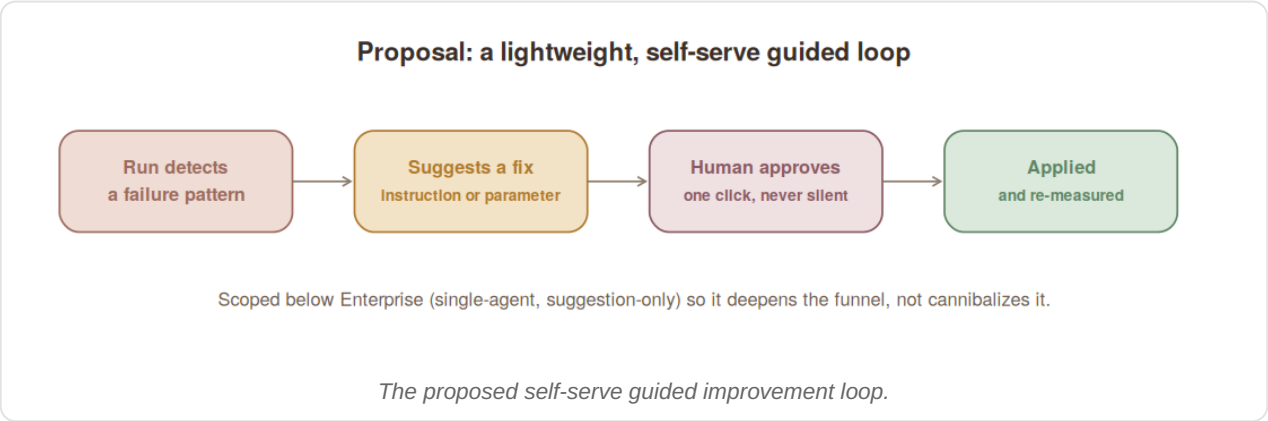
3.2 Gap two: the reliability tooling is Enterprise-gated, but Architect targets self-serve builders

Current state. Agent Eval, Responsible AI, Hallucination Manager, the Simulation Engine, and the Improvement Engine are Enterprise or Custom tier per the public pricing matrix. Architect is positioned for self-serve, non-technical builders (agentic apps from plain text).

Evidence from my build. I only diagnosed and fixed my grounding failure because I was technical enough to read verification reasoning, infer the drafting agent as the cause, know embellishment correlates with temperature, navigate to the model-parameter panel, and re-test. That is roughly eight technical inferences. Architect's actual target user cannot perform it, and would experience the product as "it escalates strangely sometimes," then lose trust, with none of the reliability tooling available to help.

Reading the decision charitably. Gating reliability tooling behind Enterprise is the textbook-correct call for an early-stage company. It protects the ARR that funds the business and concentrates scarce engineering on the accounts that pay today. This was the right stage-appropriate trade, and the recommendation below is not a correction of it. It is a next-stage question: as the self-serve funnel matures from a top-of-funnel signal into a growth engine, does a bounded version of this tooling convert more of those users into the Enterprise buyers the gating is meant to protect, rather than cannibalizing them?

FIG_STAGE **Proposal.** Offer a lightweight, self-serve-tier version of the improvement loop: detect a failure pattern from a user's own runs, surface a plain-language explanation and a specific reviewable fix (instruction or parameter), and apply on one-click human approval. Not the full Enterprise suite; a guided, bounded assistant aimed at the activation gap.



The proposed self-serve guided improvement loop.

Tradeoff. This risks cannibalizing an Enterprise differentiator and adds cost to a low-priced tier. Reliability tooling is part of what justifies the Enterprise price today.

Mitigation. Scope the self-serve version deliberately below Enterprise: single-agent, limited history, suggestion-only for the most common failure classes, with continuous, multi-agent, production-scale, and auto-hardening capabilities reserved for Enterprise. The goal is activation and trust, not feature parity. Framed correctly this deepens the funnel into Enterprise rather than undercutting it, because builders who succeed at self-serve are the ones who grow into paid tiers.

Why this is the strongest recommendation. It is a packaging and go-to-market judgment, squarely a PM domain, grounded in Lyzr's own pricing page and product positioning rather than opinion. It connects a documented pricing fact to a documented positioning fact and to a real activation gap I experienced directly, and it is framed as an evolution of a sound early-stage decision rather than a critique of it.

3.3 Gap three: no unified, self-serve closed loop from monitoring to parameter-level improvement

Current state. The components exist (Monitoring with error rates, Agent Eval, Improvement Engine, Simulation Engine), but they are not connected into a single closed loop for the self-serve builder, and the loops that exist are instruction-focused and Enterprise-gated.

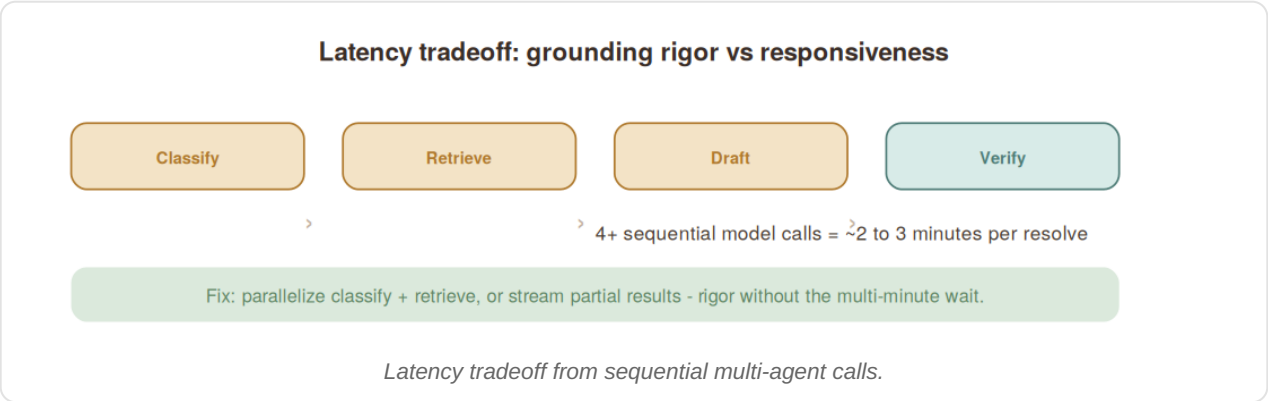
Proposal. A connected, guided loop for self-serve: a run detects a failure pattern, the improvement surface proposes the specific fix (instruction or parameter), and the user approves it in one click. The synthesis of gaps one and two.

Tradeoff and mitigation. As above: bound the scope well below Enterprise, keep humans in the loop, and measure acceptance to protect trust.

---

4. Secondary product findings and recommendations

- **Multi-agent latency.** A resolve takes roughly two to three minutes because the Manager orchestrates four-plus sub-agents in sequence, each making its own model call. For a live support context this is a real tradeoff between grounding rigor and responsiveness. Recommendation: parallelize the classification and retrieval steps, or stream partial results, so grounding does not come at the cost of a multi-minute wait.



Latency tradeoff from sequential multi-agent calls.

- **Runtime self-healing worked, and revealed a reliability signal.** During testing the app hit a transient 502 from the inference gateway, and Architect automatically diagnosed it and added retry logic (retries on 502, 503, 504 with backoff, passing genuine errors through). Good self-healing at runtime, and also a signal that the inference gateway has reliability characteristics worth monitoring at scale.
- **First-run credit economics gate the activation loop.** The free tier grants 20 credits; my first build consumed 18, and testing exhausted the remainder, so I could not complete a single build-and-test cycle on the free tier. The activation aha, seeing your own app work on real input, sits just past the credit wall. Recommendation: guarantee at least one full build-and-test cycle on the free tier, even if the build allowance shrinks, using the existing guided prompt libraries to bound cost.
- **A new-account credit-display issue.** A newly created account showed zero credits, most likely device or browser fingerprinting for anti-abuse. Sensible protection, but a confusing first moment. Recommendation: surface a short message so it reads as policy, not a bug.

---

5. Honest limits and what I would validate with internal data

This analysis is from one build session, from outside the company, as a first-time user of these products. Its conclusions are hypotheses I would resolve with internal data on day one, not assertions.

I would want to know: what share of self-serve builders hit an over-escalation or inconsistent-behavior moment on their first app; how often the Improvement Engine's instruction suggestions alone resolve a failure versus needing a parameter change; the real distribution of retrieval-score gaps between grounded and ungrounded queries across many knowledge bases, which determines whether a confidence-band approach is viable; and the actual production error and latency profile of the inference gateway. Each turns a hypothesis here into a data-backed decision.

I would also validate the parameter-optimization gap directly against the internal roadmap, since I could only confirm it is undocumented and unavailable at my tier, not that it is absent.
