GroundTruth (Part 5)

Product Requirements Document: The Guided Self-Serve Reliability Loop for Lyzr Architect

Prepared by Abhishek Katkar. This PRD specifies the headline recommendation from Part 2: a prescriptive, self-serve reliability and improvement loop for Architect. It is written as a build-ready specification with measurable success criteria, competitive positioning, scope boundaries, and a phased rollout.

> **Approach and constraints.** Written from the outside, as a first-time user, on a limited free-credit allowance, using only public information. Every requirement below maps to a diagnostic step I performed by hand during the GroundTruth build; the feature productizes that manual work for users who cannot do it themselves. Internal telemetry would refine the specific thresholds and targets.
>
> **Live app used to derive these requirements:** https://support-sentinel-bold-edge-77s7.architect.space/
>
> **How to read this.** GroundTruth was the test vehicle. This PRD is about Architect, the Lyzr product. It specifies a feature for Architect, informed by what building GroundTruth revealed.

---

1. Problem statement

Architect targets self-serve, non-technical builders who create agentic apps from plain text. When an agent misbehaves (over-escalates, hallucinates, returns inconsistent results), that builder currently has no accessible path to diagnose and fix it. The reliability tooling that would help (Agent Eval, Improvement Engine, Simulation Engine, Hallucination Manager) sits in the Enterprise tier and is oriented toward technical operators.

During the GroundTruth build I hit exactly this wall. My agent over-escalated valid tickets. Diagnosing it required roughly eight technical inferences: reading verification reasoning, isolating retrieval in a playground, identifying that the drafting agent added unsupported specificity, knowing that specificity correlates with temperature, locating the model-parameter panel, lowering temperature from 0.4 to 0.2, and re-testing. A non-technical builder cannot perform this chain. They experience the product as "it behaves strangely sometimes," lose trust, and churn before reaching the value moment.

The core problem: reliability failures on the self-serve tier are invisible to diagnose and impossible to fix for the target persona, and a failed or untrustworthy first run blocks activation entirely.

---

2. Goals and success metrics

2.1 Goals

1. Make agent reliability failures diagnosable and fixable by non-technical self-serve builders, without datasets, evaluators, or CI pipelines.
2. Convert a reliability failure from a churn event into a guided recovery moment.
3. Extend Lyzr's public reliability advantage (85 percent of agent projects reach production, versus a cited industry average below 30 percent) from the Enterprise tier down to the self-serve first-run experience.

2.2 Success metrics (targets are illustrative pending internal baselines)

Metric	Definition	Target
Fix-adoption rate	Share of surfaced suggestions the user approves and applies	>= 35 percent
First-fix time	Median time from failure detected to a fix applied	< 5 minutes
Post-fix resolution	Share of applied fixes that measurably reduce the failure on retest	>= 60 percent
Activation lift	Increase in first-run-to-activation among users who hit the loop, vs a control cohort	+8 to 12 points
Self-serve retention	30-day retention of users who engaged the loop vs those who hit a failure without it	+15 to 25 percent relative
Trust signal	Reduction in support tickets and forum posts describing "inconsistent" or "broken" agent behavior	-30 percent

The activation and retention targets are anchored to published PLG benchmarks (activation in the 20 to 40 percent range; strong first-week activation strongly associated with strong three-month retention) and should be replaced by Lyzr's measured cohort rates before launch.

---

3. Users and use case

Primary persona: the self-serve builder. Non-technical or lightly technical. Builds an agent from a prompt, tests it, sees inconsistent or wrong behavior, and does not know why or how to fix it. Today they either give up or file a support ticket. This is the persona Architect is sold to and the one currently unserved by reliability tooling.

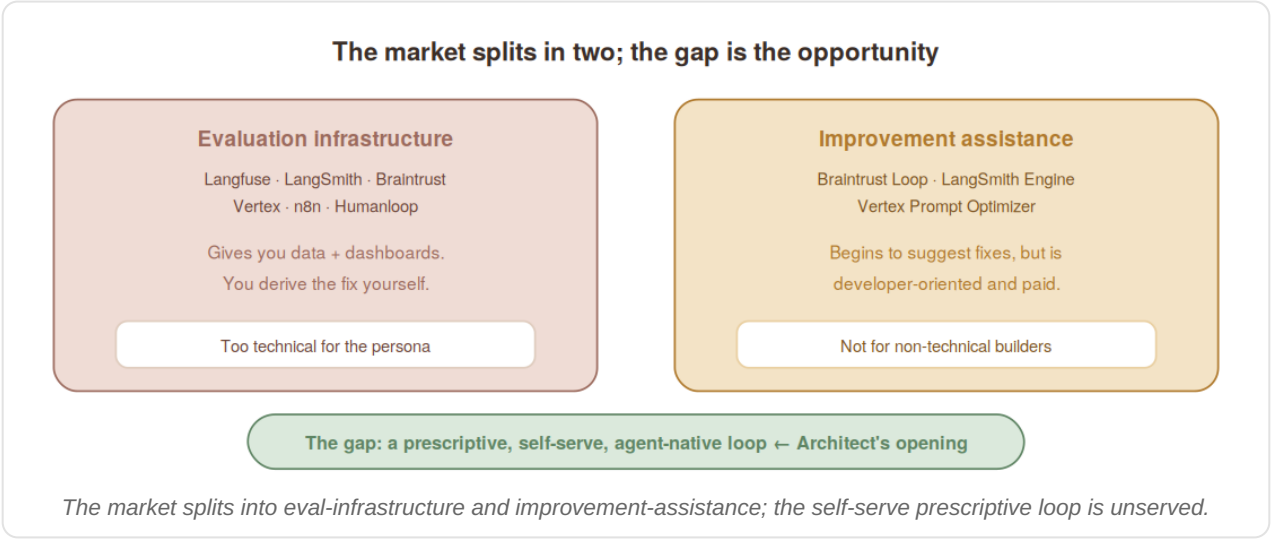
Secondary persona: the technical self-serve builder. Can read a trace but does not want to configure datasets, evaluators, and observability pipelines to fix one agent. Wants the fix suggested, not the infrastructure to derive it.

Anchor use case (from the GroundTruth build). A support agent over-escalates valid tickets. The loop detects the pattern, explains in plain language that the drafting step is adding detail the source does not support, recommends a specific fix (add a specificity constraint and lower drafting temperature to 0.2), generates a regression test from the failing cases, runs the candidate fix against those cases and a golden set, shows the quality and latency delta, and lets the user approve or roll back.

---

4. Competitive landscape and positioning

The market splits into two camps, and the opportunity sits in the gap between them.



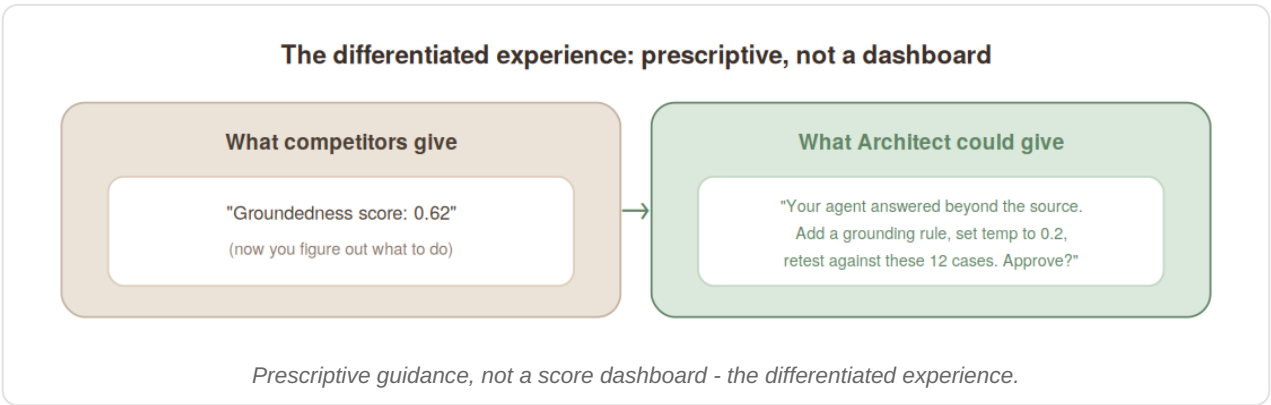
The market splits into eval-infrastructure and improvement-assistance; the self-serve prescriptive loop is unserved.

Evaluation infrastructure (Langfuse, LangSmith, Braintrust, Vertex AI, n8n, Humanloop) provides traces, datasets, judges, and dashboards. It hands the user data and requires them to derive the fix. Powerful for engineers, intimidating for the self-serve persona. Several are genuinely self-serve accessible (Langfuse Hobby is free; Braintrust Starter is free with unlimited users; LangSmith Developer is free at 5,000 traces per month), but all require the user to interpret results and implement changes themselves.

Improvement assistance (Braintrust Loop, LangSmith Engine, Vertex Prompt Optimizer) begins to recommend or generate fixes. This is the closest prior art and must be credited honestly. However, these are developer-oriented, and their fix generation is metered or paid (LangSmith Engine is not on the free Developer tier; Plus starts at 39 dollars per seat per month), and not fully connected to agent-level deployment for a non-technical builder.

The direct agent-builder competitor, CrewAI, gates its Hallucination Guardrail behind Enterprise, exactly as Lyzr currently does. This is the key positioning insight: the entire agent-builder category leaves the self-serve reliability experience unserved. This is a winnable differentiation, not a catch-up.

Positioning statement. No competitor offers a prescriptive, agent-native, self-serve improvement loop for a non-technical builder. The eval-infrastructure players are too technical for the persona; the improvement-assistance players are developer-oriented and paid; the agent-builder players gate reliability behind Enterprise. Architect can own the prescriptive, beginner-accessible experience: not "here is a score," but "here is what broke, here is the specific fix, approve to apply."



Prescriptive guidance, not a score dashboard - the differentiated experience.

5. Requirements

The loop consists of seven capabilities. Each maps to a manual step I performed during the GroundTruth build, which is the evidence that the sequence is complete and correct.

5.1 Functional requirements

R1. Detect a failure. Identify a failure from a trace signal or a user rating: an escalation where a valid source existed, a hallucination flagged by the runtime guardrail, a low groundedness score, a tool or argument error, or an explicit thumbs-down. Must run on the self-serve tier against the user's own recent runs, not require a configured dataset.

R2. Classify the likely cause. Attribute the failure to a category: prompt ambiguity, missing knowledge, poor retrieval, wrong tool choice, invalid tool arguments, excessive temperature, poor routing, or insufficient guardrails. In the GroundTruth case, the correct classification was "drafting agent adding unsupported specificity, correlated with temperature."

R3. Recommend a concrete change. Produce a specific, human-readable recommendation targeting the prompt, model, retrieval settings, tool schema, or a parameter. Critically, and unlike the documented Improvement Engine which covers instructions, this must include parameter-level suggestions (for example, temperature 0.4 to 0.2), because that was the change that actually fixed the GroundTruth failure.

R4. Generate a focused regression test. Create a small test set from the failing cases so the fix can be validated against the exact failures it targets.

R5. Run the candidate fix against the failure and a golden set. Apply the proposed change in a sandbox, run it against both the new failing cases and an existing set of known-good cases, to confirm the fix helps without regressing prior behavior. In the GroundTruth build this is the step I did manually by re-running the SSO ticket to confirm the fix did not weaken the guardrail.

R6. Show the quality, cost, and latency delta. Present a clear before-and-after: failure rate, and any change in cost or latency, so the user makes an informed decision.

R7. Approve, compare, and roll back. The change is never applied silently. The user approves it in one click, can compare versions, and can roll back. This preserves the human-in-the-loop governance that is central to Lyzr's trust positioning.

5.2 Non-functional requirements

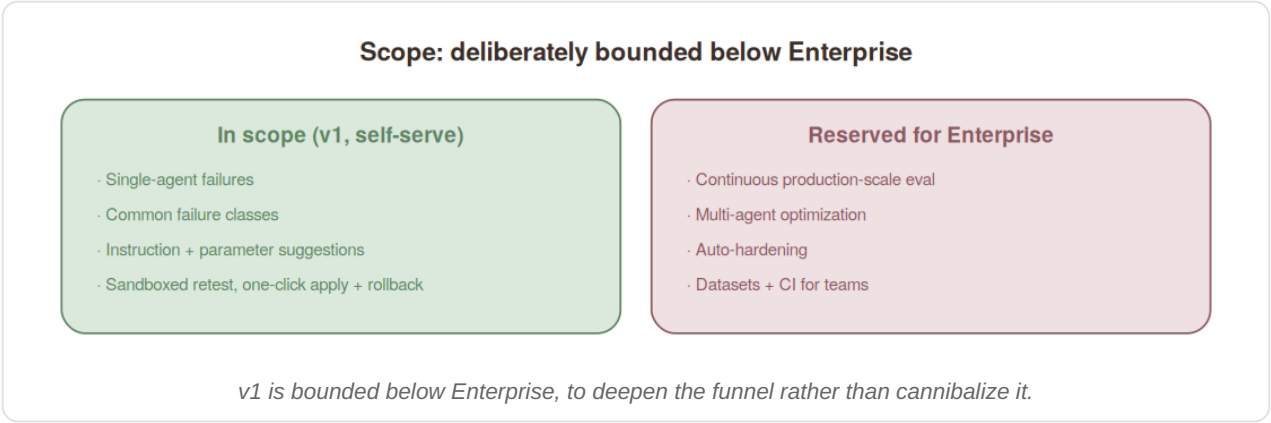
- Runs on the self-serve tier within a defined credit budget per loop invocation, disclosed to the user before running.
- First suggestion surfaced within seconds of a detected failure; a full sandboxed retest completes within a single-digit-minute window consistent with current resolve latency.
- Plain-language output at all times. No requirement to understand datasets, evaluators, or observability configuration.
- All suggestions are reviewable and reversible. No autonomous production changes.

---

6. Scope and non-goals

In scope (v1): single-agent failures on the self-serve tier; the most common failure classes (over-escalation, hallucination or groundedness, temperature-driven variance, missing-knowledge); instruction and parameter suggestions; sandboxed retest against failing cases and a small golden set; one-click apply and rollback.

Explicit non-goals (v1): - Not the full Enterprise suite. Continuous, always-on, production-scale evaluation, multi-agent optimization, and auto-hardening remain Enterprise. This bounding is deliberate, to deepen the funnel into Enterprise rather than cannibalize it. - No silent or autonomous production changes. - No arbitrary parameter optimization beyond a curated, low-risk set (temperature and top-p first; model selection deferred, since it is expensive to get wrong). - Not a replacement for datasets and CI for teams that want them; those remain the Enterprise path.



v1 is bounded below Enterprise, to deepen the funnel rather than cannibalize it.

7. Phased rollout

Phase 1 (foundational). R1, R2, R3 for a single failure class (over-escalation and groundedness), instruction-level suggestions only, surfaced as a plain-language explanation plus a reviewable suggestion. Ship to a small self-serve cohort behind a flag. Success gate: fix-adoption rate above 25 percent.

Phase 2 (the fix loop). Add R4, R5, R6: regression-test generation and sandboxed retest with a quality and latency delta. Add parameter-level suggestions (R3 extended), starting with temperature. Success gate: post-fix resolution above 60 percent and no measurable increase in guardrail bypass.

Phase 3 (breadth and trust). Add R7 comparison and rollback UI, expand failure classes (tool and argument errors, routing), and instrument the activation and retention impact against a control cohort. Success gate: measured activation lift on the treated cohort.

8. Risks and mitigations

Risk	Mitigation
Cannibalizes Enterprise reliability tooling	Bound v1 strictly below Enterprise (single-agent, suggestion-only, limited history). Position as an on-ramp that grows Enterprise demand.
A poor suggestion erodes trust in the loop	Keep all suggestions human-approved, start with low-risk levers, and measure fix-adoption and post-fix resolution as guardrail metrics; pause a suggestion class that underperforms.
Parameter suggestions are harder to attribute than instruction rewrites	Scope to temperature and top-p first, with clear rationale and expected effect; defer model-selection suggestions.
Credit cost of sandboxed retests on the free tier	Disclose the credit budget before each loop run; cap retest size in v1; use the existing guided prompt libraries to bound cost.
Non-determinism means a fix may not fully resolve a borderline case	Frame outcomes honestly (failure-rate reduction, not elimination), consistent with the fail-safe design; surface residual variance rather than hiding it.

9. Open questions for internal validation

1. What is the real distribution of first-run failure classes across self-serve builders? This sizes which failure classes v1 should cover.
2. What is the measured activated-to-paid conversion difference between users who hit a failure with the loop versus without? This validates the business case in Part 3.
3. Does the Improvement Engine already cover any parameter-level suggestions internally, even if undocumented? This scopes R3 precisely.
4. What credit budget per loop invocation is acceptable on the free tier without eroding unit economics?
5. Where should the safe-fail boundary default sit for the most common failure classes, and should it be user-tunable?

---

10. Appendix: why this specification is credible

Every one of the seven requirements corresponds to a step I performed manually to fix a real failure in GroundTruth, built on Architect. The feature is not speculative; it is the automation of a diagnostic process I completed by hand, for the users who cannot complete it themselves. The competitive gap is documented across nine platforms. The success metrics are anchored to published PLG benchmarks and to Lyzr's own stated reliability advantage. The remaining unknowns are listed honestly in section 9, to be resolved with internal data.

Competitive sources: public documentation and pricing for LangSmith, CrewAI, Vellum, Braintrust, Langfuse, Humanloop, Vertex AI Agent Builder, Microsoft Copilot Studio, and n8n, as of August 2026. Benchmark sources: OpenView, ProductLed, Amplitude, a16z. All competitor capabilities and pricing should be re-validated before external use, as they change frequently.