Ledgerkeep

A Guardrailed Autonomous Operations Agent as a Decentralized AI Service

Dipankar Sarkar
doom2quake

August 2026
Version 0.1.0 (Milestone 1)

Abstract

Anomaly detection is a solved commodity; the unsolved step is the one after the alert. An operations agent that a team would let near its systems must be trusted to investigate, attribute a moved number to a specific change, quantify the loss, and propose the exact fix, without being trusted to execute anything a human has not approved. That gap is almost entirely guardrails, durable memory, correct routing, and an honest report of what the agent actually did. Ledgerkeep is that layer, packaged as a decentralized AI service on the ASI / Fetch.ai stack: a stable, typed request and response, and a verifiable audit object returned with every call. This paper documents the milestone-1 deliverable: the service contract, the guardrailed investigation loop, and the adapter seam that makes the funder's technology load-bearing while keeping the whole system runnable offline and keyless. We report reproducible test counts and state plainly what has not been built.

1 The problem

Consider the incident Ledgerkeep is named for. A payments configuration is pushed at 14:03. It routes one region's card traffic to a gateway that declines most authorisations. Order volume still looks normal, because customers keep trying to buy; completed, settled revenue quietly falls, and nobody notices until the next morning. The person who suffers this is the on-call operator, paged at 03:00, who must reconstruct from a dashboard that shows only the symptom which of forty deploys that day moved a number, how much money has already been lost, and what one line to change. That reconstruction is slow, it happens under pressure, and it is where the cost lives.

The reason this stays unsolved is not detection. Trailing-window outlier detection is textbook [4]. The unsolved part is the step after the alert: an agent trusted to investigate and propose, but not to act. The market is full of demonstration agents that will happily run generated SQL against production or act on model text that contains an injection [6]. The distance between a demo agent and one an operator would let near their systems is guardrails, durable memory, correct routing, and an honest report. That is the layer nobody wants to build twice.

2 Why a decentralized service, and why ASI

ASI Deep Funding funds decentralized AI services: agents that others can discover, call, and pay for, while the builder keeps ownership [2]. An operations agent that a team must trust is exactly the kind of component that benefits from being a standalone, independently callable service with a published, auditable contract, rather than a black box buried inside one company's stack.

Ledgerkeep maps onto that thesis directly. The agent is a marketplace service: a stable, typed request and response, a guarantee about what it will and will not do, and an audit object returned with every call. On the ASI / Fetch.ai stack a service is a uAgents agent registered on Agentverse [3]; the SingularityNET marketplace is where such a service is discovered and monetized [7]. The Model Context Protocol [1] is the interoperability seam: the same agent that answers a marketplace call can be consumed as a tool by another agent, which is the composition story the Alliance is built around.

3 The service contract

Milestone 1 delivers Ledgerkeep as a callable service, so the interface is the product. The contract is transport-agnostic: plain typed objects with an explicit JSON round-trip, mapped onto a uAgents message by the marketplace adapter, onto an MCP tool by the MCP bridge, or exchanged directly by a local caller. One contract, many transports.

Request. InvestigationRequest is typed and all-defaults, so the simplest possible call is InvestigationRequest(). Fields are the KPI to watch, the z-score threshold, a free-text note recorded for the audit trail but never executed, and a caller correlation id echoed back on the response.

Response. InvestigationResponse carries the anomaly, the attribution, the quantified impact, the proposed fix, a verification note, and a delivery_mode that is always present and reads offline_fixture when the figures came from the in-repo ledger rather than a live source, so a stub is never mistaken for a live figure. Two invariants are enforced by the schema itself: the proposed fix is always returned with approved = false (the merge stays with a human), and the response cannot be constructed without an audit object.

Audit object. AuditObject is returned on every call. It records the run id, the ordered guardrail decisions by name, the classified domain, the recurrence record, and a SHA-256 content_hash over the canonical JSON of the decision chain. A caller can recompute the hash from the visible fields and confirm the audit trail was not edited. This is the surface that milestone 3 anchors to a public testnet; here it is computed and returned, so the tamper-evidence primitive already exists.

```
audit.run_id       = run-20260825T204713-a75959
audit.status       = acted
audit.domain       = finance
audit.guardrails   = [CONTENT_SAFETY:clean, DOMAIN_ROUTER:classified,
                     ACTION_LIMITER:allowed]
audit.content_hash = 7d735e4ee91547f25deb2bd262182fd85439c0a87ed79c...
```

4 The guardrailed loop

run_service takes a request and runs a deterministic, offline loop against a fixture ledger, enforcing every guardrail from the vendored control plane and recording each decision by name on a durable run document.

1. Detect the settled-revenue drop by trailing z-score.

2. Attribute it to a specific change by a deterministic rule over the change-event log. The attribution text is screened by CONTENT_SAFETY before anything downstream trusts it; a flagged run is quarantined with no proposed fix, so a poisoned diagnosis cannot talk the agent into a remediation.
3. Route the incident through a keyword domain classifier so the audit trail records the classified domain.
4. Quantify the loss to the dollar.
5. Propose the one-line fix, returned unapproved. Recording the proposal passes the ACTION_LIMITER rate cap.
6. Verify by re-reading the metric after a fixture failover, as the next live cycle would after a real remediation.

The bound on autonomy is three named guardrails, each tested without a model. A security incident that were also a finance incident is routed to security first, because misrouting it would lose the incident [5]. Nothing in this loop reaches the network.

5 The adapter seam

The funder's technology is load-bearing behind a seam. When a seed phrase is set and the uagents SDK is installed, on a test network, the marketplace adapter builds a real uagents.Agent, registers a handler that decodes an incoming message into an InvestigationRequest, runs the identical run_service, and replies with the response. Otherwise an in-process transport answers the exact same contract, so the service is callable and testable with no SDK and no credentials. The live handler is a one-line funnel into the same call the offline path uses, so the two transports cannot diverge in behaviour. All on-chain interaction is testnet only for the duration of the grant; a non-test network is refused.

A describe() manifest is produced from the same source of truth the adapter uses, reporting the service title, network, contract version, the request and response model names, and which transport is active, so a published description can never drift from the running service.

6 Evaluation

The suite is hermetic: it pins the environment offline and in-memory and cuts the socket, HTTP, and subprocess layers before the package is imported, so nothing in the suite can reach an external service even by accident. Tests that would exercise an integration inject their own fake transport.

Surface	Tests
Service contract (round-trip, audit hash, invariants)	10
Guardrailed loop (detect / attribute / quantify / propose / verify)	7
Marketplace adapter (transports, manifest, testnet refusal)	10
Fixture ledger (deterministic scenario, failover)	7
Vendored agent_core control plane	48
Total (all passing, offline)	82

Every assertion about the loop is read off the returned audit object: a fact about what the agent did, not a claim about what it said. Two tests confirm the guardrails bite: a poisoned attribution quarantines the run with no fix, and a zero-cycle action cap blocks the proposal and records the block.

7 Honest limits

Milestone 1 is a callable, tested, self-contained service. It is not a deployment. There are no users, no revenue, no partnerships, and no external audit. There is no live ASI marketplace publication in this repo: the live uAgents path is present behind the seam but has never been exercised against Agentverse in this environment. There is no mainnet, and there will be none during the grant; the audit-hash anchoring is milestone 3 and is off by default. The content-safety and read-only-SQL screens in the control plane are deny-list text screens, not parsers; they are defence in depth and the real boundary for any generated query is a read-only credential with a server-side cost cap. These limits are stated in docs/LIMITATIONS.md so diligence finds nothing that was not disclosed.

8 Conclusion

The durable output is not one agent, it is the safety layer under it. The vendored agent_core control plane is MIT-licensed and separable, so other ASI builders can vendor the exact tested guardrails rather than re-implementing them. The service contract and the audit object are a worked reference for how to publish and compose a guardrailed agent service on the stack. Milestone 1 ships that reference, running and tested, keyless.

References

[1] Anthropic. Model context protocol specification. https://modelcontextprotocol.io/, 2025. Accessed 2026-08-26.

[2] Artificial Superintelligence Alliance. Deep funding: milestone-based, non-dilutive funding for decentralized AI services. https://deepfunding.ai/, 2026. Accessed 2026-08-26.

[3] Fetch.ai. uAgents: a lightweight framework for building and registering autonomous agents on Agentverse. https://fetch.ai/docs, 2026. Accessed 2026-08-26.

[4] Frank E. Grubbs. Procedures for detecting outlying observations in samples. Technometrics, 11(1):1–21, 1969.

[5] Michael T. Nygard. Release It! Design and Deploy Production-Ready Software. Pragmatic Bookshelf, 2nd edition, 2018.

[6] OWASP Foundation. OWASP top 10 for LLM applications. https://owasp.org/www-project-top-10-for-large-language-model-applications/, 2025. Accessed 2026-08-26; prompt injection as the leading risk class.

[7] SingularityNET. The SingularityNET marketplace: publishing and monetizing decentralized AI services. https://singularitynet.io/, 2026. Accessed 2026-08-26.