AEGIS: A Self-Healing Distributed Coordination Service with a Control-Plane-Confined LLM Operations Agent

Abhishek Aditya

June 2026

Abstract

Distributed coordination services — the Chubby/ZooKeeper/etcd class of systems that underpin leader election, distributed locking, and configuration storage for modern infrastructure — are correctness-critical and operationally expensive. A recurring temptation is to apply large language model (LLM) reasoning directly to consensus: choosing replication targets, vetoing log entries, or auto-applying tuning changes in real time. We argue this is exactly wrong. Consensus correctness rests on *deterministic replication* — every replica applying the same log entry must reach the same state — and a non-deterministic model in that path breaks the guarantee that makes the system safe.

We present Aegis, a self-healing distributed coordination service that takes the opposite structural stance. The *data plane* is a five-node Apache Ratis (Raft) cluster in Java 25 exposing distributed locks with fencing tokens, session leases, and a versioned key–value configuration store over gRPC; it contains no LLM and no Python. A separate *control plane* runs a Python operations agent that classifies anomalies with deterministic heuristics (*not* the LLM), proposes configuration fixes through a deterministic safety envelope as human-reviewed GitHub pull requests, and drafts incident postmortems. The two planes share a *telemetry surface* (Prometheus pull plus Redis Streams push), never a *mutation surface*: the agent's sole pathway to the cluster is a pull request a human merges. This is the project's load-bearing invariant (ADR-001).

We also release ConsensusOps-Bench, a reusable benchmark of 33 labelled Raft incidents scoring two capabilities — anomaly classification and ranked root-cause diagnosis. On this corpus the deterministic baseline achieves perfect classification (33/33) and root-cause top-1 0.970 / top-3 1.000. A LLM-with-retrieval diagnoser (a low-cost Gemini Flash model with retrieval) *matches the deterministic baseline exactly* — top-1 0.970, top-3 1.000 — in 51.4 s and $0.0088 of API spend for the full run. The headline finding: a cheap model with good retrieval suffices for diagnosis, the determinism lives in the free classifier, and neither ever touches the data path.

1 Introduction

Distributed coordination services are the quiet foundation of modern cloud infrastructure. Every Kubernetes cluster, every Cassandra ring, every distributed lock manager depends on a Chubby [1], ZooKeeper [4], or etcd [5]-class system to elect leaders, hold locks, and store configuration consistently. These systems are correctness-first: they sit beneath everything else, so a bug becomes everyone's bug. They are also operationally expensive. Split-brain scares, lease starvation, slow-follower cascades, and the dark art of tuning consensus timing parameters consume disproportionate on-call effort, and incident postmortems chronically lag incident frequency.

It is tempting to point a modern LLM agent at this pain and let it *act*: pick the replication target, veto a suspicious log entry, or push a tuning change the instant a metric crosses a threshold. We argue this temptation should be resisted at the level of system architecture, not policy. Consensus

algorithms — Paxos [3], Raft [2], ZAB — rest on a small number of narrow correctness invariants and are *extremely* sensitive to non-determinism. The core guarantee is that two replicas applying the same committed log entry produce the same state. A non-deterministic model anywhere in the commit path violates it. The blast radius is the whole system.

The determinism tension. This produces a tension that AEGIS is built to resolve cleanly. We *want* an agent's pattern-recognition and narrative reasoning for the operational toil; we *cannot* allow its non-determinism anywhere near replication. AEGIS resolves the tension structurally by physically separating two planes that share only a read-only telemetry surface:

- a **data plane** — a five-node Apache Ratis (Raft) cluster in Java 25 providing locks, leases, and a versioned KV store over gRPC — that is entirely LLM-free; and
- a **control plane** — a Python operations agent — that *observes, recommends, and documents*, but never decides consensus outcomes and never mutates the cluster directly.

The agent consumes only *exported* telemetry (a Prometheus scrape plus a Redis Streams event feed) and its only way to change the cluster is to open a GitHub pull request against a versioned configuration repository that a human reviews and merges. We call this the control-plane invariant (section 3, ADR-001): *the LLM is never in the request/data path.*

Making the position a result. An architectural stance ("LLM in the control plane only") risks reading as a *position* rather than a *result*, and risks the LLM looking decorative since anomaly classification is deterministic. AEGIS answers both concerns with measurable artifacts. The config proposer is gated by a deterministic *safety envelope* that makes unsafe consensus configurations structurally impossible to propose (section 3). The root-cause diagnoser is evaluated against a deterministic baseline on a released benchmark. The contribution is engineering, design, and evaluation — not a new consensus algorithm.

Contributions.

1. **An architecture** that confines an LLM operations agent to the control plane of a Raft coordination service, with two planes sharing a telemetry surface but never a mutation surface (section 2, 3).
2. **A deterministic safety envelope** for consensus configuration that rejects unsafe patches before any pull request is opened, so safety is a property of the code rather than the model's good behaviour (section 3).
3. **CONSENSUSOPS-BENCH**, a reusable, pluggable benchmark of 33 labelled Raft incidents scoring anomaly classification and ranked root-cause diagnosis as two separate capabilities (section 6).
4. **An honest evaluation** (section 7) showing the deterministic baseline is already near-perfect and a one-cent LLM-with-retrieval run *matches* it exactly — evidence that cheap retrieval suffices and the determinism belongs in the free classifier.

2 System Architecture

AEGIS is organised as two planes connected only by a telemetry surface. Figure 1 shows the whole system; this section describes the data plane, and section 3 describes the control plane and the invariant.

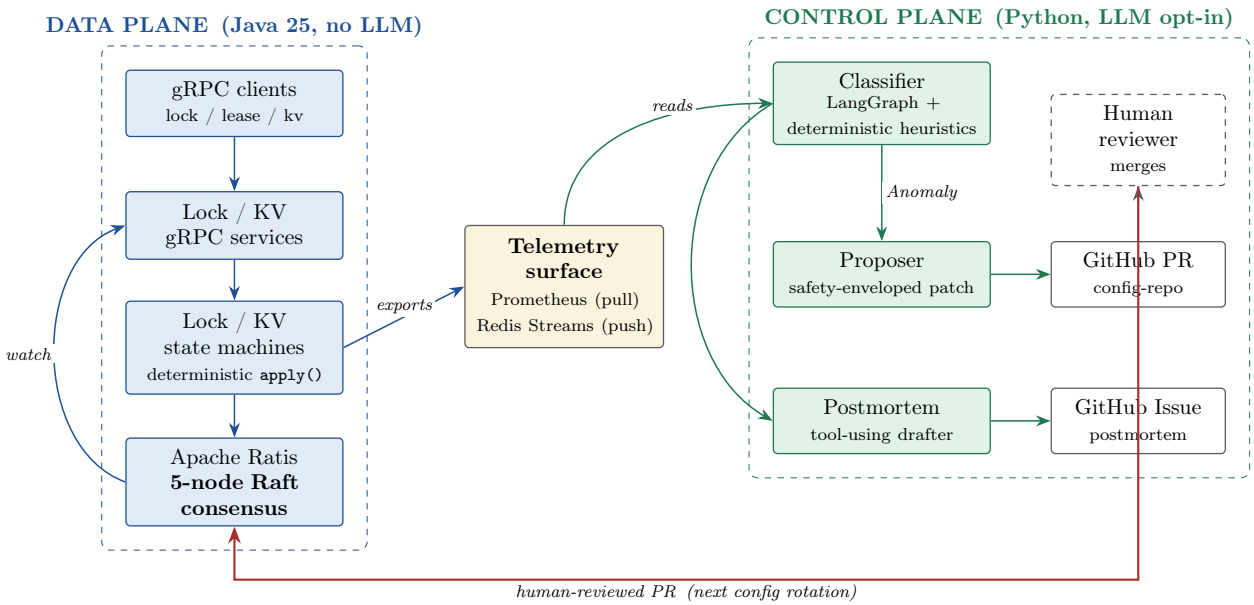

Figure 1: The two-plane architecture. The data plane (left, blue) exports telemetry one-way to a shared *telemetry surface* (centre, yellow); the control plane (right, green) only *reads* it. **No arrow runs from the LLM agent into the data path.** The single pathway back to the cluster is a human-reviewed pull request (red) picked up on the next config rotation. Planes share a telemetry surface, never a mutation surface.

2.1 Data plane: consensus on Apache Ratis

Reimplementing Raft is a common trap in distributed-systems projects and is not the contribution here. AEGIS instead uses **Apache Ratis** [6] (a production Raft library used by Apache Ozone and IoTDB) as a dependency, wrapped in a Spring Boot component that exposes a narrow four-method state-machine abstraction — `apply`, `query`, `takeSnapshot`, `restoreSnapshot` — to the downstream lock and KV modules. Those modules never touch Ratis's `RaftServer` or `RaftClient` directly. Snapshotting, log replication, leader election, and membership are Ratis's responsibility. The cluster runs five nodes (tolerating two failures), and a killed leader triggers a re-election that an integration test asserts deterministically.

2.2 Locks and leases with fencing tokens

The lock service is Chubby-style. A client first creates a *session* with a TTL; it then acquires named locks scoped to that session. A lock is held until the session's lease expires or the client releases it. The fencing token handed back on acquisition is the **Raft log index** of the acquire command — monotonic by construction across the cluster, so a delayed holder cannot clobber a newer one (Figure 2). Lease expiry is driven by a *replicated tick*, so every replica agrees on which sessions are alive.

Leader-stamped time for determinism. Every lock/lease implementation must compare "now" against stored TTLs. Reading the wall clock inside the state machine's `apply()` would break determinism: two replicas applying the same entry at different instants would compute different deadlines and drift apart. AEGIS therefore *stamps time on the leader side, not the apply side.* Each command carries an explicit `leader_timestamp_ms` in its proto envelope, set from an injectable clock at submission; the state machine reads that field and never calls `System.currentTimeMillis()`.

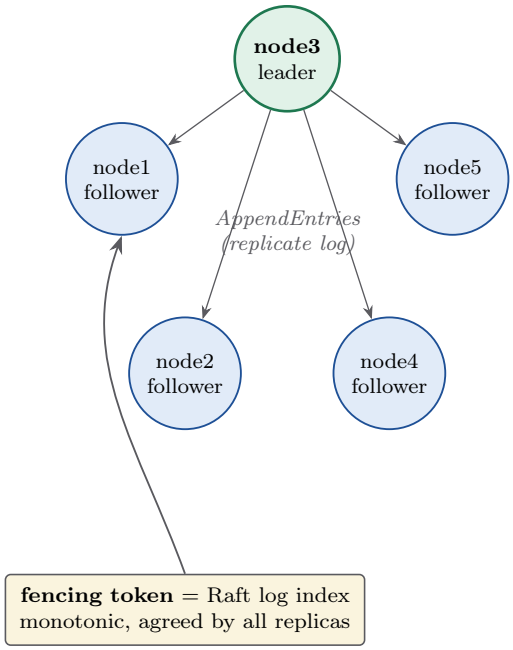

Figure 2: The five-node Raft topology. One leader (green, `node3`) replicates the log to four followers (blue) via AppendEntries; the cluster tolerates two failures. A lock's fencing token is the Raft log index of its acquire command — monotonic and agreed by every replica, so a stale holder can never overwrite a newer one.

Replicas always agree because they all read the same number from the same log entry. A consequence worth noting: clock skew between nodes becomes a *control-plane* concern the classifier flags as `clock_skew`, not a data-plane correctness bug.

2.3 Versioned key–value configuration store

The KV store is etcd-style: linearizable `Put`/`Get`, per-key version history with tombstones, prefix range queries over an ordered map, and `Watch` streams that fan out change events per replica. Watches are designed to survive leader failover.

2.4 Client SDKs

Aegis ships mirror-image Java and Python client SDKs. Both perform multi-endpoint failover (kill any node and the client rolls over to another), renew leases automatically from a background daemon thread, expose scoped-lock ergonomics (a closeable lock that releases on scope exit and surfaces its fencing token), and maintain resilient watch streams across reconnects.

3 The Control Plane and the LLM Invariant

3.1 ADR-001: the LLM is never in the data path

The defining design decision is recorded as ADR-001 and is non-negotiable: *every line of code that touches consensus submission, the state machines, or the gRPC services is LLM-free.* The agent lives in a physically separate Python package that (i) consumes only the *exported* telemetry surface — it has no gRPC client, no KV key, no lock session; (ii) emits structured `Anomaly` records; and (iii) mutates the cluster only by opening pull requests against a versioned configuration repository

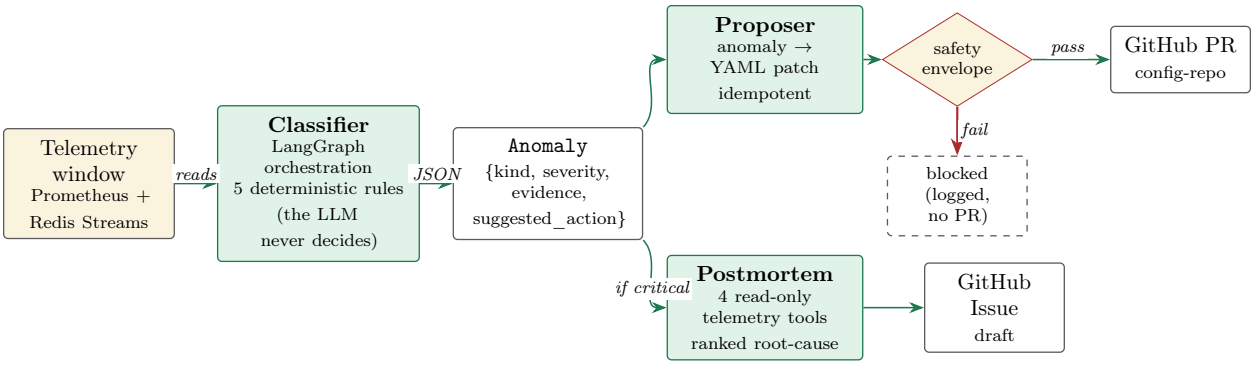

Figure 3: The control-plane pipeline. A telemetry window flows into the deterministic **classifier**, which emits a single `Anomaly` contract {kind, severity, evidence, suggested_action}. That contract drives two branches: the **proposer** turns it into a YAML patch that must pass the deterministic *safety envelope* (diamond) before a pull request is opened — failing patches are logged and never reach the queue; and the **postmortem** drafter uses four read-only telemetry tools to produce a ranked root-cause diagnosis as a GitHub Issue.

(ADR-004, section 3). The package separation makes the boundary auditable: the Python side cannot even *see* the Ratis APIs, so any import of a data-plane stub would be an immediate revert.

The trade-off is explicit and accepted: there is no real-time autonomous remediation. An obvious tuning win sits idle until a human reviews the PR queue. For a coordination service that underpins everything else, paying review latency to keep an LLM hallucination structurally incapable of breaking consensus is the right exchange.

3.2 Two planes, one telemetry surface

The two planes share a *telemetry* surface but never a *mutation* surface (Figure 1). The data plane *exports*: each node publishes Prometheus metrics on `/actuator/prometheus` and pushes Raft events to a Redis Streams topic. The control plane *reads*: the classifier pulls Prometheus and tails the stream. The telemetry path is push/pull-curated, not LLM-queried — the agent consumes a feed, it does not interrogate the cluster.

3.3 The three-stage pipeline

The agent is three composable command-line tools keyed on a single `Anomaly` JSON contract (Figure 3).

Classifier — deterministic heuristics, not the LLM. The classifier consumes a telemetry window and emits one `Anomaly{kind, severity, evidence, suggested_action_class}` per window. LangGraph *orchestrates* the pipeline, but the *decision* is made by five deterministic heuristic rules — one per anomaly kind (`slow_follower`, `lease_starvation`, `clock_skew`, `split_brain_risk`, `election_churn`, plus `none`). Each rule reads concrete signals (per-follower log lag, lease expiration/renewal ratios, cross-node timestamp spread, concurrent leadership claims, term-change counts) and returns a confidence; the graph picks the highest, breaking ties by severity. This keeps the decision reproducible, key-free, and CI-testable — and, crucially, consistent with the invariant: the LLM never classifies.

Proposer — a safety-enveloped, idempotent PR. Given an `Anomaly`, the proposer maps it to a YAML configuration patch (e.g. `slow_follower` → raise the heartbeat interval). The patch is

deep-merged onto the deployed config and the *resulting full config* is run through a deterministic **safety envelope** before any PR is opened. The envelope encodes consensus-safety constraints derived from Raft's timescale separation broadcastTime $\ll$ electionTimeout $\ll$ MTBF [2]. Its flagship rule requires election-timeout-min $\geq K \cdot$ heartbeat-interval with a conservative floor K=3: a heartbeat approaching the election timeout makes followers time out while the leader is healthy, triggering election storms and split-brain. Further rules enforce a valid (ordered, positive) election-timeout randomization range, a positive RPC timeout, leases renewable several ticks before expiry (tick $\leq$ ttl/3), non-negative skew grace, at least one retained KV version, and documented absolute bounds on every numeric field as a backstop against typos and adversarial overflow patches. The verdict aggregates *all* violations in one pass. **Any violation blocks the proposal** — it is logged with the violated rule id and never reaches the PR queue. Safety is thus a property of the code, not a judgement call by a model; unsafe configurations are structurally impossible to propose. The proposer is also *idempotent*: its branch name is `aegis-agent/<kind>/<sha1-of-evidence>`, so re-running the same anomaly reuses one open PR rather than spamming the queue.

Postmortem — a tool-using drafter. The postmortem drafter is triggered by a critical anomaly (or manually). It has a small, fixed budget of **four read-only telemetry tools** — `get_timeline`, `get_raft_term_history`, `correlate_metric`, and `find_similar_past_incident` — each of which answers only from telemetry that already happened; no ad-hoc cluster RPCs are exposed as tools. By default it renders a *deterministic template*, so every operator gets a usable draft even with no API key and no network. An LLM-narrated strategy is opt-in. The diagnoser leads with a *ranked root-cause hypothesis*: the deterministic `HeuristicDiagnoser` reuses the classifier rules, adds cross-signal correlation (e.g. follower lag co-occurring with leadership churn promotes `network_partition` over a plain `slow_follower_disk`), and blends in retrieval of similar past incidents. The optional `RetrievalAugmentedDiagnoser` asks an LLM to rank the hypotheses from the retrieved context, and *gracefully degrades* to the heuristic baseline whenever the model is unavailable.

3.4 ADR-004: a PR-only mutation pathway

The agent has exactly two output channels: **pull requests** against the versioned config repository (the data plane reads its tunables there on rotation) and **GitHub Issues** carrying postmortem drafts. There is no third channel — no admin token, no direct command path. Every agent action is therefore reviewable, revertible, and auditable through ordinary `git` machinery, and the human-in-the-loop is structural rather than aspirational. The cost is latency, accepted in exchange for the safety budget; idempotency on evidence (proposer) and on issue title (postmortem) keeps chronic anomalies from flooding the queue.

4 Telemetry and Observability

The telemetry surface is the only thing the two planes share, so its design is load-bearing. It has two complementary halves.

Prometheus metrics (pull). Each node exposes gauges and summaries on a standard Prometheus scrape endpoint. These cover the Raft cluster state (current term, leadership flag, commit index, last-applied index, and per-follower log lag), the locks-only lease and lock counters, and a gRPC request-latency summary reporting the p50, p90, and p99 quantiles. Table 1 lists the metric names and types.

Table 1: Core telemetry metrics exported at `/actuator/prometheus` on each node.

Metric	Type	Notes
`aegis_raft_term`	gauge	current Raft term
`aegis_raft_is_leader`	gauge	1 if this node is leader
`aegis_raft_commit_index`	gauge	last committed log index
`aegis_raft_last_applied_index`	gauge	last index applied to the state machine
`aegis_raft_log_lag`	gauge	per-follower lag (leader only)
`aegis_lease_renewals_total`	counter	locks module only
`aegis_lease_expirations_total`	counter	locks module only
`aegis_client_request_seconds`	summary	p50/p90/p99 gRPC latency

Redis Streams events (push). When `AEGIS_TELEMETRY_REDIS_URL` is set, a telemetry auto-configuration lights up and a Redis-backed publisher fans every Raft event (term changes, leader changes, sampled log lag) to a Redis Streams topic. Consumers tail it with `XREAD`. This push feed is what lets the classifier work even when Prometheus is momentarily unavailable, and it is what the operator dashboard streams over server-sent events.

Grafana + dashboard. A provisioned Grafana dashboard renders the cluster state (term, role, per-follower log lag, commit index, gRPC latency percentiles). A separate read-only operator dashboard (an Express proxy plus a single-page UI) shows the live five-node topology, a server-sent-events telemetry tail, and a chaos overlay — all strictly read-only consumers of the telemetry surface.

5 Chaos Engineering Harness

To give the agent something real to react to, Aegis ships a chaos harness of Bash fault injectors that operate on a running five-node cluster. The scenarios (Table 2) cover the main coordination-service failure modes: a leader partition (`docker network disconnect`), a slow follower (`tc qdisc netem delay` inside the container), a clock-skew proxy (`docker pause`/`unpause` cycles), a kill-and-restore, a flapping network, and a chained `critical-cascade` that drives the postmortem demo. Each script writes a canonical JSONL line to `chaos/events.jsonl` with an `at` timestamp, a phase (`start`/`end`), and the scenario — that event format, not the underlying chaos primitive, is the load-bearing contract the dashboard correlates against the Redis event log.

Two deliberate choices: `tc`/`docker` primitives rather than Toxiproxy (which would require inserting $5 \times 4 = 20$ proxies between Raft peers — a non-trivial compose rewrite — whereas `tc` runs inside the existing container), and a `CHAOS_DRY_RUN` mode that logs intended commands without touching containers, used as the first invocation and in CI.

6 ConsensusOps-Bench

ConsensusOps-Bench is a reusable benchmark for coordination-service ops agents, released as the byproduct of building the root-cause diagnoser: the labelled incident corpus *is* the benchmark dataset.

Table 2: Chaos scenarios and the classifier output each is designed to trigger.

Script	Mechanism	Expected anomaly
`partition-leader`	network disconnect	`split_brain_risk` → `election_churn`
`slow-follower`	`tc netem` delay	`slow_follower`
`clock-skew`	pause/unpause cycles	`clock_skew`
`kill-and-restore`	kill then start	`election_churn` / `none`
`flapping-network`	repeated disconnect	`election_churn`
`critical-cascade`	chained faults	mixed critical (post-mortem)

Table 3: ConsensusOps-Bench label sets (6 anomaly kinds, 6 root causes) and per-cause counts. 33 incidents total.

Anomaly kind	Root cause	Count
`slow_follower`	`slow_follower_disk`	8
`split_brain_risk`	`network_partition`	5
`clock_skew`	`clock_drift`	5
`election_churn`	`election_storm`	5
`lease_starvation`	`lease_starvation`	5
`none`	`healthy`	5
	Total	**33**

6.1 Corpus

The corpus is **33 labelled Raft incidents** — 7 hand-authored seed fixtures plus 26 deterministically generated synthetic variants (no RNG) — as self-contained JSON files. Each incident carries a ground-truth `classifier_label` (one of 6 anomaly kinds), a ground-truth `root_cause` (one of 6 causes), a severity, and `events` / `metrics` arrays byte-compatible with the chaos traces and the classifier replay format, so an incident can be replayed through the classifier *and* drafted into a postmortem with no translation. The label sets and their counts appear in Table 3.

6.2 Two capabilities, scored separately

The benchmark scores two deliberately distinct capabilities:

1. **Anomaly classification accuracy** — given an incident's telemetry window, does the agent fire the right anomaly *kind*? Scored against `classifier_label`.
2. **Root-cause accuracy** — does the diagnoser rank the right *root cause* at **top-1**, and within **top-k** (k=3)? Scored against `root_cause`.

The separation is principled: a classifier answers *"what signal fired?"* and a diagnoser answers *"what actually caused it?"*. One classifier kind can map to several root causes (a lagging follower may be a slow disk, a saturated link, or a partition) and one root cause can light up several signals — so the corpus labels both, independently.

Table 4: ConsensusOps-Bench scorecard (33 incidents, k=3). The LLM-with-retrieval diagnoser matches the deterministic baseline exactly.

Metric	Deterministic baseline ($0)	LLM-with-retrieval†
Anomaly classification accuracy	**1.000** (33/33)	**1.000** (33/33)
Root-cause top-1 accuracy	**0.970** (32/33)	**0.970** (32/33)
Root-cause top-3 accuracy	**1.000** (33/33)	**1.000** (33/33)

† `google/gemini-3.1-flash-lite` via OpenRouter, 2026-06-01: 51.4 s for all 33 incidents, $0.0088 total. The classifier is deterministic in both columns.

6.3 Pluggable interface

The benchmark scores *any* agent, not just Aegis's. Both scored pieces are swappable: the classifier is any `Callable[[TelemetryWindow], Anomaly]` (only the emitted `Anomaly.kind` is scored) and the diagnoser is any object implementing a `diagnose(*, anomaly, window, source)` → `DiagnosisResult` protocol returning a ranked list of hypotheses. Pass either, both, or neither; an omitted component falls back to the Aegis deterministic baseline for that capability.

7 Results

We evaluate two configurations on the full 33-incident corpus with top-k cutoff k=3: a **deterministic baseline** (Aegis's own `ClassifierGraph` plus `HeuristicDiagnoser`, run fully offline, $0, asserted in CI) and a **LLM-with-retrieval** run (the same deterministic classifier paired with the `RetrievalAugmentedDiagnoser` on `google/gemini-3.1-flash-lite` via OpenRouter, dated 2026-06-01). In *both* columns the classifier is deterministic — the LLM never classifies — so the classification accuracy is identical by construction.

7.1 Headline scorecard

Table 4 and Figure 4 give the result. The deterministic baseline already achieves perfect anomaly classification (33/33) and root-cause top-1 0.970 / top-3 1.000. The LLM-with-retrieval diagnoser **matches the deterministic baseline exactly** — top-1 0.970, top-3 1.000 — completing all 33 incidents in **51.4 s** (≈1.6 s/incident) for a **total API spend of $0.0088** (under one cent).

7.2 Per-class detail

Anomaly classification is perfect across all six classes in both runs: `clock_skew` 5/5, `election_churn` 5/5, `lease_starvation` 5/5, `none` 5/5, `slow_follower` 8/8, `split_brain_risk` 5/5 (Table 5, left). Root-cause top-1 is perfect on five of six causes in both runs — `clock_drift` 5/5, `election_storm` 5/5, `healthy` 5/5, `lease_starvation` 5/5, `network_partition` 5/5 — with a single miss on `slow_follower_disk` (7/8) (Table 5, right).

7.3 The single honest miss

The one top-1 miss is identical in both runs and is reported transparently. On incident `seed-critical_cascade` the ground-truth root cause is `slow_follower_disk`, but the top-1 prediction is `election_storm`; the correct cause is recovered at top-3 (`election_storm`, `network_partition`, `slow_follower_disk`). This is a genuine cascade: a disk-bound follower falls behind,

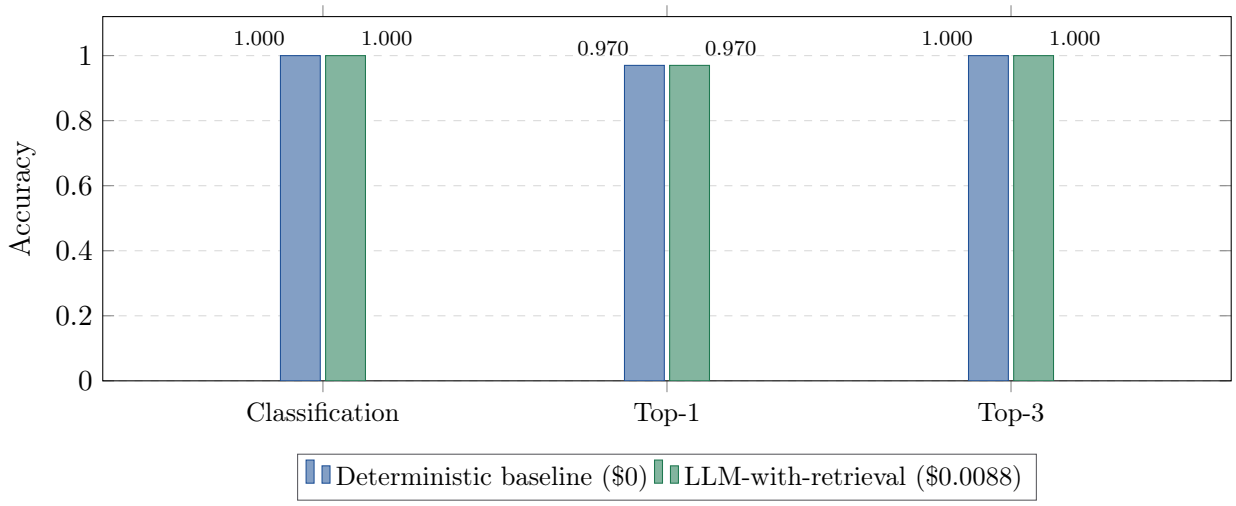

Figure 4: Deterministic baseline vs. LLM-with-retrieval on the three benchmark metrics. The bars are identical: the cheap LLM-with-retrieval diagnoser exactly matches the free deterministic baseline on this corpus, and the classifier (left group) is deterministic in both runs.

Table 5: Per-class anomaly classification (left) and per-cause root-cause top-1 (right). Identical in both runs.

Anomaly kind	
`clock_skew`	5/5
`election_churn`	5/5
`lease_starvation`	5/5
`none`	5/5
`slow_follower`	8/8
`split_brain_risk`	5/5
all	**33/33**

Root cause	
`clock_drift`	5/5
`election_storm`	5/5
`healthy`	5/5
`lease_starvation`	5/5
`network_partition`	5/5
`slow_follower_disk`	7/8
top-1	**32/33**

the resulting instability triggers an election storm, and the *proximate* election signal masks the *root* disk cause in the top-ranked hypothesis. The cross-signal correlation correctly surfaces all three candidates, so the operator sees the true cause within the top-3 ranking. We prefer to keep and explain this miss rather than tune it away — it is exactly the kind of multi-signal incident a benchmark should contain.

7.4 Discussion: why is the LLM here at all?

The headline finding is deliberately uncomfortable for the LLM: on this corpus, a one-cent LLM-with-retrieval diagnoser is *exactly as accurate* as the free deterministic baseline, neither better nor worse, including the same miss. We read this as a positive architectural result rather than a negative one. First, it validates the invariant: the determinism that matters lives in the *free* classifier, and the (metered) LLM is confined to advisory diagnosis where being wrong costs a human a second of review, not a split-brain. Second, it sets an honest bar — on well-structured telemetry with good retrieval, a cheap model suffices, and one should not pay for a frontier model where a Flash model and a heuristic baseline tie. Third, the LLM's value is not captured by accuracy alone: it produces the human-readable narrative postmortem prose around the ranked hypotheses, which the deterministic template renders more rigidly. The benchmark's job is to keep that value claim honest,

and here it does.

8 Related Work

Coordination services. Aegis sits squarely in the lineage of Chubby [1], ZooKeeper [4], and etcd [5]. Chubby established the coarse-grained lock service with sessions and fencing; ZooKeeper generalised it to a hierarchical znode store with watches; etcd paired a Raft core with a versioned KV API and lease-based locks. Aegis adopts the same data-plane semantics — sessions, fencing (here, the Raft log index), versioned KV with watches — and does not claim to advance them. Its contribution is the control plane layered above.

Consensus. The data plane stands on Raft [2], whose explicit goal was understandability, and on the practical-Paxos lessons of "Paxos Made Live" [3], which documents how sensitive a real consensus deployment is to subtle non-determinism — precisely the sensitivity ADR-001 respects. Aegis uses **Apache Ratis** [6] as the Raft implementation rather than reimplementing the algorithm, and the safety envelope's timescale constraints come directly from the Raft paper's broadcastTime $\ll$ electionTimeout $\ll$ MTBF guidance.

LLMs and AIOps. A growing body of work applies LLM agents to operations — incident triage, log analysis, runbook automation, and self-healing frameworks. Some proposals go further and place model reasoning inside critical decision paths. Aegis takes the contrarian, conservative stance for a correctness-critical substrate: the agent observes, recommends, and documents, but its only mutation pathway is a human-reviewed pull request, and unsafe proposals are structurally blocked before review. The closest spirit is operational tooling that augments rather than replaces the human operator (e.g. the postmortem discipline of the Google SRE practice [7]), made concrete here with a released benchmark and a deterministic safety envelope.

9 Conclusion

Aegis demonstrates that an LLM operations agent can deliver real value to a distributed coordination service — anomaly classification, safety-enveloped config proposals, and ranked-root-cause postmortems — *without ever entering the consensus data path.* The architecture makes the invariant structural: two planes share a telemetry surface but never a mutation surface, the classifier's decision is deterministic, the proposer's safety is a property of the code, and the only pathway back to the cluster is a pull request a human merges. On ConsensusOps-Bench, the deterministic baseline is already near-perfect and a one-cent LLM-with-retrieval run matches it exactly — a finding that, rather than embarrassing the LLM, confirms the right division of labour: put the determinism in the free classifier, confine the model to advisory diagnosis, and keep the human on the merge gate.

Limitations and future work. The benchmark is modest in size (33 incidents) and substantially synthetic; broader, real-incident coverage is future work. The agent is observation-only by design — we treat this as a feature, not a limitation, for a correctness-critical substrate. The largest deferred directions are predictive lead-time detection (forecasting a cascade from leading indicators), generalisation of the same agent to a third-party etcd/ZooKeeper cluster's metrics, and counterfactual sandbox verification that embeds an empirical before/after proof in each proposal. None of these requires relaxing ADR-001.

References

[1] M. Burrows. The Chubby lock service for loosely-coupled distributed systems. In *OSDI*, 2006.

[2] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm (Raft). In *USENIX ATC*, 2014.

[3] T. D. Chandra, R. Griesemer, and J. Redstone. Paxos made live: an engineering perspective. In *PODC*, 2007.

[4] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. ZooKeeper: wait-free coordination for internet-scale systems. In *USENIX ATC*, 2010. `https://zookeeper.apache.org`.

[5] The etcd authors. etcd: a distributed, reliable key-value store. `https://etcd.io`.

[6] The Apache Ratis project. Apache Ratis: a Java library implementing the Raft protocol. `https://ratis.apache.org`.

[7] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy (eds.). *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly, 2016.