FROM ASSISTANT TO AUTONOMOUS OPERATOR

KubeIntellect: A Human-Governed AI SRE for Kubernetes

Continuous perception, tiered reasoning, and auditable autonomy for LLM-driven Kubernetes operations

A technical white paper for platform, SRE, DevOps, and cloud-native decision-makers

Mohsen Seyedkazemi Ardebili
Creator, KubeIntellect

Website: https://kubeintellect.com · Live demo: https://kubeintellect.com/demo
Docs: https://mskazemi.github.io/kubeintellect/
Source: https://github.com/MSKazemi/kubeintellect

Natural-language root-cause analysis · Zero-token failure detection · Dry-run diffs with human approval on every write · Four-tier RBAC · Tamper-evident audit and deterministic replay · Runs on AKS / EKS / GKE / Kind.

KubeIntellect turns Kubernetes operations from fragmented, tool-by-tool investigation into guided, explainable, and reviewable workflows — so teams move faster without surrendering control of what changes in production.

> **2:47 a.m.** Your phone lights up: *checkout — 5xx spiking*. You are on call. Terminal open, `kubectl get pods` — one is `CrashLoopBackOff`. You `describe` it, scroll the events, tab to Grafana for memory, tab to Loki for the logs, hold four half-answers in your head, and make the call: raise the memory limit. `kubectl apply`. Watch. Hope. Twenty minutes gone — and tomorrow the same fault will page someone else, who starts again from nothing.
>
> Now imagine that page never fires — because something was already watching, already recognized the failure at zero cost, already gathered the evidence, and is holding a one-line fix and a diff, waiting for you to approve it. That gap — between a tool you drive and an operator that has your back — is what this paper is about.

Executive summary

Large-language-model (LLM) assistants can already answer Kubernetes questions in plain English — *"why is the payments pod crashing?"* — and return a plausible diagnosis. But an assistant is not an operator. Today's LLM assistants are **reactive** (they act only when a human prompts them), **expensive** (they run a single large model on every turn), and **opaque** (they leave no tamper-evident

record of what they did or why). None of those properties is acceptable for a component that is allowed to change a production cluster.

KubeIntellect is the evolution of an LLM Kubernetes assistant into a continuous, cost-aware, and auditable **operator**. It is structured explicitly as a **MAPE-K autonomic-computing loop** — Monitor, Analyze, Plan, Execute, around a shared Knowledge base — over a live Kubernetes cluster. Four mechanisms carry the change from assistant to operator:

1. **A zero-token perception layer** that watches the cluster through streaming `kubectl` events and recognizes known failures with compiled predicates — spending *no* LLM tokens to detect a problem.
2. **A graduated autonomy ladder (A0–A3)**, governed per namespace and by an explicit allowlist, that lets the system open its own investigation *without* a human prompt when a detector fires.
3. **A tiered "Cortex" reasoning graph** that turns triage and evidence-gathering into explicit, streamed steps and reserves a large model for final synthesis.
4. **A hash-chained flight recorder** that makes every decision tamper-evident and deterministically replayable, capturing rollback points for any change it makes.

We evaluated four architectural generations of the system (V1–V4) on a shared **62-scenario fault-injection benchmark** run against a live cluster, scored by a stronger, independent judge model. Under a fair protocol that holds the model fixed across versions, the V4 operator attains the **highest diagnostic quality** (**32.0 of 40** on the independent judge's rubric) at **roughly half the token cost** (**13.4k vs. 25.6k** tokens per scenario), and all three modern generations sharply exceed a capability-maximal thirteen-agent baseline (18.4/40). The result is stated honestly, with its limits (V4 still trails the lean coordinator on end-to-end remediation — see Section 9): KubeIntellect ships its tiered reasoning path gated behind a remediation-parity criterion rather than switching it on prematurely.

What to do next. KubeIntellect is open source and every number here is reproducible from the released harness. The lowest-risk first step is a **read-only evaluation** against your own cluster — no write access, no autonomy — followed by an architecture review. The evaluation ladder is in How to evaluate KubeIntellect.

32.0/40	**13.4k**	**0**	**0.80**
best diagnostic quality of four generations	tokens/scenario — roughly half the lean baseline	tokens to detect a known failure	autonomous/prompted quality ratio

Who this white paper is for

This is a **decision paper for a technical audience**, not a product manual and not an academic manuscript. Different readers can stop at different depths:

If you are...	Read for...	The one thing to take away
A **platform / SRE lead** (decides)	operating leverage and governance	Faster time-to-understanding on incidents *without* ceding control of production changes.
An **SRE** (operates)	how it shortens root-cause analysis	Logs, metrics, events, and config correlated in parallel; you approve any change after seeing the diff.
A **DevOps / platform engineer** (integrates)	tool and workflow fit	It coordinates the `kubectl` / Helm / Prometheus / Loki stack you already run — it does not replace it.
An **architect** (evaluates trust)	boundaries and control paths	Explicit approval gates, four-tier RBAC, hard credential blocks, and a tamper-evident audit trail.

What we assume about you. You run Kubernetes (managed — AKS/EKS/GKE — or self-hosted); you have Prometheus and Loki (or can point KubeIntellect at equivalents); you want to reduce operational toil but require a human in control of mutating actions and a provable record of what happened.

An honesty note up front. KubeIntellect is an open-source system with a published, reproducible benchmark. It is **pre-commercial** — we are seeking design partners, not claiming installed-base traction. Accordingly, every claim in this paper is a **descriptive** capability or an **evaluative** result backed by the released harness; where the system is not yet ahead, we say so plainly (see Section 9).

1. The problem: an assistant is not an operator

Running Kubernetes at scale is a continuous operational burden. Failures are diverse (crash loops, OOM kills, image-pull errors, pending pods, missing service endpoints, admission rejections) and the diagnostic knowledge required to resolve them is spread across the cluster API, metrics, and logs. Today, an on-call engineer bridges those tools by hand:

```
Alert  -->  kubectl get/describe  -->  read logs  -->  open Grafana/PromQL
  |                                                          |
  +------------- correlate in your head <--------------------+
                        |
                decide a fix under pressure  -->  kubectl apply  (hope it worked)
```

Every arrow is a context switch, and the whole chain lives in one engineer's head and terminal history — slow, person-dependent, and unaudited. And it repeats: the same crash loop is re-diagnosed from scratch next week by whoever is holding the pager, because nothing remembered the last fix. The promise of an LLM agent is that it can absorb that knowledge and act as a tireless on-call engineer. But in practice, first-generation LLM operations assistants share three structural weaknesses:

- **They are reactive.** The agent does nothing until a human notices a problem and types a question. A cluster degrading at 3 a.m. is invisible until someone looks. The most valuable operational work — catching a slow-burn failure before it pages anyone — is exactly the work a prompt-only assistant cannot do.

- **They are uniformly expensive.** A naïve agent calls one large, costly model on every turn, including for trivial triage decisions ("is this even a problem?") that a compiled rule could answer for free. Worse, "more agents" is often mistaken for "more capability": a sprawling multi-agent design multiplies token cost without improving answers.
- **They are unauditable.** When an agent touches a cluster, an operator needs to know *exactly* what it did, in what order, on what evidence, and be able to prove the record was not altered after the fact. A chat transcript is not an audit log.

KubeIntellect is designed to remove each of these weaknesses without giving up the natural-language interface that makes an LLM agent useful in the first place. The shift it makes is from a fragmented, manual operating model to a guided one:

Operating step	Manual / tool-by-tool today	KubeIntellect-guided
Detect a problem	Wait for an alert or a human to notice	Zero-token detectors watch the event stream continuously
Gather evidence	Switch between `kubectl`, logs, Grafana	Four specialists fan out over API, metrics, logs, events in parallel
Diagnose	Correlate by hand, under time pressure	One structured root cause with supporting *and* conflicting evidence
Decide a fix	Craft a command from memory	A concrete recommended change, shown as a dry-run diff
Apply	`kubectl apply` and hope	Gated behind explicit human approval; blast-radius actions always confirmed
Prove what happened	Scroll shell history	Hash-chained, replayable flight recorder

2. What KubeIntellect is

KubeIntellect is a **Human-Governed AI SRE for Kubernetes**. You ask questions in plain English; it investigates your cluster with real tools, explains what it found, and — *with your approval* — fixes it. You do not need to know `kubectl`, PromQL, or LogQL syntax.

Every answer is grounded in live data from up to four sources:

Source	What it provides
Cluster API (`kubectl`)	Pods, deployments, services, events, nodes, endpoints, logs, `describe`, specs — read freely; writes are gated.
Helm (read-only)	Release list, values, status, history — to reason about what was deployed.
Metrics (Prometheus)	CPU/memory/throughput/error-rate time series.
Logs (Loki)	Application and system logs over a time window.

For a simple question, you get a direct, plain-English answer. For a genuine outage, KubeIntellect runs a **root-cause analysis (RCA)**: it fans out four specialist agents in parallel — one each for the pod, its events, its logs, and its metrics — and synthesizes their findings into a single struc-

tured conclusion — root cause, confidence, supporting *and* conflicting evidence, reasoning, and a concrete recommended fix.

Beyond answering questions, the V4 operator works *between* your questions: zero-token detection of known failures, self-opened investigations bounded by an autonomy level, a "morning digest" of what happened while you were away, tamper-evident audit replay, and — opt-in — anticipatory detection that warns before a slow-burn failure manifests.

3. What it looks like in practice

Return to the 2:47 a.m. page from the opening. The three walkthroughs below are **illustrative of the documented workflow**, not benchmark results — they show how a request moves from plain English to a governed action. The first is that same incident, handled by the operator. Numbers (limits, replica counts) are examples.

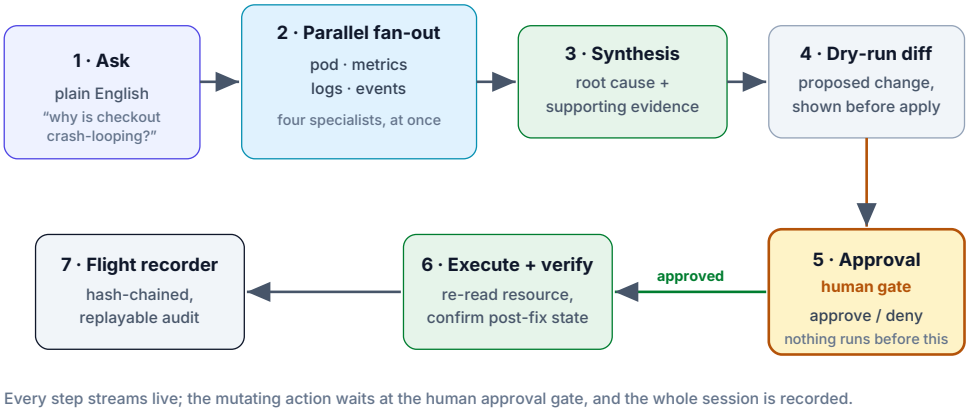

Figure 1: From a plain-English question to a governed, recorded action.

3.1 Incident RCA — a crash-looping pod

> **You ask:** *"Why is the checkout pod crash-looping?"*

KubeIntellect fans out four specialists in parallel: the pod agent reads status and `describe`, the events agent scans recent events, the logs agent pulls the last crash window from Loki, and the metrics agent checks memory against limits in Prometheus. It synthesizes one conclusion:

> **Root cause (confidence: high):** the container is `OOMKilled` — memory usage hit the 256Mi limit during checkout bursts.
> **Supporting evidence:** `Reason: OOMKilled` in events; RSS plateau at the limit in Prometheus; `MemoryError` in logs.
> **Recommended fix:** raise the memory limit to 512Mi.
> **Proposed change (dry-run):**
> `kubectl set resources deploy/checkout --limits=memory=512Mi`
> **Apply this change? [approve / deny]**

Nothing is applied until you approve. After approval, the operator re-reads the resource and re-

ports the *actual* post-fix state ("Pod is now Running — verified"), and the whole session is written to the flight recorder.

3.2 Access-control audit — "who can delete pods in prod?"

> **You ask:** *"Which subjects can delete pods in the* `payments` *namespace?"*

KubeIntellect reads RoleBindings, ClusterRoleBindings, and the referenced roles, and returns a plain-English answer with the exact bindings that grant the verb — a read-only investigation that never touches a write path. Crucially, the agent **cannot read Secrets or ServiceAccounts** (blocked at the tool layer for every role), so an access audit cannot become a credential leak.

3.3 Safe scaling with approval

> **You ask:** *"Scale checkout to handle tonight's launch."*

KubeIntellect proposes `kubectl scale deploy/checkout --replicas=6`, labels it with a risk level, and **stops for approval** before running it. Genuinely destructive actions — deleting a namespace, a PersistentVolume, or a CRD, or draining a node — *always* require confirmation, even if auto-approve is enabled elsewhere. The autonomy ladder (Section 5.2) decides how much of this the system may ever do on its own, per namespace.

4. Architecture: a MAPE-K autonomic loop

KubeIntellect is deliberately organized as a **MAPE-K loop** [8] — the reference model for autonomic (self-managing) computing. Each phase maps to a concrete subsystem, and all phases read from and write to one shared Knowledge base.

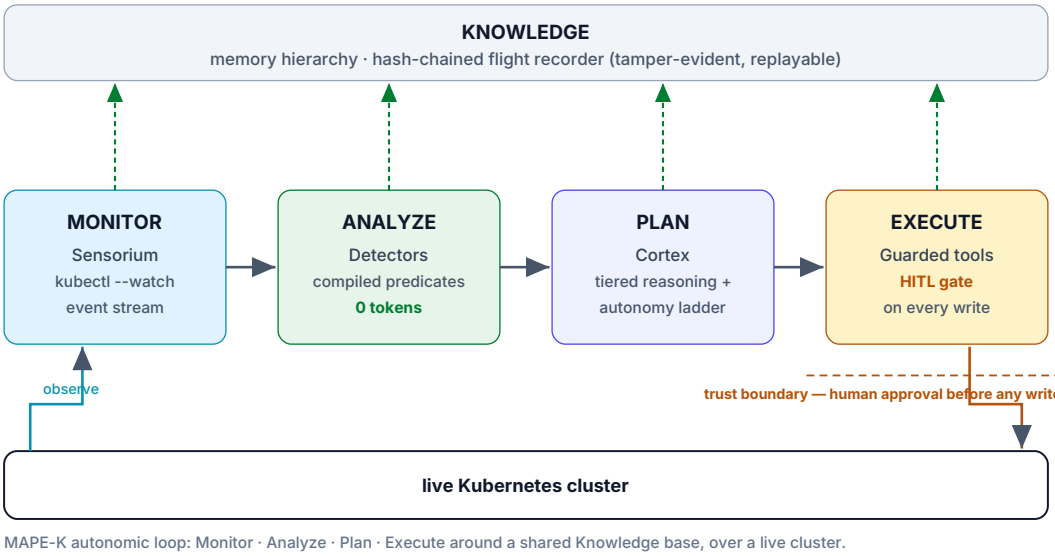

Figure 2: The MAPE-K autonomic loop. Every write crosses a trust boundary that requires human approval.

- **Monitor — Sensorium.** A `kubectl --watch` perception layer streams cluster events continuously.
- **Analyze — Detectors.** Compiled playbook predicates recognize known failure conditions in that stream at **zero LLM cost**.
- **Plan — Cortex + autonomy ladder.** A tiered reasoning graph plans the investigation; the autonomy ladder decides how far it may go on its own.
- **Execute — Guarded tools + HITL.** Actions run through role-checked tools; any mutating command stops for human approval.
- **Knowledge — memory + flight recorder.** A memory hierarchy carries context across turns; a hash-chained flight recorder makes every decision auditable.

Under the hood, the request path is a **FastAPI + LangGraph** service. Queries arrive over HTTP, run as an asynchronous session, and stream typed events (status, tool calls, tokens, plans, approval requests) back to the client over Server-Sent Events. State is checkpointed per conversation (Postgres in production, SQLite locally), so a session can pause at a human-approval gate and resume cleanly.

5. The four pillars

5.1 Zero-token perception (Monitor + Analyze)

The core cost-control idea is simple: **most of the time, deciding whether something is wrong should not require an LLM at all.**

KubeIntellect's Sensorium watches the cluster through streaming `kubectl` events. A library of **compiled detectors** — RE2 predicates over pod status, event reasons and messages, and instant PromQL queries — recognizes the most common Kubernetes failures directly in that stream. A `CrashLoopBackOff`, an `OOMKilled`, an `ImagePullBackOff`, a service that lost its endpoints — each is matched by a predicate that costs **zero generation tokens** to evaluate. Detection is debounced (a condition must hold for a configured interval) to avoid firing on transient churn.

The same knowledge feeds the reasoning path. KubeIntellect ships **18 built-in playbooks** — deterministic investigation recipes covering container-lifecycle, node-and-scheduling, networking, lifecycle/job, and admission failures. When the cluster snapshot matches a pattern, the playbook guides the investigation automatically. The evaluation shows this layer detects every injected fault with **no false firings under benign churn**, at zero token cost.

An **opt-in anticipatory mode** extends detection forward in time: trend predicates fit a least-squares line to a range-PromQL metric, project its ETA to a threshold, and raise a `predicted` finding *before* a slow-burn failure (such as a gradual memory climb toward an OOM) manifests — still at zero token cost. Predictions are capped at autonomy level A1: they may investigate, never auto-fix.

5.2 The autonomy ladder (Plan)

Autonomy is not a single on/off switch — it is a **ladder**, set per namespace:

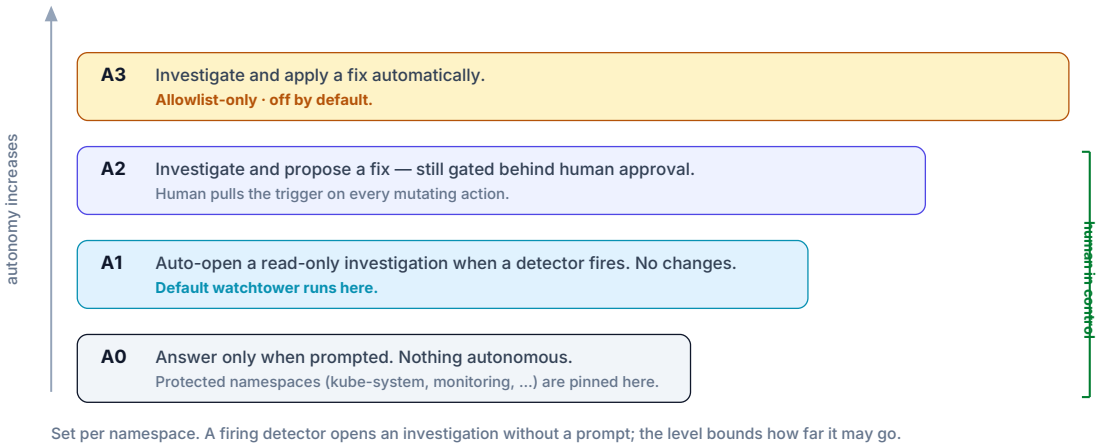

Figure 3: The autonomy ladder (set per namespace). The default watchtower runs at A1; auto-fix (A3) is allowlist-only and off by default.

Level	What the system may do on its own
A0	Nothing autonomous. Answer only when prompted. (Pinned for protected namespaces.)
A1	Open a read-only investigation when a detector fires; publish a report. No changes.
A2	Investigate and propose a fix, still gated behind human approval.
A3	Investigate and apply a fix — **only** for actions on an explicit allowlist.

A firing detector opens an investigation *without any human prompt*, but how far that investigation may go is bounded by the namespace's level. Protected namespaces (`kube-system`, `monitoring`, `kubeintellect`, and others) are pinned to A0. The default watchtower runs at **A1** — it will surface a problem and a recommended fix, but a human still pulls the trigger. This is the mechanism that turns a reactive assistant into a continuous operator while keeping a human firmly in control of change.

5.3 The tiered Cortex (Plan)

The reasoning path is an explicit graph rather than an open-ended agent loop: **triage** → **gather (parallel tool calls)** → **synthesize** → **remember.** Making the steps explicit has three benefits: the user sees a live investigation plan before work begins; step transitions and tokens stream in real time; and, critically, the graph can **route work to different model tiers**. Cheap triage and evidence-gathering can run on a small model, reserving a large model for the final synthesis where reasoning quality matters most.

The measured effect of the leaner, staged design is large: the V4 operator reaches the highest diagnostic quality of any generation at roughly **half** the token cost of the previous lean coordinator (see Section 7). Model tiering is offered but evaluated critically, and the honest finding is a *useful negative*: routing the intermediate steps to a small model (`gpt-4o-mini`) did **not** reduce cost on this workload — the weaker model compensated with more tool-call rounds, *raising* tokens rather than lowering them. The efficiency win therefore comes from the explicit-node **architecture** run

with a uniformly capable model (staged reasoning, pre-fetched evidence, trimmed context), not from swapping in a smaller model. Small/large tiering ships as an optional deployment knob, not a load-bearing cost claim.

5.4 The flight recorder (Knowledge)

Every decision the operator makes is written to a **hash-chained, append-only flight recorder**. Each record links to the previous one by hash, so any after-the-fact edit breaks the chain — the log is **tamper-evident**. For any mutating action, the recorder captures a **rollback point** (the resource state before the change). Two operations turn the log into an operator-grade audit trail:

- **Deterministic replay** (`kq replay <session-id>`) reproduces a session and verifies chain integrity — a non-zero exit means the chain was broken.
- **Grounded postmortems** (`kq postmortem <session-id>`) render the log into a human incident timeline where every line cites its recorded event and the chain validity is stated explicitly.

In evaluation, the hash chain verified on a real autonomous episode and **broke under a single-record tamper** — establishing that the auditability property holds in practice, not just by construction.

5.5 How KubeIntellect compares

The four pillars map onto four axes that separate an *operator* from an *assistant*: continuous zero-token **perception**, graduated bounded **autonomy**, tiered cost-aware **reasoning**, and tamper-evident **auditability**. Surveying the most relevant systems, the pattern is consistent — each owns one or two of these axes; none couples all four, and cost-aware reasoning is essentially absent among Kubernetes tools.

System	Perception	Autonomy	Reasoning	Auditability
K8sGPT [1] / kubectl-ai [2]	✗	✗	✗	✗
HolmesGPT [3]	✗	∼	✗	✗
LLM cost routers (RouteLLM [4], Hybrid LLM [5])	✗	✗	✓	✗
Staged-approval agents (HULA [6])	✗	✓	✗	∼
Autonomous SRE (STRATUS [7])	∼	✗	✗	∼
KubeIntellect V2 (prior lean assistant)	✗	✗	∼	✓
KubeIntellect V4 (operator)	✓	✓	✓	✓

✓ *full* · ∼ *partial* · ✗ *absent*. Two honesties belong with this table. First, it maps **design** coverage, not measured remediation outcome — on end-to-end *fixing*, the lean V2 assistant still leads (see Section 9). Second, V4's auditability is tamper-*evident* against a reader or after-the-fact editor, not tamper-*proof* against an adversary who controls the log writer (Section 9). The contribution is not any single axis in isolation — rule-based detection, model routing, staged approval, and hash-chained logs each exist in prior work — but their **coupling** in one governed operator loop.

5.6 Related work

Positioned against the four axes, the closest systems each own a subset. The Kubernetes LLM assistants — K8sGPT [1], kubectl-ai [2], and HolmesGPT [3] — own natural-language **diagnosis** of a live

cluster, and HolmesGPT adds a measure of investigative autonomy, but none couples cost-aware reasoning or bounded write-autonomy to that diagnosis. The LLM cost routers — RouteLLM [4] and Hybrid LLM [5] — own reasoning-cost, learning to route queries across model tiers, but they are **domain-agnostic**: they carry no cluster perception, no operational autonomy, and no audit trail. Staged-approval agents — HULA [6] — own the human-in-the-loop **approval** gate for a software-development agent, but without a continuous perception layer that opens work on its own. Autonomous-SRE research — STRATUS [7] — explores multi-agent **autonomy** for cloud reliability, but does not couple that autonomy to the tiered cost story or the tamper-evident audit record. Framing all of this, the MAPE-K model [8] supplies the autonomic-computing loop — Monitor, Analyze, Plan, Execute over a shared Knowledge base — that KubeIntellect instantiates.

The pattern is that each prior system owns one or two axes: diagnosis, or reasoning-cost, or approval, or autonomy. KubeIntellect's contribution is not any single one of them — each already exists in the literature above — but the **coupling** of continuous zero-token perception, graduated bounded autonomy, tiered cost-aware reasoning, and tamper-evident auditability in one governed MAPE-K loop.

6. Safety and governance

KubeIntellect is built on the premise that an agent allowed to change a cluster must be governed like one. Safety is layered, not bolted on:

- **Human-in-the-loop (HITL) on every write.** When the agent wants to run a mutating command (`scale`, `patch`, `apply`, `delete`, `rollout`, ...), it stops, shows the exact command and its risk level, and waits for `yes/no`. The LangGraph checkpoint pauses at the approval point and resumes on the human's decision. A small set of cascading-blast actions (delete namespace, delete PV, delete CRD, drain) *always* require confirmation, even in auto-approve mode.
- **Four-tier role-based access control.** API keys map to `readonly`, `operator`, `admin`, or `superadmin`. Read-only keys are rejected before the LLM ever sees a write; each higher tier unlocks a broader, explicitly enumerated set of verbs. The agent cannot act beyond the caller's role.
- **Hard resource and namespace blocks.** The agent **cannot read Secrets or ServiceAccounts** — blocked at the tool layer for every role, so it cannot exfiltrate credentials. Protected namespaces are off-limits for writes by default and pinned to autonomy A0.
- **Post-change verification.** After a change is applied, the agent re-reads the resource and reports the actual post-fix state ("Pod is now Running — verified") rather than assuming success.
- **Tamper-evident audit.** The flight recorder (Section 5.4) provides the record that makes all of the above accountable.

Together these give a defensible answer to the question every platform team asks before letting automation near production: *"What is it allowed to do, and how do I prove what it did?"*

7. Evaluation and results

7.1 Method

KubeIntellect is evaluated on a **62-scenario fault-injection benchmark** run against a live Kubernetes cluster. Each scenario ships an injection manifest, a gold expected diagnosis, and a recovery-verification script. All four generations (V1–V4) run under a **fair, model-controlled protocol**: the same model (uniform GPT-4o for coordinator and sub-agents) across every version, so the com-

parison isolates *architecture*, not model choice. Responses are scored by a **stronger, independent judge model**[1] on a 40-point rubric, with a judge-agreement study to check the judge's reliability, and results are reported with multi-seed confidence intervals and Holm-corrected paired significance tests.

The four generations under test:

- **V1 (maximal):** a capability-maximal thirteen-agent design — the "more agents" hypothesis (27,260 SLOC).
- **V2 (lean):** a lean LangGraph coordinator with a four-way RCA fan-out (5,744 SLOC).
- **V3 (framework):** a framework-based (DeepAgents) spike (5,197 SLOC).
- **V4 (operator):** the MAPE-K operator described in this paper (≈12,800 app SLOC).

7.2 Headline results

Under the fair protocol, per-scenario means (95% CI):

Version	Judge /40	Pass rate	Tokens	Resolution
V1 (maximal)	18.4 [16.3, 20.7]	22.6%	n/a†	34.3%
V2 (lean)	29.3 [28.4, 30.2]	68.0%	25,569	**77.4%**
V3 (framework)	28.3 [26.8, 29.7]	64.8%	47,941	62.5%
V4 (operator)	**32.0 [31.2, 32.9]**	**75.3%**	**13,352**	44.6%

† V1/V3 expose no token telemetry for a subset of cells; V3's cost is dominated by LLM calls (47.9k tokens), not container resources.

The findings, stated honestly:

- **Highest quality at half the cost.** V4 reaches **32.0/40**, the best of any generation, at **13.4k tokens** — roughly half of V2's 25.6k. The quality gain over the lean coordinator is strongly significant over four seeds (Wilcoxon, $p_{\text{Holm}} = 0.0013$).
- **The redesign, not the supervisor, carries the capability.** All three modern generations (V2/V3/V4) sharply exceed the thirteen-agent V1 baseline (18.4/40). Adding agents did not add capability; a leaner architecture did.
- **The gain is axis-dependent — and we do not hide it.** The lean V2 coordinator still leads on end-to-end **remediation** (77.4% vs. 44.6%; McNemar, $p_{\text{Holm}} = 0.032$). KubeIntellect therefore ships V4's tiered Cortex **gated behind a remediation-parity criterion** — the operator's higher *diagnostic* quality does not license switching the reasoning path on until it matches the incumbent on *fixing* things.

"The lean redesign — not the supervisor — carries the system's capability."

[1] The judge is a markedly stronger, higher-capability reasoning model — the same provider as the systems under test but decoupled onto its own resource (in our runs, `gpt-5.4` judging a GPT-4o-family system); the exact judge is recorded in the released harness. Using a stronger, decoupled judge — rather than a model grading its own family — is what the judge-agreement study below validates.

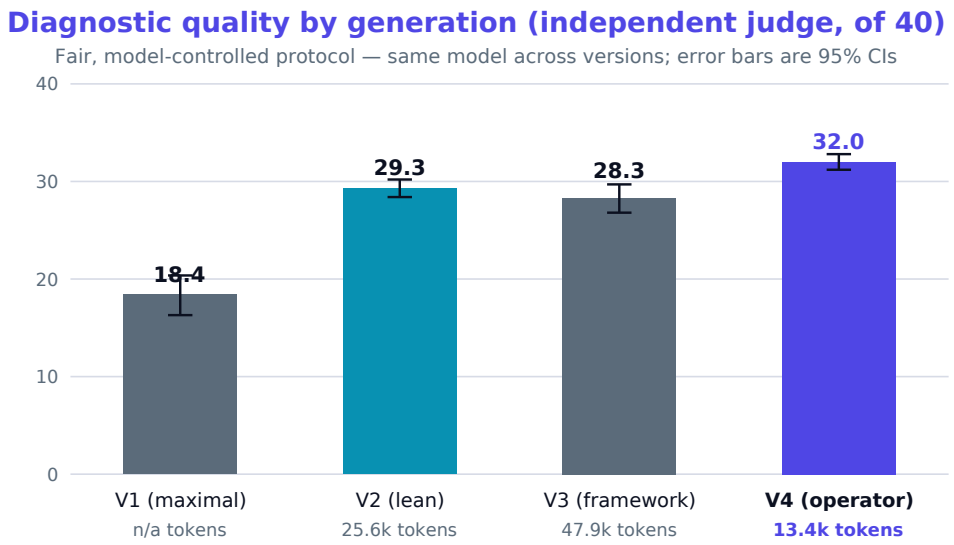

Figure 4: Diagnostic quality (independent judge, of 40) across the four generations under the fair, model-controlled protocol, with 95% confidence intervals and per-scenario token cost. V4 is highest-quality at roughly half the token cost of the lean V2 baseline.

7.3 Autonomy, detection, and audit

- **Autonomous investigations recover 80% of prompted quality.** Investigations opened by a firing detector — with no human prompt — reach a **0.80 autonomous/prompted quality ratio**, above the pre-registered 0.60 gate.
- **Perfect detection on the benchmark, zero cost, no false fires.** On the 62-scenario corpus the perception layer detects every injected fault at **zero generation cost** with **no false firings** under benign churn.
- **Auditability holds in practice.** The flight recorder's hash chain verifies on a real autonomous episode and **breaks under a single-record tamper**.
- **The judge is reliable but stricter.** A within-provider judge-agreement study finds the stronger judge ranks responses consistently (Spearman $\rho = 0.68$) yet is systematically stricter than a weaker same-family judge — which had inflated scores in prior work. This is why the evaluation uses an independent, stronger judge throughout.

7.4 Runtime footprint

On single-node Kind, the operator's steady-state footprint is modest: ≈**0.08 CPU cores** and ≈**260–300 MiB** memory mean over the evaluation window, with zero restarts. Its efficiency advantage is in **tokens**, not container resources — the place where LLM-agent cost actually accrues.

8. Deployment and operations

KubeIntellect deploys as a Helm chart into any Kubernetes cluster (a local Kind cluster is fully supported for evaluation and demos). A single `values.yaml` is the source of truth for configuration: Prometheus/Loki URLs, blocked namespaces and resources, ingress hosts, and the feature flags

that gate each V4 layer.

Observability is shared, not per-instance: one Prometheus/Grafana/Loki stack and one Langfuse instance serve every version, with per-version trace tagging so token and cost can be attributed without standing up separate monitoring. Logs, traces, and metrics are correlatable by a single `session_id`, so any answer can be traced end-to-end from the SSE stream to the underlying LLM calls.

Operators drive the system through the `kq` CLI:

```
kq --query "why is the checkout pod crashing?"   # ask a question
kq findings                                       # zero-token detector firings
kq digest --hours 12                              # what the watchtower did overnight
kq replay <session-id>                            # deterministic audit replay
kq postmortem <session-id>                        # grounded incident timeline
```

Every V4 capability is **feature-flagged**, defaulting to a conservative posture (watchtower at A1, autonomous auto-fix off, experimental memory slices off). Teams adopt autonomy at the pace their governance allows.

9. Limitations and honest positioning

This white paper deliberately states where KubeIntellect is *not* yet ahead:

- **Remediation parity is unfinished — with a known, fixable cause.** The lean V2 coordinator still resolves more scenarios end-to-end than V4's tiered path (77.4% vs. 44.6%). The gap is *not* a diagnostic deficit: the Cortex tends to **present** a correct fix and defer the mutating action rather than apply it. The operator leads on diagnostic quality and cost, so V4 keeps the proven V2 reasoning path as the default until the Cortex meets a resolution-parity criterion. This is a shipped constraint with a concrete path to closing it, not a footnote.
- **Audit is tamper-evident, not tamper-proof.** The hash chain detects any modification, deletion, or reorder of committed records by a reader or after-the-fact editor — and it did break under a single-record tamper in evaluation. It does *not* defend against an adversary who controls the log writer (the hash is unkeyed and lives in a single database); a keyed forward-secure MAC or external anchoring of chain digests would close that gap and is future work. Relatedly, the recorder *captures* a rollback point for every change but does not yet *apply* automatic reversal — autonomous mutation should be enabled only where operator-driven recovery is acceptable.
- **Evaluation is on an injection benchmark, on Kind.** The 62-scenario corpus is broad and reproducible, but it is a fault-injection harness on a single-node cluster, not a multi-tenant production fleet. Runtime-footprint numbers are indicative on Kind.
- **The judge is an LLM.** A stronger, independent judge with an agreement study mitigates self-evaluation bias, but LLM-as-judge remains a proxy for expert human assessment.
- **Autonomy is opt-in for a reason.** Auto-fix (A3) is allowlist-only and off by default. KubeIntellect is designed to earn trust incrementally, not to be handed a production cluster on day one.

Every number in this document is regenerable end-to-end from the released evaluation harness; the exact commands and seeds are documented in the repository.

10. Why it matters

KubeIntellect demonstrates that an LLM agent can cross the line from *assistant* to *operator* without abandoning the properties operations teams require:

- **Continuous, not reactive** — it watches, and it acts between prompts, bounded by an explicit autonomy ladder.
- **Cost-aware, not uniformly expensive** — zero-token detection and a staged, tiered reasoning graph deliver the best measured quality at roughly half the token cost of the prior lean design.
- **Auditable, not opaque** — a hash-chained flight recorder makes every decision tamper-evident and replayable, with rollback points for every change.
- **Governed, not unfettered** — four-tier RBAC, hard credential blocks, human-in-the-loop on every write, and post-change verification.

The evidence is presented with its limits intact: the operator wins on diagnostic quality and cost, it does not yet win on end-to-end remediation (Section 9), and it ships its newest reasoning path gated behind a parity criterion rather than switched on for its own sake. That honesty is itself part of the design philosophy — an operator you can trust with a cluster is one whose claims you can check.

Picture on-call six months from now. The 2:47 a.m. page from the opening mostly does not come, because the operator caught the memory drift at 2:10 and left a fix waiting for morning. When a page *does* come, it arrives with the evidence already gathered, a one-line change already written, and an audit trail already recording — and all you do is read it and approve. That is the shift from firefighting to operating. It is available today, one read-only evaluation away.

How to evaluate KubeIntellect

KubeIntellect is open source; you can assess it on your own terms, at zero commitment, in roughly increasing order of investment:

1. **Start a read-only evaluation.** Deploy with a `readonly` API key against a non-production cluster. The agent can investigate and explain but **cannot make any change** — the safest possible first look. Local Kind is fully supported.
2. **Run the incident-workflow demo.** See a diagnosis, dry-run diff, and approval gate end-to-end at `https://kubeintellect.com/demo`.
3. **Reproduce the benchmark.** Run the released 62-scenario harness yourself — every number in Section 7 is regenerable end-to-end, seeds and commands included.
4. **Request an architecture review or design-partner conversation.** For a deeper look at trust boundaries, the autonomy model, and fit with your governance — or to shape the roadmap as an early design partner.

Get started:
Source and harness — `https://github.com/MSKazemi/kubeintellect`
Docs — `https://mskazemi.github.io/kubeintellect/`
Website — `https://kubeintellect.com` · Contact — hello@kubeintellect.com.

Availability

KubeIntellect and the full evaluation suite are released as open source at https://github.com/MSKazemi/kubeintellect. The repository contains the four architectural generations (V1–V4) behind a shared streaming contract, the complete evaluation harness (scenario runner, signal collectors, automated scorer, LLM rubric judge, multi-seed statistics, runtime-footprint collector), and the 62-scenario fault-injection corpus. Every result in this paper is reproducible end-to-end from the released harness.

This white paper summarizes, for a technical and decision-making audience, the peer-reviewed study "From Assistant to Autonomous Operator: Continuous Perception, Tiered Reasoning, and Auditable Autonomy for LLM-Driven Kubernetes Operations." For the full method, statistics, and related-work positioning, see the academic manuscript in the project repository.

References

The comparison in Section 5.5 ("How KubeIntellect compares") refers to systems [1]–[7]; [8] is the autonomic-computing reference model this paper is organized around.

[1] K8sGPT Authors. *K8sGPT: Giving Kubernetes Superpowers to Everyone.* CNCF Sandbox project. https://github.com/k8sgpt-ai/k8sgpt

[2] Google Cloud. *kubectl-ai: AI-powered Kubernetes Assistant*. https://github.com/GoogleCloudPlatform/kubectl-ai

[3] Robusta Dev. *HolmesGPT: AI Agent for On-Call Engineers and Cloud-Native Troubleshooting.* https://github.com/robusta-dev/holmesgpt

[4] I. Ong, A. Almahairi, V. Wu, W.-L. Chiang, T. Wu, J. E. Gonzalez, M. W. Kadous, and I. Stoica. RouteLLM: Learning to Route LLMs with Preference Data. *arXiv:2406.18665*, 2024.

[5] D. Ding, A. Mallick, C. Wang, R. Sim, S. Mukherjee, V. Rühle, L. V. S. Lakshmanan, and A. H. Awadallah. Hybrid LLM: Cost-Efficient and Quality-Aware Query Routing. *ICLR*, 2024 (arXiv:2404.14618).

[6] W. Takerngsaksiri et al. Human-In-the-Loop Software Development Agents. *arXiv:2411.12924*, 2025.

[7] Y. Chen, J. Pan, J. Clark, Y. Su, N. Zheutlin, B. Bhavya, R. Arora, S. Jha, T. Xu, and Y. Deng. STRATUS: A Multi-agent System for Autonomous Reliability Engineering of Modern Clouds. *NeurIPS*, 2025.

[8] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, 2003.