Int. j. adv. multidisc. res. stud. 2026; 6(4):1114-1121

Received: 16-06-2026
Accepted: 26-07-2026

International Journal of Advanced Multidisciplinary Research and Studies

ISSN: 2583-049X

OpenReliOps: A Safety-Constrained Architecture for Autonomous Reliability in Open-Weight Model Services

[1] Prudvi Saisaran Ponduru, [2] Pavani Priya Vyshnavi Nandanavanam, [3] Sai Kesav Kumar Ponduru

[1] Department of Computer Science, University of the Potomac, Washington, Washington D.C, United States of America
[2] Department of Computer Science, California State University, Northridge, CA, United States of America
[3] Department of Business Analytics, University of Amherst, Amherst, MA, United States of America

DOI: https://doi.org/10.62225/2583049X.2026.6.4.6707
Corresponding Author: **Prudvi Saisaran Ponduru**

Abstract

Open-weight language models enable private deployment, local adaptation, and independent audit, but they also transfer responsibility for model artifacts, inference infrastructure, telemetry, security, and generated actions to the deploying organization. This paper presents OpenReliOps, a self-healing control architecture that treats an open-weight model as a replaceable reasoning component inside an evidence-grounded and policy-bounded reliability loop. The architecture integrates service-level-objective forecasting, topology-aware telemetry retrieval, causal root-cause diagnosis, a typed repair domain-specific language, independent policy verification, calibrated escalation, progressive execution, rollback, and post-action semantic and operational validation. A reliability process preference optimization objective is defined to learn from successful, rejected, failed, and rolled-back incident trajectories without rewarding unsafe outcome-only behavior. Formal analysis establishes policy confinement and blast-radius bounds under complete mediation, current state, typed actions, and sound declared policies. The evaluation protocol separates diagnostic correctness, evidence grounding, policy safety, and measured service recovery, and specifies comparisons on RCAEval, OpenRCA, OpenRCA 2.0, AIOpsLab, and controlled model-serving faults. The analytical result is that model substitution alone cannot provide autonomous reliability: the reliability boundary is created by the interfaces among evidence, uncertainty, policy, execution, and verification. The framework is therefore positioned as a falsifiable methodological contribution whose operational gains must be demonstrated through incident-level experiments, ablations, multiple seeds, and complete failure reporting.

Keywords: Open-Weight Language Models, Self-Healing Systems, AIOps, Root Cause Analysis, Safe Remediation, Autonomous Reliability

1. Introduction

Open-weight language models make it possible to host model parameters locally, choose the serving stack, inspect artifacts, apply quantization, fine-tune on private operational data, and retain detailed audit records. These advantages are important for regulated and latency-sensitive environments, but they remove the managed-provider boundary that previously absorbed part of the operational risk. The deploying organization becomes responsible for weights, tokenizers, quantization recipes, custom kernels, schedulers, caches, retrieval stores, credentials, rollback procedures, and the consequences of model-generated actions [9-13, 18-24].

The term self-healing is often used too loosely. A process restart is local recovery, not a complete self-healing system. A root-cause ranking is diagnosis, not repair. A model that generates a command without independent authorization is unsafe automation rather than autonomous reliability. A defensible self-healing system must detect and predict failures, reconstruct a causal explanation, propose a bounded action, verify policy and current state, execute progressively, measure service recovery, roll back when necessary, and retain both success and failure as auditable learning evidence [1-6].

Recent benchmarks demonstrate why this separation is necessary. OpenRCA evaluates long-context root-cause analysis over heterogeneous enterprise telemetry; RCAEval provides hundreds of reproducible microservice failures and multiple baselines; OpenRCA 2.0 introduces step-wise causal annotations; and AIOpsLab supports deployable services, workloads, fault injection, telemetry export, and interactive agent evaluation [3-6]. These resources expose a central failure mode: a model may name a

plausible component without grounding the diagnosis in the actual propagation path, or may propose an action that is technically plausible but operationally prohibited.

This study asks three questions. First, what architecture is required to make an open-weight model service self-healing without granting the model unrestricted control-plane access? Second, what algorithms can combine telemetry, causal diagnosis, preference learning, and calibrated escalation while preserving deterministic safety boundaries? Third, how should autonomous reliability be evaluated so that diagnostic fluency is not mistaken for safe and effective recovery? The contributions are a provider-neutral architecture, a typed repair language, two formal safety properties, a reliability-specific trajectory objective, and a pre-registered evaluation protocol.

2. Materials and Methods

2.1 Research design and evidence base

The study follows a design-science methodology. Requirements were derived from site reliability engineering, model-serving systems, root-cause analysis benchmarks, uncertainty calibration, preference optimization, open-weight model reports, quantization research, and AI risk guidance [1-34]. Sources were included when they provided an implementable mechanism, a reproducible benchmark, a formal reliability concept, or an authoritative operational control. Marketing claims and unverifiable performance assertions were excluded. Recent preprints are clearly identified in the reference list and are used only where peer-reviewed alternatives do not yet cover the emerging benchmark or model family.

The output is a falsifiable system design rather than an unexecuted performance claim. The architecture is specified through typed interfaces, state assumptions, equations, algorithms, safety propositions, baselines, ablations, and reporting requirements. This makes it possible for a future implementation to confirm or reject each claimed benefit independently.

2.2 Failure model and reliability objectives

The system boundary includes the open-weight model artifact, tokenizer, quantization, inference runtime, retrieval and tool interfaces, model-serving infrastructure, observability plane, policy bundle, and state-changing control adapters. Failures are classified by the layer in which the fault originates and by the evidence required for repair.

Table 1: Failure taxonomy for self-healing open-weight model services

Failure layer	Representative failure	Required evidence	Bounded recovery
Artifact integrity	Corrupt, incompatible, poisoned, or unverified weights; tokenizer or quantization mismatch	Hash and signature failure; semantic regression; load error	Quarantine artifact; restore known-good revision; revalidate
Serving infrastructure	GPU out-of-memory, scheduler saturation, cache exhaustion, replica or node failure	Latency, memory, queue depth, health, topology	Scale, restart, drain, re-route, or switch model under bounds
Semantic behavior	Hallucination surge, quality drift, prompt-policy mismatch, unsafe tool intent	Task quality, safety tests, judge disagreement, calibration drift	Fallback model; prompt or policy rollback; disable risky tool class
Retrieval and tools	Stale index, missing evidence, tool outage, poisoned telemetry	Freshness, citation coverage, tool status, provenance	Rebuild or isolate index; use alternate source; abstain
Configuration and change	Bad deployment, stale policy, topology drift, incompatible dependency	Change events, version graph, admission state	Progressive rollback; invalidate stale approval; reconcile desired state
Adversarial input	Prompt injection, telemetry injection, denial-of-service pattern	Input anomaly, policy conflict, untrusted source indicator	Sanitize, isolate, rate-limit, escalate; never bypass verifier

Let q(t) in [0,1] denote a user-facing service-level indicator and s the corresponding objective. The normalized error-budget burn rate is:

$$B(t) = [1 - q(t)] / [1 - s] \quad (1)$$

A burn rate above one consumes the permitted error budget faster than the long-term objective allows. OpenReliOps predicts whether B(t+h) will cross a policy threshold within horizon h and estimates the available lead time. Reliability is not accepted from the forecast alone; the full incident outcome also includes diagnostic correctness, evidence grounding, action admissibility, recovery time, rollback, and semantic health.

2.3 OpenReliOps self-healing architecture

Fig 1 presents the proposed architecture. The open-weight model never receives direct control-plane credentials. It is enclosed by a reliability-aware retriever, a structured diagnostician, a typed planner, an independent verifier, an escalation controller, a least-privilege executor, and a post-action validator. Immutable versions of the model, tokenizer, quantization, prompt template, retrieval index, topology snapshot, and policy bundle are recorded for every incident trajectory.

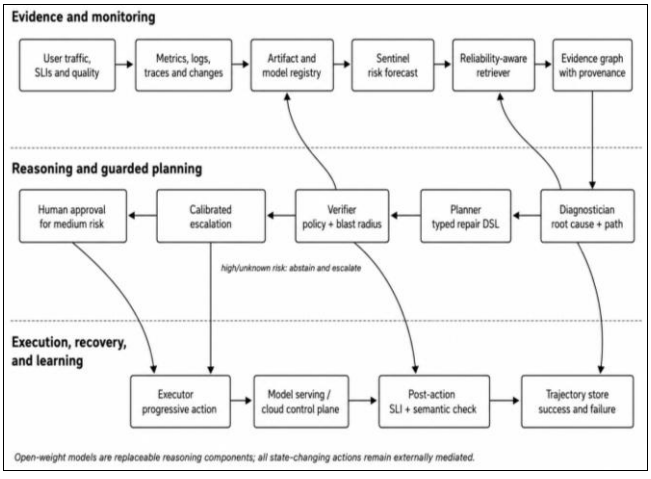

Fig 1: Provider-neutral OpenReliOps architecture for self-healing open-weight model services

Table 2: Separation of duties and interfaces in OpenReliOps

Component	Primary inputs	Primary output	Safety boundary
Sentinel	SLIs, quality probes, serving telemetry, changes	Forecast burn-risk and severity	No write access
Retriever	Metrics, logs, traces, topology, artifacts, runbooks	Compact evidence graph with provenance	Token, freshness, and source constraints
Diagnostician	Evidence graph and structured hypotheses	Root-cause set, fault type, causal path, citations	Cannot execute actions
Planner	Diagnosis, allowed action vocabulary, runbooks	Typed candidate repairs and rollback triggers	No free-form shell commands
Verifier	Current policy, ownership, topology, quotas, budgets	Approve, require human approval, or reject	Independent of planner
Executor	Signed action object and short-lived identity	Progressive bounded execution trace	Complete mediation and least privilege
Validator	Post-action SLIs, quality, safety, and cost	Close, rollback, or re-enter diagnosis	Recovery must be measured
Trajectory store	All successes, failures, abstentions, and rollbacks	Auditable learning evidence	No deletion of negative outcomes

2.4 Reliability-aware retrieval and causal diagnosis

Raw telemetry is converted into a time-localized evidence graph G=(V,E). Vertices represent services, replicas, nodes, model revisions, tokenizers, storage dependencies, network edges, retrieval sources, and change events. Edges encode calls, containment, resource dependence, temporal precedence, and candidate fault propagation. Each evidence item e receives the score:

$$Score(e) = w1T(e) + w2S(e) + w3R(e) + w4C(e) + w5D(e) - w6U(e) \quad (2)$$

T is temporal alignment with the anomaly window; S is symptom proximity; R is graph reachability to the violated SLI; C is change recency; D rewards corroboration across independent sources; and U penalizes uncertainty, staleness, or untrusted provenance. Retrieval remains bounded by latency and context budgets, but the provenance pointer is preserved so that every diagnostic claim can be traced to evidence.

The Diagnostician outputs a root-cause set, fault type, propagation path, and cited evidence. A diagnosis is grounded only when the path starts at an admissible cause, terminates at the observed user-facing symptom, and every edge is supported by telemetry or topology. Exact root-cause recovery, any-hit recovery, path node and edge F1, and citation precision and recall are therefore reported separately [3-5].

2.5 Typed repair language and independent verification

The Planner cannot emit an executable shell command. It compiles one or more candidate repairs into a restricted domain-specific language (DSL) with explicit parameters, preconditions, expected effect, rollback trigger, evidence identifiers, and a maximum affected set. The action vocabulary is intentionally smaller than the underlying cloud or orchestration API.

Table 3: Illustrative repair DSL; production action vocabularies should be narrower

Typed action	Purpose	Required guards	Rollback
rollback_model(route, revision, traffic)	Return traffic to a known-good model artifact	Signed artifact; quality floor; canary limit	Re-promote only after validation
switch_model(route, fallback)	Move requests to a warmed fallback model	License approval; capacity; semantic compatibility	Restore primary after canary
scale(service, delta, ceiling)	Increase or reduce serving capacity	Quota; ownership; saturation evidence; max delta	Restore previous replica target
restart(selector, rate)	Restart a bounded subset of replicas	Healthy peers; disruption budget; rate limit	Stop sequence and restore traffic
drain(node, budget)	Evacuate a suspected node or accelerator	Capacity reserve; zone balance; workload mobility	Uncordon after verification
isolate_source(index_or_tool)	Disable a stale or poisoned retrieval/tool source	Alternative evidence; incident record; time limit	Re-enable after integrity checks
open_ticket(severity, evidence)	Escalate without changing state	Minimum evidence package	Not applicable

The Verifier evaluates service ownership, identity, allowlists, maintenance windows, deployment state, model and tokenizer compatibility, capacity, disruption budgets, replica and zone minima, dependency health, rollback availability, and action-specific risk ceilings. Its output is ternary: approve, require human approval, or reject. Approval is bound to current policy and topology versions; any relevant state change invalidates the signature before execution.

2.6 Confidence-calibrated escalation

The escalation controller combines model uncertainty, evidence coverage, novelty, estimated blast radius, and policy ambiguity:

$$Risk = b0 + b1(1 - confidence) + b2(1 - coverage) + b3(novelty) + b4(blast\ radius) + b5(policy\ ambiguity) \quad (3)$$

Thresholds theta_auto and theta_human partition the decision space. Risk below theta_auto is eligible for

automatic progressive execution; intermediate risk requires human approval; and risk above theta_human abstains and escalates. Thresholds are selected using a calibration set and selective-risk or conformal procedures so that the observed error among automatically executed actions can be bounded at a declared level [14-16]. Calibration must be repeated after material changes to the model, policy, topology, retrieval corpus, or workload.

2.7 Reliability Process Preference Optimization

OpenReliOps introduces Reliability Process Preference Optimization (RPPO), a trajectory-ranking objective specialized for operational recovery. Let tau denote an incident trajectory. Its utility is:

$$U(tau) = aD + bG + cS + dI - eL - fC – gH \quad (4)$$

D measures diagnostic correctness, G evidence grounding, S policy safety, I measured post-action improvement, L operational latency, C computational and human cost, and H is a hard penalty for unsafe or policy-violating behavior. An unsafe action cannot be offset by a later recovery. Preference pairs are formed from successful and failed trajectories, verifier rejections, human-reviewed runbooks, benchmark ground truth, abstentions, and rollbacks. The optimization can be implemented with pairwise preference methods related to PPO or direct preference optimization, but the reward decomposition and hard safety penalty remain specific to reliability [17,18].

The learning store retains negative evidence. A trajectory with a correct root cause but an unsupported path is inferior to a grounded trajectory; a plausible plan rejected by policy is retained as a negative example; and an admitted action that fails to improve the SLI is not labeled successful. This prevents the training process from rewarding outcome-only fluency.

2.8 Formal safety properties

The safety analysis uses four explicit assumptions: complete mediation of every state-changing action; a typed action space; policy soundness relative to the declared policy set; and state freshness, so that approvals become invalid after relevant policy or topology changes. These assumptions do not cover a compromised executor or privileged actors outside the control loop.

Proposition 1 (policy confinement): Under complete mediation, typed validation, sound current policy, and state freshness, OpenReliOps cannot execute an action outside the verified DSL and current policy set.

Proof sketch: The Executor rejects non-DSL objects and any object without a verifier signature bound to the current policy and topology versions. Sound predicates reject prohibited objects, and a relevant state change invalidates stale signatures. Complete mediation excludes alternate model-controlled write paths. Therefore, every OpenReliOps action is typed and verified.

Proposition 2 (blast-radius bound): If every action declares its affected set and the Verifier enforces global disruption constraints over unavailable and in-flight resources, action-induced unavailability cannot exceed the configured replica and zone bounds.

Proof sketch: Admission succeeds only when the union of currently unavailable resources, in-flight affected sets, and the candidate set satisfies the disruption predicate. Conflicting actions are serialized or rate-limited. The action-induced unavailable set is therefore bounded, assuming accurate state and excluding exogenous failures after admission.

These propositions establish confinement, not repair effectiveness. A policy-safe action can still be ineffective or harmful to service quality without violating the declared policy set. Operational success is therefore accepted only when the relevant user-facing SLI and semantic quality improve within a verification window and no secondary safety constraint regresses.

2.9 Reproducible evaluation protocol

The primary experiment must exercise the complete incident loop. A fault is injected before or at early symptom onset. The Sentinel receives only the permitted pre-impact window and predicts an SLO-burn crossing. The Diagnostician receives a bounded evidence graph and must recover the ground-truth cause and propagation path. The Planner produces typed candidates; the Verifier rejects, requests approval, or signs one action; the Executor applies a progressive step; and the Validator measures whether service and semantic health improve. A run is not successful merely because a service was named or a command completed.

Table 4: Evaluation matrix required to substantiate autonomous reliability claims

Claim	Dataset or harness	Primary measures	Required comparisons
Predictive reliability	AIOpsLab traces and held-out model-serving faults	AUPRC; false-positive rate at fixed recall; median lead time; expected calibration error	Rules/statistical detector; same model without change/topology features
Diagnosis	RCAEval and OpenRCA	Top-k; exact set; component/time/reason accuracy	Published RCA baselines; single-agent LLM; tool-matched agent
Evidence grounding	OpenRCA 2.0	Path exact match; node/edge F1; citation precision/recall	Outcome-only prompting; no causal supervision; no topology retrieval
Policy safety	AIOpsLab plus policy mutation and adversarial plans	Unsafe-action rate; policy violations; rejection/approval/abstention rate	Free-form planner; DSL without independent verifier; rule-only planner
Reliability outcome	Interactive mitigation runs	MTTR; SLO-burn area; rollback; secondary-SLI and semantic regressions	Human approval only; rule remediation; single-agent end-to-end
Artifact resilience	Corrupted/mismatched artifacts and quantized variants	Detection latency; false restore; semantic regression; recovery time	Checksum only; no semantic canary; no artifact lineage
Cross-environment transfer	Leave-one-environment-out deployment	Relative degradation; calibration drift; adapter failure rate	Provider-specific prompting; no canonical ontology

All methods must receive identical incidents, tool interfaces, context budgets, action vocabularies, and evaluation windows. The study should use multiple seeds, paired incident-level outcomes, bootstrap confidence intervals, effect sizes, and predeclared primary metrics. Failed, rejected, rolled-back, abstained, and human-escalated runs must remain in the denominator. Model-family comparisons must hold the scaffold constant so that gains are not attributed to a model when they arise from privileged tools or larger context budgets.

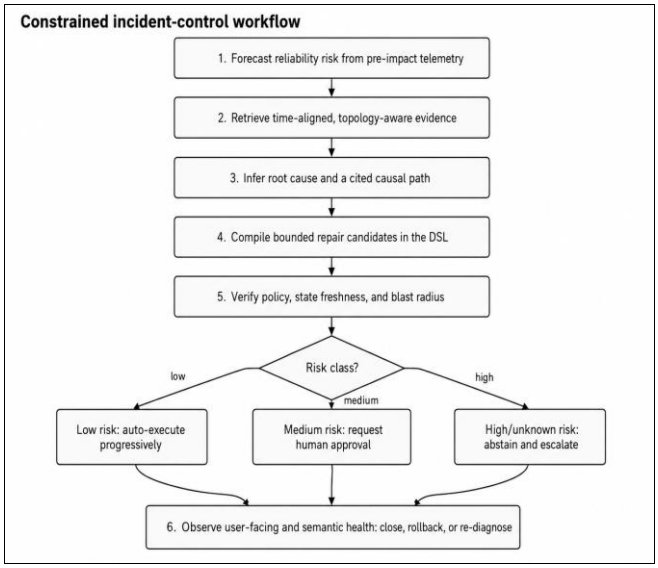

Fig 2: Constrained incident-control workflow with automatic, approval, and abstention regions

3. Results and Discussion

3.1 Analytical Result: Reliability is an interface property

The principal analytical result is that open weights are an enabling deployment property rather than a sufficient reliability mechanism. Llama, Mistral, Mixtral, Falcon, Qwen, or another model family may instantiate the Diagnostician or Planner, but replacing one model with another does not eliminate stale authorization, unsupported causal stories, unsafe tool use, or ineffective remediation [19-23]. Reliability is created by the interfaces that bind evidence to diagnosis, diagnosis to typed action, action to current policy, execution to bounded identity, and recovery to post-action verification.

This distinction also separates serving performance from incident reliability. Clockwork, ORCA, InferLine, AlpaServe, Splitwise, Vidur, and vLLM improve predictability, provisioning, batching, parallelism, phase separation, simulation, and memory management [7-13]. Their telemetry and failure modes are essential inputs to OpenReliOps, but an efficient serving engine is not a causal diagnostician or a safe control plane.

3.2 Analytical result: Four claims must be measured independently

Table 5: Non-equivalent claims in self-healing model operations

Claim	Primary question	Failure hidden by one accuracy score
Diagnostic correctness	Is the identified cause equal to ground truth?	A correct component may be guessed for the wrong reason
Evidence grounding	Does cited telemetry support the complete propagation path?	A fluent explanation may omit or invent causal links
Policy safety	Is the repair authorized, current, and within disruption bounds?	A technically plausible action may violate ownership or risk policy
Reliability outcome	Did service and semantic health actually improve?	A correct and safe action may still fail operationally

The separation prevents a common evaluation error: reporting task completion as autonomous reliability. A model can identify one correct service while failing to recover the full root-cause set, cite the wrong evidence, or execute a safe but ineffective action. The four claim families must therefore be reported with distinct denominators and uncertainty intervals.

3.3 Benchmark-grounded evidence

Published evidence supports the difficulty of the proposed problem but does not constitute an OpenReliOps result. OpenRCA reports 335 failures from three enterprise systems and more than 68 GB of heterogeneous telemetry [3]. RCAEval provides 735 failure cases and fifteen reproducible baselines across metric, trace, and multi-source settings [4]. OpenRCA 2.0 contains 500 causally annotated instances and reports, across eleven frontier models, 20.7% average exact root-cause-set recovery, 76.0% any-hit identification, and 61.5% successful grounding of a correct service through a verified causal path [5]. These statistics motivate separate grounding metrics and topology-aware retrieval.

AIOpsLab demonstrates that cloud environments can be deployed, loaded, fault-injected, observed, and exposed to agents through a common evaluation framework [6]. This makes it possible to measure post-action outcomes rather than ending the study at a text diagnosis. The proposed protocol extends that principle to model artifacts, tokenizer and quantization mismatches, semantic health, and policy mutation.

3.4 Falsifiable hypotheses and ablations

Table 6: Pre-registered hypotheses and decisive ablations

Hypothesis	Expected effect	Decisive ablation
H1	Topology- and change-aware retrieval improves causal-path edge F1 more than simple any-hit RCA	Remove topology and change features while holding model and context budget constant
H2	Typed DSL plus independent verification reduces unsafe-action rate without eliminating all recovery gains	Compare free-form planning, DSL without verifier, and full system
H3	Calibrated escalation lowers risk among automatically executed actions at the cost of more approvals and abstentions	Disable calibration; compare risk-coverage curves
H4	Failed-trajectory preference learning improves grounding and post-action success more than generic instruction tuning	Remove RPPO or discard rejected/rolled-back trajectories
H5	Artifact lineage plus semantic canaries reduce false recovery after quantization or tokenizer changes	Remove artifact checks or semantic validation
H6	A canonical reliability ontology reduces cross-environment degradation relative to provider-specific prompting	Remove ontology and test leave-one-environment-out transfer

A publishable empirical result must report the full ablation matrix and all failure categories. It must not remove incidents in which the system abstains, requests human approval, rolls back, or executes an admissible action that fails to improve service. The protocol is useful only if it can falsify the claim that OpenReliOps improves autonomous reliability.

3.5 Cost, security, and governance trade-offs

Quantization, sharding, caching, and longer contexts can improve deployability or reasoning quality, but they change the reliability envelope. Quantized weights are distinct artifacts that require separate loading, semantic, calibration, and performance regression tests [24-27]. Tensor and pipeline parallelism can enlarge the failure domain, while longer contexts increase memory pressure and may amplify denial-of-service or prompt-injection risks. Every optimization should therefore be evaluated against both performance and incident-recovery metrics.

Operational autonomy also creates a threshold trade-off. Lower execution thresholds reduce approval burden and may shorten recovery, but they increase the consequence of calibration error. Higher thresholds reduce risk but may collapse the system into a recommendation tool. The correct operating point depends on service criticality, action class, error budget, evidence coverage, and the organization's change-management policy.

Open-weight deployment expands the supply chain to weights, tokenizers, custom code, kernels, serialization formats, retrieval corpora, fine-tuning data, and policy bundles. NIST AI RMF emphasizes governance, validity, reliability, safety, security, resilience, accountability, and continuous monitoring [28]. OpenReliOps therefore requires artifact provenance, least-privilege identities, tamper-evident logs, input sanitization, policy-as-code, rollback, and network boundaries that prevent the language-model service from bypassing mediation [29-34].

3.6 Limitations and threats to validity

Table 7: Principal threats to validity and mitigation strategies

Threat	Risk	Required mitigation
No completed OpenReliOps benchmark results	The architecture may not yield operational gains	Treat the paper as a methodological and formal contribution; execute the integrated study before claiming improvement
Benchmark representativeness	Public incidents may not reflect proprietary topologies or organizational policy	Add a supplementary external-validity deployment without replacing public baselines
Telemetry leakage	Agents may access ground truth or post-impact clues	Enforce time cutoffs, sandbox sources, audit tool calls, and publish access rules
Policy incompleteness	A verifier can only enforce encoded policy	Version policies, red-team action classes, and report residual risk
State staleness	Topology may change between verification and execution	Bind approval to state versions and revalidate immediately before execution
Scaffold confounding	Tools, prompts, or context budgets may favor one model	Hold interfaces and budgets constant and publish complete configurations
Cross-environment loss	Canonicalization may remove provider-specific evidence	Preserve provenance and report raw, normalized, and adapter-level errors
Judge bias	Automated semantic evaluation may be unreliable	Prefer deterministic tests and benchmark ground truth; use blinded human review when required

The framework also does not guarantee repair effectiveness from safety alone. Policy confinement prevents a class of unauthorized actions, but a policy-safe action may still be useless, slow, or damaging to semantic quality. The validator and rollback path therefore remain mandatory even when the action has passed deterministic admission.

4. Conclusion

This paper presented a rigorous architecture, algorithmic framework, and evaluation protocol for self-healing open-weight model services. OpenReliOps combines burn-risk forecasting, reliability-aware retrieval, causal diagnosis, a typed repair DSL, independent policy verification, calibrated escalation, progressive execution, rollback, semantic and operational validation, and learning from successful and failed incident trajectories. Formal analysis bounds what the system may execute under explicit assumptions, while the evaluation protocol determines whether those bounded actions are useful.

The central conclusion is conservative: autonomous reliability is not produced by model substitution or by open weights alone. It emerges from complete mediation, evidence provenance, calibrated uncertainty, deterministic policy checks, current state, bounded execution, and post-action verification. Diagnostic correctness, evidence

grounding, policy safety, and service improvement must remain separate outcomes. Until the integrated experiment is executed, OpenReliOps should be presented as a falsifiable methodological contribution rather than evidence of reduced downtime.

5. References

1. Beyer B, Jones C, Petoff J, Murphy NR, editors. Site Reliability Engineering: How Google Runs Production Systems. Sebastopol, O'Reilly Media, 2016.
2. Beyer B, Murphy NR, Rensin DK, Kawahara K, Thorne S. The Site Reliability Workbook. Sebastopol, O'Reilly Media, 2018.
3. Xu J, Zhang Q, Zhong Z, He S, Zhang C, Lin Q, *et al*. OpenRCA: Can large language models locate the root cause of software failures? International Conference on Learning Representations, 2025.
4. Pham L, Zhang H, Ha H, Salim F, Zhang X. RCAEval: A benchmark for root cause analysis of microservice systems with telemetry data. Companion Proceedings of the ACM Web Conference, 2025, 777-780. Doi: 10.1145/3701716.3715290
5. Fang A, Yang Y, Shang J, Lu Q, Xu J, Wang R, *et al*. OpenRCA 2.0: From outcome labels to causal process supervision. arXiv preprint arXiv:2606.27154, 2026. Doi: 10.48550/arXiv.2606.27154
6. Chen Y, Shetty M, Somashekar G, Ma M, Simmhan Y, Mace J, *et al*. AIOpsLab: A holistic framework to evaluate AI agents for enabling autonomous clouds. Proceedings of Machine Learning and Systems. 2025; 7.
7. Gujarati A, Karimi R, Alzayat S, Hao W, Kaufmann A, Vigfusson Y, *et al*. Serving DNNs like Clockwork: Performance predictability from the bottom up. 14th USENIX Symposium on Operating Systems Design and Implementation, 2020, 443-462.
8. Yu GI, Jeong JS, Kim GW, Kim S, Chun BG. Orca: A distributed serving system for transformer-based generative models. 16th USENIX Symposium on Operating Systems Design and Implementation, 2022, 521-538.
9. Crankshaw D, Sela G, Mo F, Zumar C, Gonzalez JE, Stoica I, *et al*. InferLine: Latency-aware provisioning and scaling for prediction serving pipelines. ACM Symposium on Cloud Computing, 2020.
10. Li Z, Zheng L, Zhong Y, Liu V, Sheng Y, Jin X, *et al*. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. 17th USENIX Symposium on Operating Systems Design and Implementation, 2023, 663-679.
11. Patel P, Choukse E, Zhang C, Shah A, Goiri I, Maleki S, *et al*. Splitwise: Efficient generative LLM inference using phase splitting. 51st Annual International Symposium on Computer Architecture, 2024. Doi: 10.1109/ISCA59077.2024.00019
12. Agrawal A, Kedia N, Panwar A, Mohan J, Kwatra N, Gulavani B, *et al*. Vidur: A large-scale simulation framework for LLM inference. Proceedings of Machine Learning and Systems. 2024; 6.
13. Kwon W, Li Z, Zhuang S, Sheng Y, Zheng L, Yu CH, *et al*. Efficient memory management for large language model serving with PagedAttention. ACM Symposium on Operating Systems Principles, 2023, 611-626.
14. Guo C, Pleiss G, Sun Y, Weinberger KQ. On calibration of modern neural networks. Proceedings of the 34th International Conference on Machine Learning, 2017, 1321-1330.
15. Geifman Y, El-Yaniv R. Selective classification for deep neural networks. Advances in Neural Information Processing Systems. 2017; 30.
16. Angelopoulos AN, Bates S. A gentle introduction to conformal prediction and distribution-free uncertainty quantification. Foundations and Trends in Machine Learning. 2023; 16(4):494-591.
17. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.
18. Rafailov R, Sharma A, Mitchell E, Manning CD, Ermon S, Finn C. Direct preference optimization: Your language model is secretly a reward model. Advances in Neural Information Processing Systems. 2023; 36.
19. Dubey A, Jauhri A, Pandey A, Kadian A, Al-Dahle A, Letman A, *et al*. The Llama 3 herd of models. arXiv preprint arXiv:2407.21783, 2024.
20. Jiang AQ, Sablayrolles A, Mensch A, Bamford C, Chaplot DS, De Las Casas D, *et al*. Mistral 7B. arXiv preprint arXiv:2310.06825, 2023.
21. Jiang AQ, Sablayrolles A, Roux A, Mensch A, Savary B, Bamford C, *et al*. Mixtral of experts. arXiv preprint arXiv:2401.04088, 2024.
22. Almazrouei E, Alobeidli H, Alshamsi A, Cappelli A, Cojocaru R, Debbah M, *et al*. The Falcon series of open language models. arXiv preprint arXiv:2311.16867, 2023.
23. Qwen Team. Qwen2.5 technical report. arXiv preprint arXiv:2412.15115, 2024.
24. Frantar E, Ashkboos S, Hoefler T, Alistarh D. GPTQ: Accurate post-training quantization for generative pre-trained transformers. International Conference on Learning Representations, 2023.
25. Lin J, Tang J, Tang H, Yang S, Chen W, Wang W, *et al*. AWQ: Activation-aware weight quantization for LLM compression and acceleration. Proceedings of Machine Learning and Systems. 2024; 6.
26. Dettmers T, Pagnoni A, Holtzman A, Zettlemoyer L. QLoRA: Efficient finetuning of quantized LLMs. Advances in Neural Information Processing Systems. 2023; 36.
27. Frantar E, Alistarh D. SparseGPT: Massive language models can be accurately pruned in one-shot. Proceedings of the 40th International Conference on Machine Learning, 2023.
28. National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework (AI RMF 1.0). Gaithersburg, NIST, 2023. Report No: NIST AI 100-1. Doi: 10.6028/NIST.AI.100-1
29. Open Source Initiative. The Open Source AI Definition 1.0, 2024 [Cited 2026 Jul 30]. Retrieved from: https://opensource.org/ai/open-source-ai-definition.
30. OWASP Foundation. OWASP Top 10 for Large Language Model Applications, 2025 edition, 2025 [Cited 2026 Jul 30]. Retrieved from: https://genai.owasp.org/llm-top-10/.
31. Amodei D, Olah C, Steinhardt J, Christiano P, Schulman J, Mane D. Concrete problems in AI safety. arXiv preprint arXiv:1606.06565, 2016.

32. Basiri A, Behnam N, De Rooij R, Hochstein L, Kosewski L, Reynolds J, *et al*. Chaos engineering. IEEE Software. 2016; 33(3):35-41.
33. OpenTelemetry Project. Semantic conventions, 2026 [Cited 2026 Jul 30]. Retrieved from: https://opentelemetry.io/docs/specs/semconv/.
34. Open Policy Agent Project. Open Policy Agent documentation, 2026 [Cited 2026 Jul 30]. Retrieved from: https://www.openpolicyagent.org/docs/