Int. j. adv. multidisc. res. stud. 2026; 6(5):111-120

Received: 14-07-2026
Accepted: 24-08-2026

International Journal of Advanced Multidisciplinary Research and Studies

ISSN: 2583-049X

RACER: Remediation Assurance Contracts for Evidence-Gated, Risk-Bounded Autonomous Recovery in Cloud Systems

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

[1] Department of Computer Science, University of the Potomac, 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.2026.6.5.6834
Corresponding Author: **Prudvi Saisaran Ponduru**

Abstract

AI agents can diagnose cloud incidents, synthesize operational commands, and invoke state-changing APIs, but a plausible remediation is not necessarily safe to execute. This study presents RACER, a runtime-assurance mechanism that treats every AI-generated repair as an untrusted proposal until it is bound to a machine-checkable Remediation Assurance Contract. The contract specifies preconditions, protected invariants, exact resource scope, a risk and blast-radius budget, evidence requirements, verification predicates, rollback obligations, and a validity interval. RACER combines these contracts with a risk-bounded authorization policy that selects denial, human escalation, staged canary execution, or direct bounded execution while separating the proposing model from authorization, actuation, and post-action verification. Conditional properties are derived for policy confinement, declared blast-radius enforcement, and bounded violation duration under explicit mediation and rollback assumptions. A reproducible Monte Carlo decision study evaluates 1,000,000 simulated incident-remediation pairs across 20 independent seeds. Under the stated synthetic model, unsafe global actuation is 20.14% for ungated execution, 5.48% for fixed canary execution, and 1.33% for RACER; RACER recovers 82.21% of simulated incidents while autonomously handling 80.57%. Ablations attribute the modeled improvement to risk-based escalation, staged execution, and independent verification. The results are mechanism-level simulation evidence, not production-cloud evidence, and a Kubernetes/AIOpsLab validation protocol is specified for deployment-grade testing.

Keywords: Autonomous Remediation, Runtime Assurance, Cloud Reliability, AIOps, Agentic AI, Risk-Bounded Execution

1. Introduction

Cloud platforms already automate many forms of recovery. Orchestrators reconcile desired state, restart failed containers, reschedule workloads, and maintain replica counts. Such mechanisms are powerful precisely because the recovery rules and action spaces are narrow and explicit. By contrast, a contemporary AI operations agent can interpret heterogeneous telemetry, infer a root cause, choose among tools, generate parameters, edit configuration, and invoke privileged APIs. That flexibility enlarges both the space of incidents the system may address and the space of failures it can create.

Recent work on autonomous clouds has therefore shifted from isolated anomaly detection toward end-to-end agents that participate throughout the incident lifecycle [1, 2]. Benchmarks such as OpenRCA and RCAEval expose the difficulty of root-cause analysis from operational telemetry [3, 4]. More general agent benchmarks likewise show persistent failures in long-horizon reasoning and tool use [5, 6]. These results motivate a distinction that is easy to blur in system design: *an agent's ability to recommend a remediation is not evidence that the remediation should be allowed to execute globally*.

Consider four operationally distinct errors. First, a correct repair may be generated from a stale topology snapshot and therefore target resources that have changed. Second, a repair may be locally correct but violate a global invariant such as minimum regional redundancy. Third, the selected action may be reversible in principle but lack the state capture required for an actual rollback. Fourth, the proposing agent may judge its own action successful because the target metric improved while a dependent service silently degraded. None of these failure modes is resolved solely by improving root-cause accuracy.

Autonomic-computing research has long treated adaptation as a controlled feedback process [7]. Recovery blocks separated computation from acceptance testing [8]; the Simplex architecture separated a high-performance controller from a trusted

safety controller [9]; and shielding and barrier-function approaches established ways to constrain learned or optimizing controllers [10, 11]. Self-adaptive-systems research has further examined decision making under partial observability and model uncertainty [12, 13]. RACER applies the underlying separation-of-concerns principle to state-changing AI operations: the generative planner proposes; a narrower assurance substrate authorizes, bounds, observes, and reverses.

The present work also deliberately separates itself from the authors' prior architecture-level work. SAFE-HealCloud introduced a broad safety-aware healing loop with graduated autonomy [14]; OpenReliOps included policy verification, progressive execution, rollback, and post-action validation for model-serving reliability [15]; and a separate framework distinguished runtime self-healing from slower self-evolution [16]. RACER does not claim those high-level patterns again. Its research object is the *authorization boundary*: a typed contract whose clauses can be checked independently of the model that proposed the action and whose risk budget controls the extent of autonomous actuation.

This article asks four research questions:

RQ1: What information must be made explicit before an AI-generated remediation can be treated as an executable object rather than a natural-language recommendation?

RQ2: How can uncertainty, criticality, reversibility, evidence sufficiency, and observability be converted into an execution decision and a bounded rollout scope?

RQ3: What conditional guarantees can an assurance layer provide without claiming that an uncertain planner itself is formally safe?

RQ4: Under a transparent synthetic incident model, how do policy gating, canarying, independent verification, and selective escalation affect unsafe actuation and recovery outcomes?

The contributions are:

1. A typed **Remediation Assurance Contract** containing preconditions, invariants, scope, blast-radius budget, evidence requirements, verification predicates, rollback obligations, and expiry;
2. A **risk-bounded authorization rule** that maps proposals to deny, escalate, canary, or execute decisions and couples estimated risk to rollout scope;
3. A **planner-independent execution lifecycle** in which authoritative state, policy checks, bounded actuation, verification, and rollback do not depend on the planner agreeing with the safety decision;
4. Three **conditional assurance properties** clarifying exactly what can and cannot be guaranteed under complete mediation and rollback assumptions;
5. A **reproducible million-incident simulation** with ablations, released as a deterministic script with per-seed outputs; and
6. A **deployment-grade validation protocol** specifying how to test the same hypotheses in AIOpsLab and Kubernetes rather than extrapolating synthetic percentages to production systems.

1.1 Background and Related Work

1.1.1 Autonomic and Self-Adaptive Systems

The autonomic-computing vision established self-configuration, self-healing, self-optimization, and self-protection as responses to increasing management complexity [7]. Modern self-adaptive systems extend this tradition with explicit models of goals, environment, uncertainty, and adaptation. Garcia *et al*. model partially observable satisfaction of non-functional requirements and reason about uncertain effects of adaptation actions [12]. Casimiro *et al*. combine probabilistic model checking with adaptation of ML-based systems in the presence of model mispredictions [13]. Work on predicting non-functional requirement violations similarly seeks proactive evidence for adaptation decisions [17].

RACER is complementary. It does not introduce a new planner for choosing an optimal adaptation. It defines what must be true *after a planner has proposed an action but before that action is allowed to mutate system state*. This makes the mechanism applicable to rule-based planners, LLM agents, learned policies, or human-authored automation.

1.1.2 Runtime Assurance and Safety Filters

Recovery blocks combine alternative implementations with an acceptance test [8]. Simplex architectures permit a sophisticated controller to operate while a simpler safety controller can take over [9]. Shielding prevents a learned policy from selecting actions that violate formal constraints [10], while control barrier functions provide a mathematical mechanism for maintaining safe state sets in control problems [11].

Cloud remediation differs from continuous control in several respects: actions can be discrete and heterogeneous, consequences can propagate through service dependencies, rollback may require captured state, and observations are delayed or incomplete. RACER therefore adopts the architectural principle of a narrow safety boundary without claiming that its current risk score constitutes a control-theoretic safety certificate.

1.1.3 AI Agents for Operations

Shetty *et al*. identify autonomous cloud operations as a high-impact agentic-AI use case and articulate requirements for design and evaluation [1]. AIOpsLab operationalizes this agenda by deploying microservice environments, injecting faults, exporting telemetry, and providing interfaces for agents [2]. OpenRCA evaluates LLM root-cause localization on large operational datasets [3], while RCAEval provides 735 labeled failure cases from three microservice systems and a reproducible evaluation environment [4].

These environments are important because they expose diagnostic uncertainty, but diagnosis accuracy alone does not measure unsafe actuation, rollback correctness, or false recovery. RACER's evaluation protocol therefore treats diagnostic systems as replaceable proposal generators and measures the downstream execution boundary separately.

1.1.4 Tool-Using Agent Safety

AgentBench reports significant variation in agent performance across interactive environments and highlights long-horizon reasoning, decision making, and instruction following as recurring difficulties [5]. ToolEmu focuses specifically on high-stakes tool use; its experiments identify consequential safety failures even among strong agents [6]. These findings support RACER's core assumption that tool authority should not be granted solely on the basis of model confidence or linguistic plausibility.

1.1.5 Relationship to Prior Work by the Authors

The relationship to the authors' earlier work is intentionally explicit to avoid novelty inflation. SAFE-HealCloud describes an end-to-end safety-aware self-healing

architecture [14]. OpenReliOps describes policy-constrained reliability for open-weight model services and already includes progressive execution, verification, and rollback [15]. The unified self-evolving/self-healing framework addresses a broader dual-loop architecture [16].

RACER's new contribution is the formalization of the *remediation object* and its authorization semantics. The contract makes preconditions, invariants, evidence, scope, consequence budget, verification, rollback, and freshness independently inspectable. The risk budget then controls both whether autonomy is allowed and how much of the target population may be exposed in a stage. No architecture-level claim from the prior papers is presented here as new.

2. Materials and Methods

2.1 Problem Setting and Design Goals

2.1.1 Operational Model

Let a managed cloud system have state $x_t \in \mathcal{X}$ at time t. State includes workload placement, configuration, topology, health, SLO measurements, security-relevant context, and any durable data needed to reason about restoration. Let the observation bundle be:

$$E_t = \{M_t, L_t, T_t, C_t, H_t\}$$

Where M denotes metrics, L logs, T traces, C configuration/topology, and H relevant operational history. An AI planner π_θ proposes action a_t from E_t:

$$a_t = \pi_\theta(E_t)$$

Without an assurance boundary, a tool-using architecture may effectively realize:

$$x_{t+1} = F(x_t, a_t)$$

Which gives the stochastic planner direct influence over production state.

RACER replaces that path with:

$$d_t = G(x_t, E_t, a_t, \mathcal{C}_t)$$

Where $d_t \in \{\text{deny,escalate,canary,execute}\}$ and $\mathcal{C}_t$ is the assurance contract. Only a typed, authorized action reaches the executor.

2.1.2 Threat Model

RACER assumes that the planner can be wrong, overconfident, underspecified, or affected by misleading evidence. The planner is not treated as malicious by default, but the mechanism must remain useful even when a proposal is semantically incorrect. RACER also assumes that some observations can become stale between planning and execution.

The trusted computing base is narrower: authoritative state retrieval, policy evaluation, typed-action validation, scope enforcement, telemetry collection used by the verifier, and the rollback controller. The properties in the conditional assurance analysis below do *not* hold if these components are bypassed or compromised. RACER therefore reduces the authority of the generative component; it does not solve credential compromise, telemetry forgery, or unsound policy specifications by itself.

2.1.3 Design Goals

G1—Complete mediation: Every state-changing action originating from the AI workflow must traverse the same authorization path.

G2—Explicit evidence: The contract must reference operational evidence and authoritative state rather than rely on a free-form explanation as proof.

G3—Bounded consequences: An accepted proposal must have an exact resource scope and a maximum exposure per execution stage.

G4—Independent success criteria: Verification predicates must be bound before actuation and evaluated independently of the planner's post-hoc narrative.

G5—Reversibility awareness: The system must distinguish an action that is syntactically reversible from one for which the required restoration state actually exists.

G6—Human accountability: Autonomous execution should decline as uncertainty, impact, or irreversibility rises. Human escalation is an explicit operating mode, not a failure of the architecture.

2.2 RACER Architecture

Fig 1 shows the trust boundary. The AI planner is outside the trusted execution core. Its output must be parsed into a typed action and bound to a contract. The assurance boundary refreshes authoritative state, applies deterministic hard constraints, evaluates evidence and risk, and chooses an execution mode. The bounded executor enforces target scope. A separate verifier evaluates predetermined predicates using fresh observations. Verification failure triggers the rollback controller when rollback is valid.

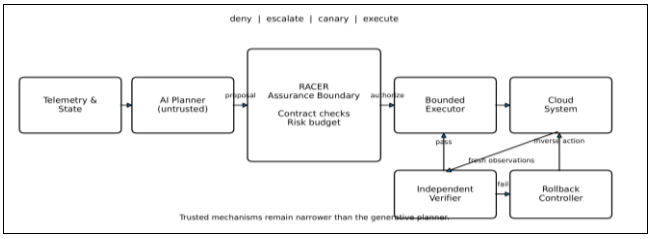

Fig 1: RACER separates an uncertain AI planner from the mechanisms that authorize, bound, verify, and reverse state-changing actions

2.2.1 Typed Remediation Action

Before risk reasoning, an AI response is normalized into an action schema. A minimal action representation is:

$$a = \langle \text{verb,resource,parameters,identity,nonce} \rangle$$

For example, "restart checkout" is not executable because the resource and allowed population are ambiguous. A typed representation can identify a deployment, namespace, replica selector, maximum concurrent disruption, and idempotency key. The executor rejects free-form shell fragments unless an organization deliberately defines a constrained shell action type with its own policy.

2.2.2 Remediation Assurance Contract

For proposed action a, RACER defines:

$$\mathcal{C}(a) = \langle P, I, S, B, E, V, R, \Delta \rangle$$

Where P are preconditions, I protected invariants, S exact scope, B blast-radius/risk budget, E evidence requirements,

V verification predicates, R rollback obligations, and Δ the validity interval. Table 1 summarizes the clauses.

Table 1: Core clauses in a Remediation Assurance Contract

Clause	Purpose	Example machine-checkable condition
P	Preconditions	observed deployment UID and version still equal the diagnosed state
I	Protected invariants	available replicas $\geq r_{\min}$; at least one cross-region replica remains
S	Exact scope	namespace, workload UID, label selector, and maximum number of targets
B	Consequence budget	estimated harm exposure for this stage $\leq B_t$
E	Evidence	minimum evidence sufficiency and required telemetry sources are present/fresh
V	Verification	target SLO improves and dependency/security predicates remain satisfied
R	Rollback	snapshot/inverse operation exists and restoration preconditions hold
Δ	Freshness	contract age remains below the action-specific validity interval

Preconditions P

Preconditions capture facts that must be true immediately before execution. Examples include the diagnosed revision still being active, the workload UID still matching, no conflicting incident being open, or a backup being available. RACER refreshes authoritative state after planning and evaluates:

$$P(x_t) = 1$$

If a precondition fails, the existing contract is invalidated rather than silently rewritten.

Protected invariants I

Invariants state conditions that the execution substrate is not authorized to violate. Examples include minimum replica availability, regional redundancy, prohibition of destructive storage operations, or a maximum unavailable percentage. Invariants are stronger than a planner preference: they are evaluated by the assurance layer.

Scope S

S binds the exact resources that the executor may touch. If $Targets(a)$ denotes the resources selected by a concrete API call, the executor requires:

$$Targets(a) \subseteq S$$

This guards against target drift and prevents an underspecified recommendation from expanding its own authority at execution time.

Risk and Blast-Radius Budget B

Let $C(a) \in [0,1]$ denote normalized consequence severity, $\hat{p}_u(a)$ an estimate that the proposed remediation is unsafe, and $q \in (0,1]$ the fraction of the eligible target population exposed in a stage. A simple expected-exposure proxy is:

$$\mathcal{H}(a,q) = \hat{p}_u(a)\, C(a)\, q$$

A stage is admissible only if:

$$\mathcal{H}(a,q) \leq B_t$$

Where B_t is an organization-defined remediation-risk budget. This expression is not asserted to be a universally calibrated probability of loss; it is a control surface that makes the consequence assumption explicit and testable.

Evidence E

Evidence is a structured set of references to metrics, traces, logs, configuration diffs, deployment history, dependency information, or incident matches. RACER computes or receives an evidence sufficiency score $\epsilon_t \in [0,1]$ and enforces action-specific minimum evidence. A low score can force escalation independent of model confidence:

$$\epsilon_t < \epsilon_{\min} \Rightarrow d_t = \text{escalate}$$

Verification V

Verification predicates are bound before execution. A composite verifier can require:

$$V = V_{\text{SLO}} \wedge V_{\text{health}} \wedge V_{\text{dependency}} \wedge V_{\text{security}}$$

This prevents a planner from redefining success after observing the consequences of its own action. For example, reduced checkout latency is insufficient if payment errors rise above a protected threshold.

Rollback R

Rollback records the restoration state, inverse action, preconditions, and post-rollback validation. An action can be semantically reversible while operationally non-reversible if the previous configuration or data state has not been captured. RACER therefore assigns lower reversibility to actions whose inverse cannot be demonstrated. Irreversible high-impact actions can be made human-approval-only.

Validity Interval Δ

A contract is bound to an observation window. If it was generated at t_0, then:

$$t - t_0 > \Delta_{\max} \Rightarrow \mathcal{C}(a) \text{ expires}$$

Expiry triggers re-observation and reauthorization rather than execution against potentially stale topology or workload state.

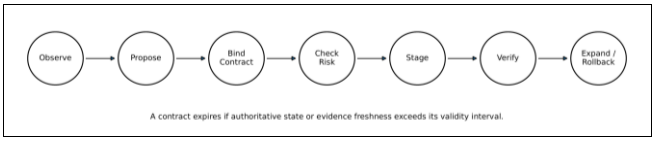

Fig 2: Contract lifecycle. Authorization is repeated when authoritative state changes or the contract expires

2.3 Risk-Bounded Authorization and Staged Execution

2.3.1 Risk Score

RACER separates hard constraints from soft risk. Hard constraints include policy denials, scope violations, failed preconditions, and unavailable mandatory rollback state. A proposal that survives hard checks receives a risk score:

$$\rho_t = w_u U_t + w_c C_t + w_r(1 - R_t) + w_o(1 - O_t) + w_e(1 - \epsilon_t)$$

With $\sum_i w_i = 1$. U_t denotes epistemic uncertainty, C_t criticality, R_t reversibility, O_t observability of consequences, and ϵ_t evidence sufficiency.

Equation (14) is a deliberately interpretable baseline. It should not be treated as a universal scoring law. A production implementation should calibrate risk from incident outcomes and monitor calibration drift. Where sufficient exchangeable calibration data exist, conformal risk control provides one possible family of techniques for bounding selected risks statistically [18]; RACER itself does not assume conformal validity in the simulation reported here.

2.3.2 Authorization Rule

A basic authorization rule is:

$$G(a) = \begin{cases} \text{deny}, & \text{hard policy or contract violation}, \\ \text{escalate}, & \rho_t > \tau_H \text{ or } \epsilon_t < \epsilon_{\min}, \\ \text{canary}, & \tau_L \le \rho_t \le \tau_H, \\ \text{execute}, & \rho_t < \tau_L. \end{cases}$$

The thresholds are operating-policy parameters. A high-risk proposal is not automatically rejected because some incidents require consequential repairs; instead it is transferred to a human authority with the evidence and contract attached.

2.3.3 Risk-Budgeted Canary Scope

A fixed 5% or 10% canary treats all actions as equally consequential. RACER instead derives an upper-bound candidate scope from Equation (10):

$$q_t = \min\left(q_{\max}, \max\left(q_{\min}, \frac{B_t}{\hat{p}_u(a)C(a) + \varepsilon}\right)\right)$$

Operational constraints can further reduce q_t. A destructive or poorly observable action can therefore receive a smaller initial exposure than a reversible restart even when the planner reports similar confidence.

2.3.4 Independent Verification and Expansion

After a canary stage, the verifier consumes fresh telemetry from sources defined in V. If $V = 0$, expansion stops and rollback is attempted. If $V = 1$, the action may advance to the next stage after refreshing state and reassessing residual risk. The sequence is:

$$q_1 \to q_2 \to \cdots \to 1$$

With a contract check between stages. Full rollout is therefore not a single authorization event but a series of bounded transitions.

2.3.5 Reference Algorithm

1. Normalize the planner output into a typed remediation action.
2. Resolve exact target resources and refresh authoritative state.
3. Bind $P, I, S, B, E, V, R, \Delta$ to form $\mathcal{C}(a)$.
4. Reject the proposal if parsing, identity, scope, or hard policy checks fail.
5. Evaluate freshness and all preconditions; expire the contract on mismatch.
6. Verify protected invariants and required rollback state.
7. Compute evidence sufficiency and the soft risk score ρ_t.
8. Deny, escalate, canary, or execute using Equation (15).
9. For canary mode, compute a bounded scope using Equation (16).
10. Execute only through the typed bounded executor.
11. Collect fresh telemetry and evaluate the pre-bound verification predicate V.
12. On verification failure, stop expansion and invoke R when valid.
13. Validate the restored state after rollback; escalate if restoration itself fails.
14. On verification success, refresh state, recompute risk, and authorize the next stage.
15. Persist the proposal, evidence references, contract, decision, action, observations, and final outcome in an audit trace.

2.4 Conditional Assurance Properties

The purpose of this section is to state limited properties that follow from the architecture under explicit assumptions. RACER does not claim that the planner is correct or that arbitrary cloud systems can be made safe by a scalar risk score.

2.4.1 Policy Confinement

Proposition 1: Assume (A1) every state-changing action originating from the agent workflow passes through the RACER executor; (A2) the policy evaluator and executor cannot be bypassed by that workflow; (A3) the executor accepts only typed actions; and (A4) the declared policy is sound with respect to the prohibited action set. Then an action rejected by the policy evaluator cannot be executed through the RACER-controlled channel.

Proof sketch. Complete mediation means each candidate action a requires $Policy(a, x_t) = Allow$ before the executor invokes the transition function. For any a^* with $Policy(a^*, x_t) = Deny$, the transition function is unreachable through that channel. The proposition fails when an alternative privileged path exists or when the policy omits a prohibited behavior.

2.4.2 Declared Blast-Radius Bound

Proposition 2: Assume (B1) the concrete target set is enumerated or deterministically selected from S; (B2) the executor enforces the maximum target count or fraction; and (B3) no hidden executor expands the action. If the contract specifies stage exposure $q \le q_{\max}$, then the number of directly actuated resources in that stage does not exceed the corresponding declared scope bound.

This is an actuation-scope guarantee, not a guarantee that indirect dependency effects remain inside the same fraction. Cascading effects must be addressed by consequence modeling and verification.

2.4.3 Bounded Violation Duration under Successful Rollback

Proposition 3: Assume (C1) a violation of V becomes observable and is detected within Δ_V; (C2) rollback is feasible; (C3) rollback completes within Δ_R; and (C4) rollback restores the relevant protected pre-action state. Then the interval between observable contract violation and restoration is bounded by:

$$T_{\text{violation}} \le \Delta_V + \Delta_R$$

The proposition does not apply to irreversible actions, silent failures outside the verifier's observability, corrupted restoration state, or rollbacks that introduce a distinct failure. These exceptions are why reversibility and observability are explicit risk factors rather than implicit assumptions.

2.5 Reproducible Synthetic Decision Study

2.5.1 Purpose and Scope

The study answers RQ4 at the level of the authorization mechanism. It asks whether the proposed combination of gating, staged execution, verification, and escalation behaves as intended under controlled uncertainty. It does *not* estimate a production incident rate and does not provide evidence that RACER will achieve the same percentages on Kubernetes or any specific service.

We use 20 independent random seeds with 50,000 incident/remediation pairs per seed, for a total of 1,000,000 simulated incidents. The complete script and per-seed outputs are included in the submission supplement.

2.5.2 Synthetic Incident Model

For each incident, evidence quality is:

$$e \sim Beta(3,2)$$

Epistemic uncertainty is correlated inversely with evidence but includes noise:

$$u = clip(1 - e + \mathcal{N}(0,0.12), 0,1)$$

Criticality, reversibility, and observability are:

$$c \sim Beta(2,3),$$
$$r \sim Beta(3,2),$$
$$o \sim Beta(3,2).$$

The probability that the proposed repair is technically correct is:

$$p_{correct} = \sigma(-0.8 + 3.0e + 0.8o - 1.8u - 0.3c)$$

Conditional on an incorrect proposal, the probability of material harm is:

$$p_{harm|wrong} = clip(0.20 + 0.65c + 0.20(1 - r), 0,1)$$

These coefficients are scenario assumptions chosen to create heterogeneous incidents; they are not learned from a cloud-incident dataset.

Some harmful proposals are recognizable as hard policy violations with probability:

$$p_{policy} = clip(0.08 + 0.35c + 0.15(1 - r), 0,0.70)$$

Human review succeeds with synthetic probability:

$$p_{human} = clip(0.90 + 0.05e - 0.05c, 0.70,0.98)$$

For RACER, the simulated soft risk score is:

$$\rho = 0.28u + 0.30c + 0.22(1 - r) + 0.20(1 - o)$$

A proposal is high-risk if $\rho > 0.58$ or $e < 0.35$, medium-risk if $0.34 \leq \rho \leq 0.58$ after the evidence check, and otherwise low-risk.

For a harmful action, the probability that the canary detects the failure is:

$$p_{canary} = clip(0.45 + 0.35o + 0.15e, 0,0.98)$$

And the independent post-action verifier has:

$$p_{verify} = clip(0.35 + 0.45o + 0.10e, 0,0.95)$$

When rollback is invoked, synthetic rollback success is:

$$p_{rollback} = clip(0.45 + 0.50r, 0,0.98)$$

2.5.3 Compared Policies

Ungated executes every proposal globally. **Policy-only** diverts hard policy violations to human review but globally executes all remaining proposals. **Canary-all** applies a fixed canary to every proposal and contains detected harmful actions. **RACER** combines hard policy checks, high-risk/evidence escalation, canarying for medium risk, direct execution for low risk, and independent verification for automated harmful actions.

2.5.4 Metrics

Automation rate is the fraction of incidents handled without human escalation. **Unsafe global actuation** is the fraction of all incidents in which a materially harmful proposal reaches global execution without being contained by the modeled safety mechanisms. **Recovery success** counts technically correct automated actions, successful human resolutions for escalated cases, and successful rollbacks of contained harmful actions. **Rollback rate** is the fraction of incidents that invoke rollback. The script also emits a normalized synthetic resolution-cost quantity, but we do not use it to make claims about measured wall-clock MTTR because no real infrastructure is involved.

For each metric we report the mean across 20 seeds and a 95% confidence interval computed as $1.96s/\sqrt{20}$ over the per-seed estimates.

2.5.5 AI-Assisted Research Tooling Disclosure

OpenAI ChatGPT (GPT-5.6 Sol) was used to assist in writing and executing the Python implementation of this synthetic Monte Carlo experiment and in checking arithmetic summaries. The probability model, thresholds, metrics, and interpretation are stated explicitly above; the submitted script reproduces the reported tables from fixed seeds. The authors remain responsible for inspecting the code, rerunning the artifact, and verifying every result before submission. This disclosure is included because the AI tool was used in the research workflow rather than solely for prose editing.

3. Results and Discussion

3.1 Primary Results

Table 2 reports the primary simulation results. The confidence intervals are sampling variability under the synthetic generator, not uncertainty about production behavior.

Table 2: Synthetic decision-study results over 1,000,000 incidents (20 seeds 50,000). Values are mean percentages 95% CI across seeds

Policy	Auto.	Escal.	Unsafe	Recovery	Rollback
Ungated	100.00	0.00	20.14±0.11	63.04±0.11	0.00
Policy-only	93.89±0.06	6.11±0.06	14.03±0.06	68.54±0.07	0.00
Canary-all	100.00	0.00	5.48±0.05	73.93±0.07	14.65±0.10
RACER	**80.57±0.09**	**19.43±0.09**	**1.33±0.02**	**82.21±0.05**	**8.58±0.05**

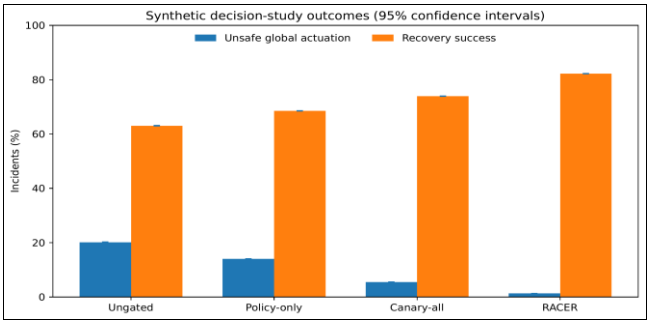

Fig 3: Unsafe global actuation and recovery success under the synthetic model. Error bars are 95% confidence intervals across 20 seeds

3.1.1 RQ4: Effect of Execution Controls

Ungated execution produces unsafe global actuation in 20.14% of simulated incidents. Policy-only gating reduces this to 14.03%, showing that categorical policy checks catch only a subset of the modeled harmful actions. Fixed canarying produces a larger reduction to 5.48% because harmful proposals can be observed before global exposure. RACER's combination of selective escalation and independent verification reduces the modeled rate further to 1.33%.

Relative percentages should be interpreted only inside the synthetic model. For example, RACER's unsafe-actuation value is approximately 93.4% lower than the ungated value and 75.7% lower than canary-all, but those relative reductions are consequences of the specified distributions and detection functions. They are not a claim of a 93.4% production safety improvement.

RACER automates 80.57% of incidents and escalates 19.43%. The lower automation rate is intentional. The architecture optimizes useful autonomy subject to risk constraints rather than treating 100% autonomy as the objective. Under the model, RACER also reaches the highest recovery success (82.21%), because high-risk cases can be successfully resolved by the modeled human path and detected harmful actions can recover through rollback.

3.2 Ablation Analysis

To identify which controls drive the modeled safety effect, we rerun the study with three components removed. The ablation uses a disjoint deterministic random-number stream and the same number of seeds/incidents.

Table 3: Ablation results. Unsafe actuation and recovery are mean percentages 95% CI

Variant	Automation	Unsafe	Recovery
Full RACER	80.57±0.09	**1.28±0.02**	**82.22±0.06**
No canary execution	80.57±0.09	3.18±0.04	80.82±0.05
No post-action verification	80.57±0.09	4.11±0.04	80.04±0.06
No risk-based escalation	93.97±0.04	2.81±0.03	76.83±0.07

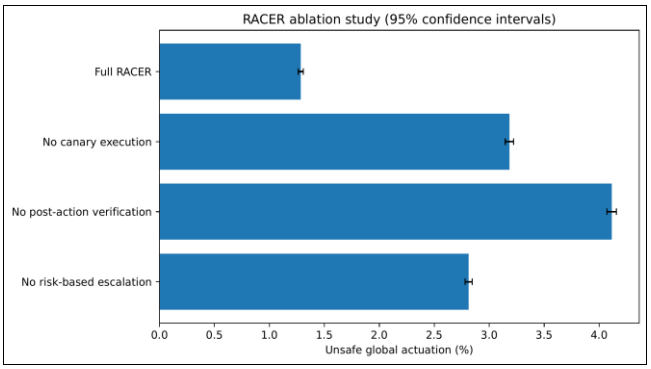

Fig 4: Unsafe global actuation in the RACER ablation study. All three controls contribute under the synthetic model

Removing canary execution more than doubles unsafe actuation, indicating that a pre-execution score cannot compensate for an incorrect action whose harm is discoverable only after actuation. Removing post-action verification has a larger effect because harmful direct actions and harmful actions that evade canary detection lose the second containment opportunity. Removing risk-based escalation raises automation to 93.97% but lowers recovery to 76.83% and increases unsafe actuation. The ablation supports the mechanism-level hypothesis that safe autonomy is a composition of controls rather than a single confidence threshold.

The small difference between the full-RACER unsafe percentage in Table 2 (1.33%) and the ablation baseline (1.28%) is expected because the ablation deliberately uses a separate random stream. Both experiments use the same generator and report their own per-seed confidence intervals.

3.3 Deployment Architecture and Evaluation Protocol

3.3.1 Kubernetes Mapping

A concrete RACER implementation can be realized as an execution proxy between an AIOps agent and the Kubernetes API. The planner receives read-only telemetry and produces a typed proposal. The proxy resolves immutable resource UIDs, obtains current objects directly from the API server, and evaluates contracts. Service-account credentials exposed to the planner should not themselves permit unrestricted mutation; write authority resides in the proxy and is constrained by action type and scope.

Contract clauses can map to Kubernetes mechanisms. Preconditions can check resourceVersion, generation, rollout revision, or current replica state. Protected invariants can enforce PodDisruptionBudget-like availability bounds or organization-specific policy. Canary scope can be implemented by label-selected subsets, traffic splitting, or progressive-delivery controllers. Rollback state can reference the previous Deployment revision or a captured configuration object. Verification can consume SLOs, health probes, traces, and dependency metrics from an independent telemetry path.

3.3.2 Deployment-Grade Research Questions

A real-system evaluation should test at least the following:

RQ-D1: Does RACER reduce unsafe or excessive state-changing actions relative to ungated agents, hard policy gating, and fixed canaries?

RQ-D2: What MTTR and SLO-impact cost is introduced by contract checks, staged execution, and escalation?

RQ-D3: Does the benefit persist across planners with different diagnostic accuracy and calibration?
RQ-D4: Which contract clauses most frequently block or contain failures in realistic fault classes?

3.3.3 Recommended Environments and Faults

AIOpsLab is a natural end-to-end environment because it can deploy microservices, generate workload, inject faults, expose telemetry, and interface with agents [2]. RCAEval provides a broad catalog of microservice failures and baselines [4], while OpenRCA supplies a demanding root-cause benchmark that can be used to vary proposal quality [3]. A Kubernetes campaign should include multiple applications, such as Online Boutique plus at least two independent microservice systems, and faults spanning resource exhaustion, network delay/loss, dependency failures, configuration defects, and application-level anomalies.

3.3.4 Baselines and Models

The same incident set should be replayed against: (1) recommendation-only; (2) ungated autonomous execution; (3) RBAC/policy-only gating; (4) fixed canary execution; (5) human approval for every action; (6) RACER without evidence gating; (7) RACER without independent verification; and (8) full RACER. At least one strong proprietary model, one open-weight model, and a weaker baseline should be used as planners. A deterministic runbook baseline is valuable where a known remediation exists. RACER should not be tuned separately for each model without reporting that tuning.

3.3.5 Primary Metrics

A deployment-grade evaluation should report root-cause accuracy separately from execution outcomes. Execution metrics include unsafe-action rate, contract-rejection rate, blast-radius violations, rollback attempts and success, false recovery, human interventions, MTTR, SLO degradation area, customer-impact duration, and control-plane overhead. Cost should include model/token use where relevant.
We define false recovery rate as:

$$FRR = \frac{\#\{\text{actions declared recovered while impairment persists}\}}{\#\{\text{actions declared recovered}\}}$$

This metric is important because a self-healing system that incorrectly declares success can suppress further investigation and prolong hidden degradation.

3.3.6 Experimental Discipline

Model versions, prompts, tool schemas, contract rules, and thresholds should be frozen before the final campaign. Runs should use repeated seeds or repeated workload/fault realizations, all failures and rollbacks should remain in the dataset, and confidence intervals should be reported. Action traces should be immutable. The replication package should include fault definitions, deployment manifests, analysis scripts, and exact model/API identifiers where licensing permits.

3.4 Trustworthiness and Socio-Dependability

RACER is designed for trustworthy autonomy rather than autonomy maximization. This has three socio-technical implications relevant to operational environments.
First, *authority is explicit*. A human operator can see which actions are categorically denied, which require escalation, and which may proceed within a bounded scope. This makes organizational risk policy part of the control loop rather than an informal expectation placed on a language model.
Second, *evidence and accountability travel with the action*. The audit trace records the evidence references, contract, authorization decision, concrete API call, verification result, and any rollback. Such provenance supports incident review and policy improvement, and it makes it possible to distinguish planner error from verifier, policy, or executor failure.
Third, *human escalation is a designed boundary*. Highly consequential or weakly evidenced actions should not be forced into autonomous execution merely to increase an automation metric. The appropriate target is calibrated delegation: machine autonomy where the consequence can be bounded and observed, and human authority where it cannot.

3.5 Security Considerations

RACER does not eliminate adversarial risk. An attacker who can falsify telemetry can attempt to satisfy evidence and verification predicates. Evidence sources should therefore carry provenance, and critical predicates should use authoritative or independently protected telemetry where possible. Similarly, a compromised verifier can falsely approve expansion; separating planner and verifier reduces correlated error but does not help if both depend on the same compromised source.
Rollback is also security-sensitive. A malicious or simply destructive action may eliminate the state required for restoration. For high-impact operations, restoration prerequisites should be checked before actuation, and rollback credentials should be isolated from the planner. Finally, the audit trail itself is security evidence and should not be modifiable by the same identity that performs production mutations.

3.6 Threats to Validity and Limitations

3.6.1 Construct Validity

The synthetic "unsafe global actuation" metric collapses many real consequences into a binary harmful/not-harmful outcome. Real systems require severity-weighted measures and dependency-aware blast radius. Recovery success similarly abstracts the quality and duration of a recovery.

3.6.2 Internal Validity

The simulation is internally reproducible but depends directly on hand-specified probability functions. Thresholds and coefficients were selected to exercise all policy paths, not fitted to historical incident data. The results therefore demonstrate comparative behavior under those assumptions, not causal effectiveness in production.

3.6.3 External Validity

No Kubernetes cluster, production service, or AIOpsLab environment is used in the present evaluation. The reported percentages must not be generalized to a real cloud. Latency, correlated failures, hidden dependencies, operator behavior, and rollback semantics can materially change outcomes. The deployment protocol in the previous section is necessary before making empirical claims about operational effectiveness.

3.6.4 Assurance Limitations

The formal propositions are conditional. Complete mediation is a strong architectural assumption, policy soundness cannot be guaranteed by RACER, indirect cascading effects can exceed the directly actuated scope, and rollback may be impossible. The framework should

therefore be understood as a means of making assumptions explicit and enforceable where possible, not as a universal safety proof.

3.6.5 Model and Data Drift

A calibrated risk model can drift as applications, workloads, models, and incident populations change. Production deployments need calibration monitoring and conservative fallback behavior. An uncalibrated planner confidence score should not be treated as $\hat{p}_u$ without evidence that the mapping is meaningful.

3.7 Reproducibility and Artifact Availability

The submission package contains *racer_simulation.py*, the four per-seed/summary CSV files, and the figure-generation script used for the synthetic study. Running the simulation with Python 3, NumPy, and pandas reproduces the tables from deterministic seeds. The artifact intentionally contains no proprietary operational data.

For the current submission, the replication package supports only the synthetic study. A future deployment-grade artifact should additionally contain Kubernetes manifests, fault-injection definitions, the RACER execution proxy, policy bundles, verifier configurations, model/tool schemas, and a script that replays an identical incident set across baselines.

4. Conclusion

RACER addresses a specific boundary in autonomous operations: the transition from an uncertain AI recommendation to a production-changing action. The framework converts a proposed repair into a Remediation Assurance Contract whose preconditions, invariants, scope, risk budget, evidence, verification, rollback, and validity are machine-checkable. A risk-bounded policy decides whether the proposal is denied, escalated, canaried, or executed, while a planner-independent verifier controls expansion and rollback.

The conditional properties clarify what an execution boundary can guarantee without claiming correctness of the underlying model. The reproducible simulation provides mechanism-level evidence that the combination of policy checks, selective escalation, staged execution, and independent verification behaves more conservatively than the compared baselines under the stated assumptions. It does not establish real-cloud effectiveness. The decisive next step is therefore an implementation and fault-injection campaign in AIOpsLab/Kubernetes that measures safety, recovery, SLO impact, false recovery, human intervention, and overhead across multiple planners.

The broader claim is intentionally modest but consequential: an AI agent should not receive state-changing authority merely because it can formulate a plausible remediation. Trustworthy autonomy requires a separate, inspectable execution boundary that can say *no*, limit scope, demand evidence, verify consequences, and reverse when the evidence changes.

5. Recommendations

The next empirical stage should implement RACER as a model-independent execution proxy in a fault-injected Kubernetes environment and replay the same incident set across ungated execution, policy-only gating, fixed-canary execution, human approval, component ablations, and full RACER. The evaluation should report unsafe-action rate, false recovery, rollback success, MTTR, SLO degradation, human intervention, and control-plane overhead across multiple planners. Thresholds and contract rules should be frozen before the final campaign, all failed and rolled-back runs should remain in the dataset, and the complete replication package should be released where licensing and operational-security constraints permit.

6. References

1. Shetty M, Chen Y, Somashekar G, Ma M, Simmhan Y, Zhang X, *et al*. Building AI agents for autonomous clouds: Challenges and design principles. In: Proceedings of the 2024 ACM symposium on cloud computing, 2024, 99-110. Doi: 10.1145/3698038.3698525
2. 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. In: Proceedings of the eighth conference on machine learning and systems [Internet], 2025. Available from: https://openreview.net/forum?id=3EXBLwGxtq
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? In: The thirteenth international conference on learning representations [Internet], 2025. Available from: https://openreview.net/forum?id=M4qNIzQYpd
4. Pham L, Zhang H, Ha H, Salim F, Zhang X. RCAEval: A benchmark for root cause analysis of microservice systems with telemetry data. In: Companion proceedings of the ACM on web conference 2025, 2025, 777-780. Doi: 10.1145/3701716.3715290
5. Liu X, Yu H, Zhang H, Xu Y, Lei X, Lai H, *et al*. AgentBench: Evaluating LLMs as agents. In: International conference on learning representations [Internet], 2024. Available from: https://openreview.net/forum?id=zAdUB0aCTQ
6. Ruan Y, Dong H, Wang A, Pitis S, Zhou Y, Ba J, *et al*. Identifying the risks of LM agents with an LM-emulated sandbox. In: International conference on learning representations [Internet], 2024. Available from: https://openreview.net/forum?id=GEcwtMk1uA
7. Kephart JO, Chess DM. The vision of autonomic computing. Computer. 2003; 36(1):41-50. Doi: 10.1109/MC.2003.1160055
8. Randell B. System structure for software fault tolerance. In: Proceedings of the international conference on reliable software, 1975, 437-449. Doi: 10.1145/390016.808467
9. Seto D, Krogh BH, Sha L, Chutinan A. The simplex architecture for safe online control system upgrades. In: Proceedings of the american control conference, 1998, 3504-3508. Doi: 10.1109/ACC.1998.703255
10. Alshiekh M, Bloem R, Ehlers R, Könighofer B, Niekum S, Topcu U. Safe reinforcement learning via shielding. In: Proceedings of the AAAI conference on artificial intelligence, 2018. Doi: 10.1609/aaai.v32i1.11797
11. Ames AD, Xu X, Grizzle JW, Tabuada P. Control barrier function based quadratic programs for safety critical systems. IEEE Transactions on Automatic Control. 2017; 62(8):3861-3876. Doi: 10.1109/TAC.2016.2638961
12. Garcia L, Samin H, Bencomo N. Decision making for self-adaptation based on partially observable satisfaction of non-functional requirements. ACM

Transactions on Autonomous and Adaptive Systems. 2024; 19(2):1-44. Doi: 10.1145/3643889
13. Casimiro M, Soares D, Garlan D, Rodrigues L, Romano P. Self-adapting machine learning-based systems via a probabilistic model checking framework. ACM Transactions on Autonomous and Adaptive Systems. 2024; 19(3):1-30. Doi: 10.1145/3648682
14. Ponduru PS, Nandanavanam PPV, Ponduru SKK. SAFE-HealCloud: Safety-aware, agentic self-healing for cloud infrastructure. International Journal of Scientific Research in Computer Science, Engineering and Information Technology. 2026; 12(4):163-188. Doi: 10.32628/CSEIT26124220
15. Ponduru PS, Nandanavanam PPV, Ponduru SKK. OpenReliOps: A safety-constrained architecture for autonomous reliability in open-weight model services. International Journal of Advanced Multidisciplinary Research and Studies. 2026; 6(4):1114-1121. Doi: 10.62225/2583049X.2026.6.4.6707
16. Ponduru PS. A unified framework for self-evolving and self-healing artificial intelligence agents. Advanced International Journal of Multidisciplinary Research. 2026; 4(4). Doi: 10.62127/aijmr.2026.v04i04.1435
17. Fang X, Getir Yaman S, Calinescu R, Wilson J, Paterson C. Predicting nonfunctional requirement violations in autonomous systems. ACM Transactions on Autonomous and Adaptive Systems. 2024; 19(1). Doi: 10.1145/3632405
18. Angelopoulos AN, Bates S, Fisch A, Lei L, Schuster T. Conformal risk control. In: International conference on learning representations [Internet], 2024. Available from: https://openreview.net/forum?id=33XGfHLtZg