Journal of Computer Science and Technology Studies
ISSN: 2709-104X
DOI: 10.32996/jcsts
Journal Homepage: www.al-kindipublisher.com/index.php/jcsts

JCSTS
AL-KINDI CENTER FOR RESEARC
AND DEVELOPMENT

| RESEARCH ARTICLE

Evidence-Gated Search: Controlling Operational Search Explosion in LLM-Driven Incident Response

Sudhakavya Bodapati Venkata
DevOps Engineer, Starkey Laboratories Inc.
Corresponding Author: Sudhakavya Bodapati Venkata, **E-mail**: bvsudhakavya@gmail.com

| ABSTRACT

Large language model (LLM) assistants are increasingly used during outages and failed deployments, yet their remediation behavior can degrade into long lists of loosely justified changes such as restarts, redeploys, rollbacks, scaling changes, and configuration edits. In practice, this ap- pears as operational search explosion (OSE), where the branching factor of candidate fixes grows faster than teams can validate them, increasing change risk, time-to-recovery, and operator distrust. This paper formulates incident response as a bounded search problem over a constrained library of remediation primitives, where each primitive carries explicit preconditions, risk and cost meta- data, and evidence requirements that must be satisfied before execution. The proposed mechanism, Evidence-Gated Search (EGS), blocks any state-changing action unless the required evidence is present in a normalized incident state. Missing evidence forces bounded read-only evidence gathering, such as logs, metrics, traces, deployment diffs, Terraform plan outputs, DNS checks, and access-control verification, before another action is considered. Across 175 incident episodes, EGS reduces executed remediation actions by 13.47% and lowers the action explosion rate (AER) by 16.65%, while maintaining a recovery success rate of 98.86%. The results show that requiring evidence before irreversible operational steps can significantly reduce operational search complexity without materially degrading recovery outcomes.

| KEYWORDS
Incident response automation, large language models, operational search explosion, evidence- gated search, safe remediation, budgeted search control, reliability engineering, DevOps

| ARTICLE INFORMATION

ACCEPTED: 20 February 2026 **PUBLISHED:** 19 March 2026 **DOI:** 10.32996/jcsts.2026.8.5.8

1. Introduction

Production incidents are messy in a way that is hard to capture in clean decision trees. A deployment fails, the service goes partially dark, and the telemetry arrives out of order: one dashboard shows a spike in 5xx, another shows stable CPU, the pipeline reports a permissions error, and the application logs complain about a database connection that looks healthy from the outside. In that gap, teams increasingly lean on large language models (LLMs) to summarize signals and suggest next steps. The benefit is real, but a serious failure mode appears when the model is allowed to “just keep trying” fixes.

The failure mode is not a bad suggestion. The problem is the concentration of numerous plausible suggestions which are weakly based. Restart service, redeploy previous build, scale out, rotate credentials, roll back or rerun the pipeline. Every action is justifiable in its own right, but every action also alters the system and generates new noise. The loop begins to resemble exploration instead of diagnosis. Engineers are aware of the pattern since it has a well-known smell: a restart loop when health checks flap, a redeploy loop when the pipeline passes but the runtime still fails, or a rollback attempt that hits an irreversible migration. When an automated assistant accelerates that pattern, the operational cost spikes. “Fast” becomes “fast at making changes.”

We refer to this pattern as *operational search explosion* (OSE). OSE is the unmanaged expansion of candidate remediation measures that are taken into account and implemented in response to an incident, which is often accompanied by repetitive cycles and increasing risk. The price of a misstep is seldom local in microservice and cloud environments. Restart may increase load on a failing dependency. A broken configuration can be rolled forward again by a redeploy. Assumptions may be silently corrupted by a rollback when schema changes have already been propagated. Even read-only operations are not free when sprayed over the entire hypothesis space, as they increase the size of the hypothesis space and slow down validation. Current incident tooling is more concerned with detection and diagnosis, and the literature on AIOps has good outcomes on anomaly localization and root cause analysis [8,7,9,1,11,14,10,5,12]. The gap is what happens *after* a hypothesis is formed: the decision to execute state-changing remediation, under pressure, with incomplete evidence.

The core stance here is simple and intentionally strict: state-changing actions should not run unless there is explicit evidence that makes the action reasonable *in the current incident state*. This is the way on-call engineers act in reality. They would like to observe a narrow time correlation with the previous release and a strategy on what to do with data before a rollback. They desire tangible indications that authentication is not working and that the blast radius is known before rotating secrets. They desire a broken DNS resolution or private endpoint probe, rather than an ambiguous ”connectivity problem”, before they touch networking. The architecture presented in this paper imposes the same discipline on the loop.

Evidence-Gated Search (EGS) is an implementation of this discipline, which views incident response as a finite-library finite-search problem over a finite library of remediation primitives. Every primitive is characterized by (i) preconditions, which should be true, (ii) evidence requirements, which should be met, and (iii) risk and cost metadata. An LLM is a proposer which ranks a small set of candidate primitives and provides an evidence claim on each. The claim is then checked against a normalized incident state by an evidence gate. In the absence of evidence, the loop will be driven to read-only evidence collection, including targeted log windows, metric queries, deployment diffs, Terraform plan outputs, DNS and endpoint probes, and access-control checks [2,3,13]. A budgeted search controller prunes the action space using risk-aware ordering, top-k selection, cycle prevention, and no-repeat rules, which reduces the chance of “restart until it works” behavior.

There are three operationally important outcomes, which influence the assessment in this paper. To start with, the action stream must be reduced, the number of actions performed must be reduced, the number of branches explored must be reduced, the number of repeated loops must be reduced. Second, the risk profile is expected to become better, with fewer high-impact actions being undertaken unnecessarily, and fewer policy breaches in limited settings. Third, recovery must not become worse, as a safe assistant that is slow is not of use during an outage. The rest of the paper describes the EGS architecture, the evidence rules and controller, and an analysis that reports recovery results and search-complexity metrics, such as the number of actions, effective branching factor, rate of action explosion, and a budget-weighted cost of operation search.

The main contributions of this paper are as follows:

– We formalize *operational search explosion* (OSE) as the uncontrolled growth of remediation actions considered and executed during LLM-assisted incident response.
– We propose *Evidence-Gated Search* (EGS), a control-loop architecture that blocks state-changing

actions unless explicit evidence requirements are satisfied.

- We design a budget-aware controller that combines top-k pruning, no-repeat constraints, and cycle prevention to suppress unsafe restart and redeploy loops.
- We evaluate the approach on incident episodes derived from microservice traces and report both recovery outcomes and operational search metrics, including executed actions, action explosion rate, and budget-weighted search cost.

2. Related Work

Incident response with the help of LLM is based on the automated failure diagnosis and operational telemetry knowledge. Cloud and microservice systems RCA approaches model dependency structure and propagation of anomalies to localize faults in the face of partial observability [8,9,12]. AIOps surveys highlight that diagnosis is not enough, remediation should be combined with effective operational controls to prevent unsafe automation and operator overload [4,6]. Log-based anomaly techniques and practical log-parsing pipelines assist in extracting evidence out of logs and telemetry data [3,2,13]. EGS focuses on the control gap once the diagnosis is made, it limits the action space of remediation, it demands explicit evidence prior to state changes, and it implements budgets and cycle prevention to curb operational search explosion.

3. Problem Formulation and Definitions

Incident response can be modeled as a sequential decision process over a live production system. The system evolves due to external workload, background automation, and the response actions taken during the incident. The goal is not to “solve” the system in an offline sense, it is to restore acceptable service behavior under constraints that are non-negotiable in real operations (policy, safety, and change budget).

3.1 Incident state

Let S_t denote the incident state at time t. S_t is a structured representation assembled from heterogeneous signals: logs, metrics, traces, deployment and pipeline data, configuration snapshots, and infrastructure drift indicators. Besides symptoms (e.g., high error rate, failed health checks), S_t also has operational context: environment (dev, staging, production), recent change set (last deploy, config diffs), and explicit constraints (for example, “no public network exposure,” “no secret printing,” “rollback allowed only if migration is reversible”). The representation is deliberately pragmatic, it is a normalized view that is used to check evidence and to measure search behavior, not an ideal model of the underlying system.

3.2 Action primitives and execution

Let A be a finite library of remediation primitives. Each primitive $a \in A$ is an operationally meaningful action with bounded scope, such as restarting a service, re-running a deployment stage, rolling back to a prior release, reverting a configuration key, or scaling a component. The library is finite by design; it serves two purposes. First, it does not allow the assistant to come up with new actions that would not be reviewed by safety. Second, it renders evaluation repeatable through action space repair.

The metadata of each action a is defined:

– **Preconditions** $\mathrm{Pre}(a)$: predicates over S_t that should be true before execution (e.g., there is a previous version, the target is reachable, a feature flag is on).
– **Evidence requirements** $\mathrm{Ev}(a)$: required evidence items that must be present in S_t to justify the action (e.g., regression correlated with last deploy, DNS resolution fails to a private endpoint, access denied errors to a managed identity).
– **Risk level** $r(a)$ and **cost** $c(a)$: operationally defined attributes that encode the blast radius, reversibility, and anticipated disruption.

Execution produces an observation and a successor state. Let $\mathrm{Exec}(a, S_t)$ denote an execution attempt that returns an outcome tuple (o_t, Δ_t), where o_t captures observable results (pipeline status, error rate shift, new log signatures) and Δ_t captures state updates that flow into S_{t+1}.

3.3 Evidence, budgets, and OSE metrics

Evidence is a structured fact derived from telemetry or configuration and anchored to time and scope. Let E_t denote the evidence items present in S_t. Each action $a \in A$ declares evidence requirements $\mathrm{Ev}(a)$ that must hold before execution. Budgets are explicit to prevent unbounded exploration:

$$B = (B_{\mathrm{act}}, B_{\mathrm{risk}}, B_{\mathrm{time}}, B_{\mathrm{read}}),$$

where B_{act} bounds state-changing actions and B_{read} bounds read-only evidence gathering. Policy constraints Π encode non-negotiable rules (e.g., disallow network exposure changes in production).

Operational Search Explosion (OSE) is quantified by executed action volume and candidate fan-out. Let b_t be the number of candidate actions considered at step t and T the number of steps. We report action count N_{exec} and action explosion rate:

$$\mathrm{AER} = \frac{1}{T}\sum_{t=1}^{T} b_t .$$

To reflect operational severity, we also report a budget-weighted operational search cost:

$$\mathrm{OSC} = \sum_{a \in X} (\alpha c(a) + \beta r(a)) + \gamma N_{\mathrm{read}},$$

where X is the multiset of executed state-changing actions and N_{read} is the number of read-only queries.

3.4 Problem statement

Given S_0, A, budgets B, and hard policies Π, the objective is to reach $\mathrm{Healthy}(S_t)$ while minimizing OSE (lower N_{exec}, lower AER, lower OSC) subject to B and Π.

4. Evidence-Gated Search (EGS): Architecture and Control Loop

EGS is built for one job: keep incident response from turning into uncontrolled experimentation. The design assumes the assistant can be wrong, the telemetry can be incomplete, and production policies are strict. Instead of trusting a free-form agent loop, EGS forces two disciplines that experienced on-call engineers already use: (i) choose from a finite set of known operational moves, and (ii) demand evidence before executing a move that changes system state.

4.1 Components

Fig. 1 shows the control loop. The main blocks are:

State collector and normalizer. Raw signals are in the form of logs, metrics, traces, pipeline outputs, and infrastructure diffs. They are transformed into a structured incident state S_t containing evidence items E_t, recent-change context, environment metadata, and hard constraints (policies).

Action primitive library. The action space is a finite set A of remediation primitives. Each primitive carries preconditions $\mathrm{Pre}(a)$, evidence requirements $\mathrm{Ev}(a)$, and operational attributes like risk $r(a)$ and cost $c(a)$. Here domain practice is coded: rollback is not a similar type of move to a restart, and the checks that warrant each are different.

LLM proposer. The LLM does not execute actions. It ranks a small set of candidates from A and attaches an evidence claim for each candidate, typically by pointing to specific evidence fields in S_t and explaining the intended effect. In implementation, this can be top-k proposals with short rationales and explicit references to evidence keys.

Evidence gate. The gate determines the admissibility of a proposed action in the prevailing state. It verifies evidence requirements, checks preconditions, imposes policy constraints, and consults budgets. The gate does not guess in case the proposal fails because of the absence of evidence. It demands evidence collection and prevents state-altering execution.

Evidence gatherer (read-only). This module runs limited, focused read-only queries to augment S_t: log windows aligned to the failure time, metric slices, DNS and endpoint probes, access-control checks, deployment diffs, and Terraform plan outputs. It is purposefully tedious and deterministic since it is the foundation of reproducibility.

Search controller. This block deals with exploration within budgets. It picks the next admissible action to perform, imposes top-k pruning and risk-sensitive ordering, avoids cycles, and avoids repeating failed actions unless new evidence is received. This is the place where OSE is actively repressed.

Executor and evaluator. The executor executes the approved state-changing primitive and logs inputs, outputs, and side effects. The evaluator updates S_{t+1} and decides whether the recovery predicate $\mathrm{Healthy}(S_{t+1})$ holds, and it updates the search traces used to compute OSE metrics.

4.2 EGS control loop

The loop alternates between two modes: evidence acquisition and action execution. The switch is controlled by the gate. When evidence is incomplete, the loop spends budget on read-only queries. When evidence is sufficient, the loop spends budget on a state-changing action. This creates friction in the right place: evidence becomes a required currency for risky or disruptive operations.

4.3 Gate semantics

The evidence gate implements an admissibility predicate. For a candidate action a at time t, the action is admissible if:

$$\mathrm{Admit}(a, S_t) \equiv \mathrm{Pre}(a) \wedge \ \mathrm{Ev}(a) \subseteq E_t \ \wedge \neg \mathrm{Viol}(a, \Pi) \wedge \mathrm{Within}(a, B_t).$$

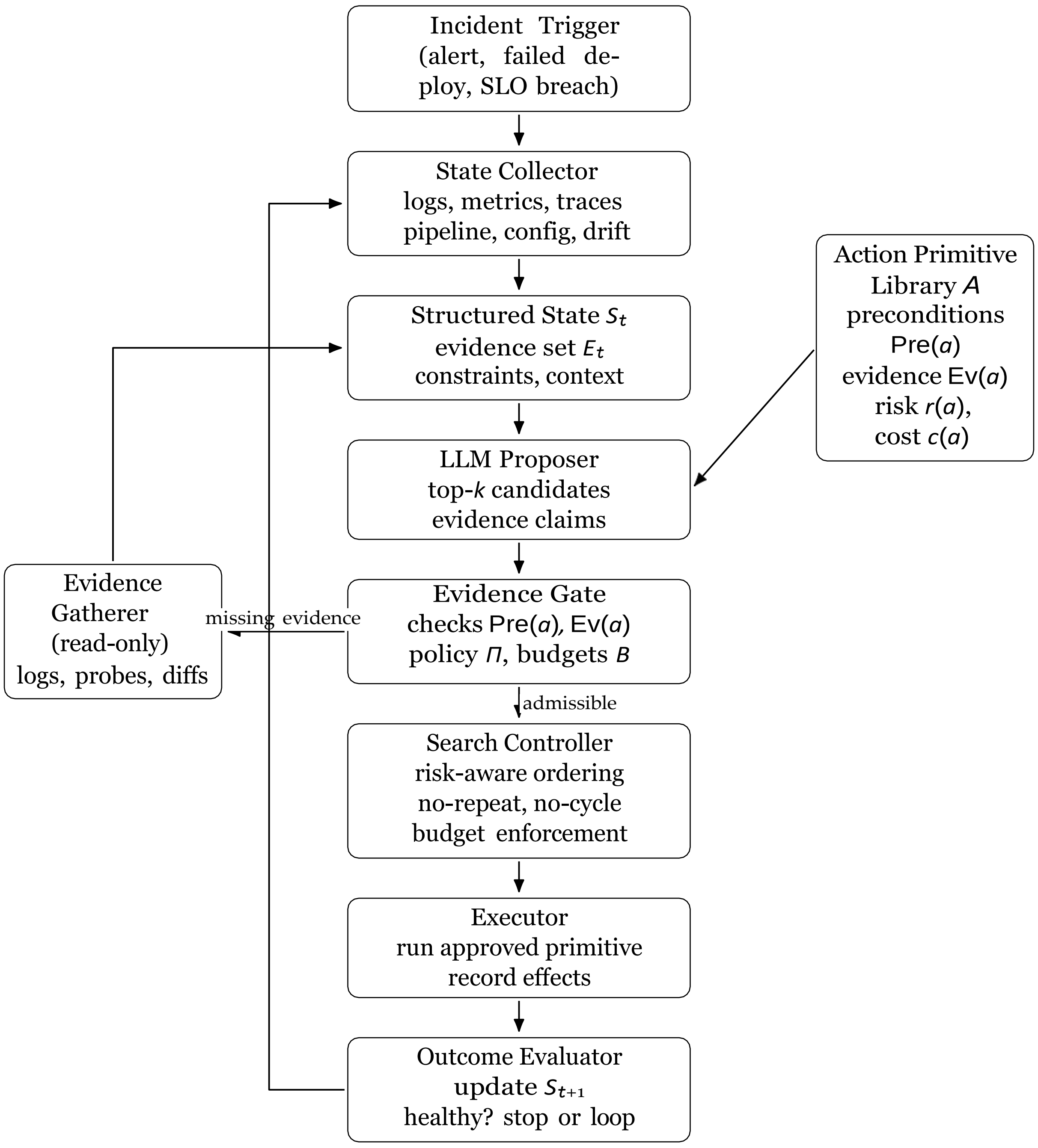

Fig. 1. Evidence-Gated Search (EGS) control loop. The LLM proposes candidates from a constrained primitive library, but execution is blocked unless preconditions and evidence requirements are satisfied. Missing evidence triggers bounded read-only evidence gathering before another action is considered.

The predicate is deliberately conservative. If $Ev(a)$ is not satisfied, EGS does not automatically downgrade to a weaker action. Rather, it asks certain evidence items, which makes the following loop quantifiable and verifiable.

4.4 Search control under budgets

The controller is in charge of maintaining the loop at pressure. Two rules are of most practical importance:

- **No-repeat without new evidence.** when action a failed recently and E_t hhas not evolved to respond to the failure mode, a repeat of action a is considered churn and is rejected.
- **Cycle prevention.** The typical cycles are restart loops, redeploy loops and rollback-forward oscillations. The controller keeps a short action history fingerprint and prevents transitions that recreate known cycles unless an explicit override condition is satisfied (such as an environment change or a newly observed piece of evidence that breaks the symmetry).

These controls ensure that exploration is limited but can be escalated when the evidence is in favor of it.

Algorithm 1 Evidence-Gated Search Control Loop

```
Require: initial state $S_0$, action library $A$, budgets $B$, policies $\Pi$, proposer top-k
 1: $t \leftarrow 0$, history $\leftarrow \emptyset$
 2: while $\neg$Healthy($S_t$) and $\neg$BudgetExhausted($B$) do
 3:     $C_t \leftarrow$ ProposeTopK($S_t$, $A$)                          ▷ LLM proposer
 4:     $A_t \leftarrow \{a \in C_t : \text{Admit}(a, S_t)\}$             ▷ evidence + policy checks
 5:     if $A_t = \emptyset$ then
 6:         $Q_t \leftarrow$ MissingEvidenceRequests($C_t$, $S_t$)
 7:         $S_t \leftarrow$ GatherReadOnly($S_t$, $Q_t$)
 8:         Consume($B_{read}$)
 9:         continue
10:     end if
11:     $a^\star \leftarrow$ Select($A_t$, history, $B$)                  ▷ risk-aware, no-repeat, no-cycle
12:     $(o_t, \Delta_t) \leftarrow$ Exec($a^\star$, $S_t$)
13:     $S_{t+1} \leftarrow$ UpdateState($S_t$, $o_t$, $\Delta_t$)
14:     history $\leftarrow$ Append(history, $a^\star$, $o_t$)
15:     Consume($B_{act}$, $B_{risk}$, $B_{time}$)
16:     $t \leftarrow t + 1$
17: end while
```

5. Search Controller and Evidence Rules

EGS suppresses operational search explosion by restricting remediation to a finite primitive library and enforcing evidence requirements before state-changing actions run. The proposer emits top-k candidates from A with explicit evidence claims. The evidence gate admits an action only when preconditions, evidence, policies, and budgets hold (Sec. IV). If no action is admissible, the loop spends budget on bounded read-only evidence gathering and then re-proposes.

5.1 Evidence templates

Table 1 summarizes the evidence requirements used in our implementation.

Table 1. Evidence templates for representative primitives (abbreviated).

Primitive	Required evidence (examples)
Restart(svc)	Health check failure or error spike
Redeploy(svc,build)	Deploy or pipeline failure tied to build
Rollback(svc,prev)	Regression-after-deploy, migration-safe
RevertConfig(key)	Config change near onset + signature match
FixAccess(p,res,role)	403/auth denied + role missing or IaC drift
FixDNS/PE(target)	NXDOMAIN or wrong IP + probe failure

5.2 OSE suppression rules

EGS enforces three rules that directly reduce fan-out and churn: (i) top-k pruning bounds breadth, (ii) no-repeat without new evidence blocks restart or redeploy thrash, and (iii) cycle prevention blocks common loops. Among admissible actions, selection is risk-aware, preferring lower $r(a)$ and lower $c(a)$, and escalating only when the evidence supports it.

6. Evaluation

We evaluate whether Evidence-Gated Search (EGS) suppresses operational search explosion (OSE) without sacrificing recovery performance. The evaluation reports (i) search behavior, including executed action volume, candidate fan-out, and budget-weighted search cost, and (ii) recovery outcomes, including success and lightweight safety proxies.

6.1 Dataset and incident episodes

Experiments use incident episodes derived from the sock-shop microservice traces packaged as sock-shop.zip and sock-shop-2.zip. Each episode is labeled with a fault category (e.g., svc_crash, bad_config, dns_fail, auth_denied). At each step, the simulator constructs a structured incident state S_t and evidence set E_t from the episode's fault label and evidence availability, exposing only what would be observable under the episode's telemetry conditions. Evidence availability is simulated by exposing only a subset of telemetry signals at each step of an episode. This models realistic operational conditions in which logs, metrics, deployment outputs, or infrastructure signals may arrive with delay or may require explicit retrieval. Additional evidence becomes visible only through bounded read-only queries executed by the evidence gatherer. In this way, the simulator approximates production telemetry delays and partial observability without allowing unrestricted access to all evidence at the initial step.

6.2 Compared methods (baselines)

We compare two remediation loops that share the same action library and proposer, differing only in whether evidence requirements block execution:

- **Baseline (ungated).** The proposer selects from top-k candidates and execution proceeds when basic preconditions hold. Evidence is advisory and does not prevent state-changing steps.
- **EGS (ours).** The same top-k proposer is used, but the evidence gate enforces $Ev(a) \subseteq E_t$ and hard constraints Π. When required evidence is missing, the loop switches to bounded read-only evidence gathering (consuming B_{read}) and then re-proposes.

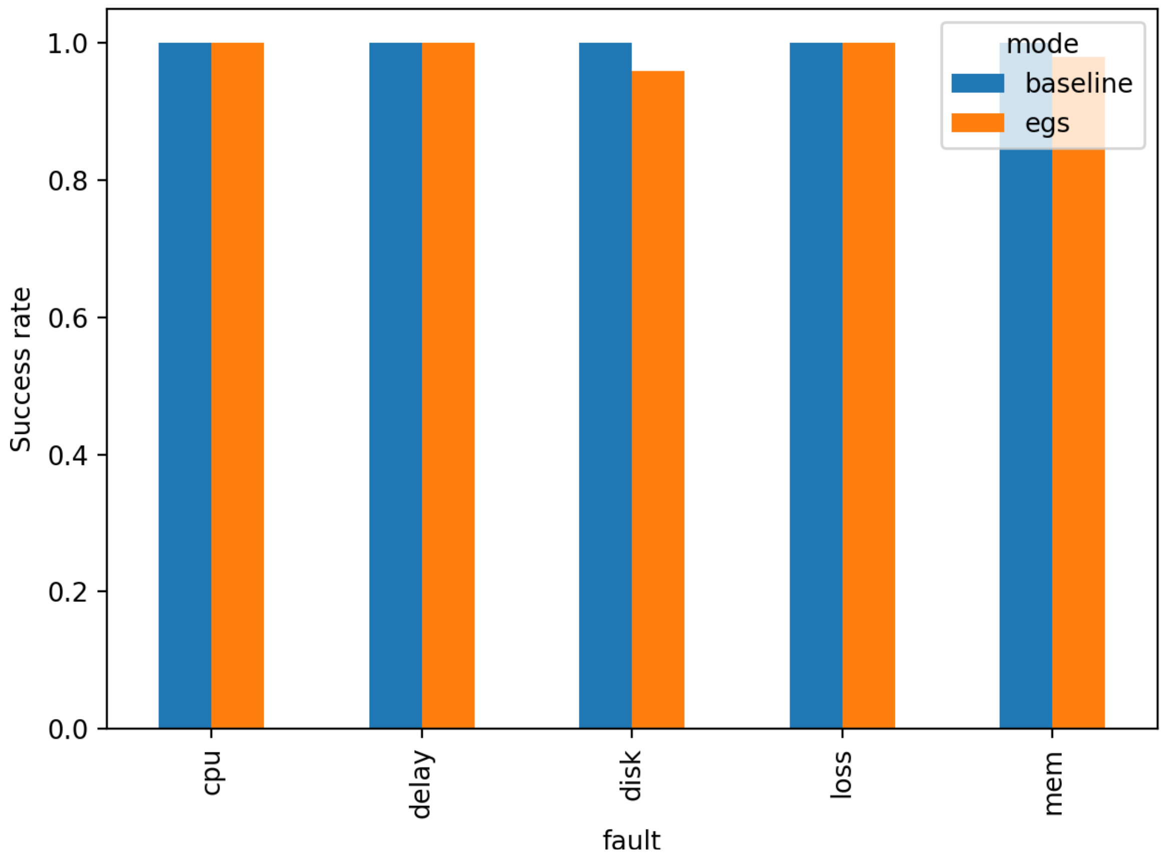

Fig. 2. Recovery success rate by fault type (EGS vs baseline).

6.3 Simulator Assumptions and Limitations

The analysis is based on a simulator under control that re-executes incident episodes based on the sock-shop microservice traces. The simulator reveals structured evidence signals, depending on the fault label of each episode and a predetermined evidence availability schedule.

The simulator assumes: (i) the action primitive library is finite and represents typical operational remediation actions, (ii) telemetry data like logs, metrics, traces, and deployment outputs can be modeled into structured pieces of evidence, and (iii) the incident state is deterministically evolved following every action performed.

With these assumptions, it is possible to compare ungated remediation and evidence-gated remediation in repeatable conditions. Nevertheless, the simulator fails to model all of the characteristics of real production systems, including simultaneous operator interventions, incomplete cross-service visibility, stochastic infrastructure behavior, or organization-specific escalation processes. Based on this, the results reported must be viewed as controlled operational experiments and not complete production replay.

6.4 Illustrative Example of Evidence Blocking

Take a deployment case where the error rate of the service increases as soon as a configuration change is made. The proposer of the LLM can prioritize two possible actions: to restart the service and to revert the modified configuration key.

For the primitive Restart(svc), tthe evidence template must be confirmed to have failed health-check or a persistent error spike of the target service. When the current incident state has a recent configuration diff but no evidence of service-level failure, the evidence gate prevents the restart action.

The controller does not restart the application speculatively, but rather asks the application to provide more read-only evidence, including recent application logs, metric windows, and deployment metadata. After the evidence confirms that the configuration change is time correlated with the failure, the primitive RevertConfig(key) is admissible and can be safely run.

This example demonstrates that evidence gating avoids state changes that are loosely justified and shifts the loop to actions that are justified by concrete operational evidence.

6.5 Simulator and implementation details

The controller follows Alg. 1. The remediation library A includes restart, redeploy, rollback, revert configuration, fix access, and fix DNS/private endpoint primitives. Each primitive specifies Pre(a) and Ev(a), and is annotated with risk r(a) and cost c(a). Budgets limit both state-changing actions and read-only queries per episode. To isolate the effect of evidence gating, both methods use identical proposer settings, the same A, and the same budget configuration; only the admissibility checks and controller constraints differ.

6.6 Metrics

We report:

- **Recovery success rate:** fraction of episodes reaching Healthy(S_t) within the episode budgets.
- **Executed actions** N_{exec}**:** number of state-changing primitives executed per episode.
- **Candidate fan-out:** summarized as $AER = \frac{1}{T}\sum_{t=1}^{T} b_t$ (Sec. III).
- **Read-only queries** N_{read}**:** number of evidence gathering operations executed per episode.
- **Budget-weighted operational search cost (OSC):** the budget-weighted cost defined in Sec. III, which penalizes disruptive or high-risk actions more than read-only checks.

6.7 Overall results

The evaluation reports three primary operational search metrics: executed action volume (N_{exec}), action explosion rate (AER), and budget-weighted operational search cost (OSC), along with recovery success. Table 2 summarizes mean performance across all episodes.

EGS reduces both action volume and search cost. Relative to the baseline, mean executed actions drop from 1.143 to 0.989 (13.47%), mean action explosion rate (AER) drops from 6.001 to 5.000 (16.65%), and mean OSC drops from 7.114 to 6.203 (12.80%). These gains come with a modest increase in read-only evidence gathering (mean $N_{read} = 0.377$ under EGS) and a small decrease in success rate (EGS 98.86% vs baseline 100%). The failure cases are consistent with the intended conservatism of strict gating: when required evidence does not become available within the read-only budget, some episodes cannot progress to an admissible state-changing action before the episode budget is exhausted.

Table 2. Overall evaluation results (mean per episode).

Method	Incidents	Success (%)	Exec. Actions ↓	Read-only ↓	AER ↓	OSC ↓
Baseline	175	100.0	1.14	0.00	6.00	7.11
EGS	175	98.9	0.99	0.38	5.00	6.20

Table 3. Mean executed remediation actions per episode by fault type.

Fault	Baseline	EGS
cpu	1.08	1.00
delay	1.28	1.00
disk	1.20	0.96
loss	1.16	1.00
mem	1.10	0.98

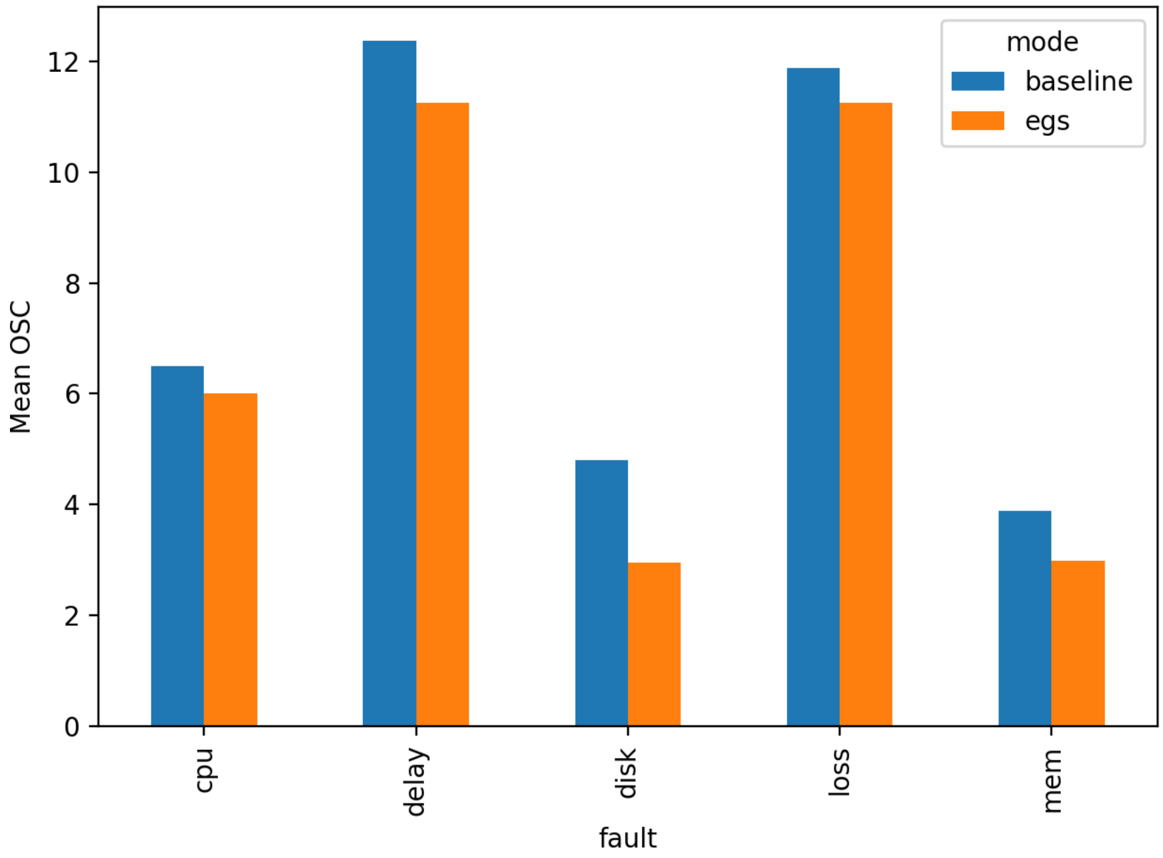

Fig. 3. OSC per episode by fault type (EGS vs baseline).

6.8 Results by fault type

Table 3 breaks down action volume and search cost by fault category.

Across fault types, EGS executes fewer state-changing actions than the baseline. The largest reductions appear in categories with several plausible remediations (notably configuration and connectivity-related faults), where ungated loops are most likely to drift into restart or redeploy churn.

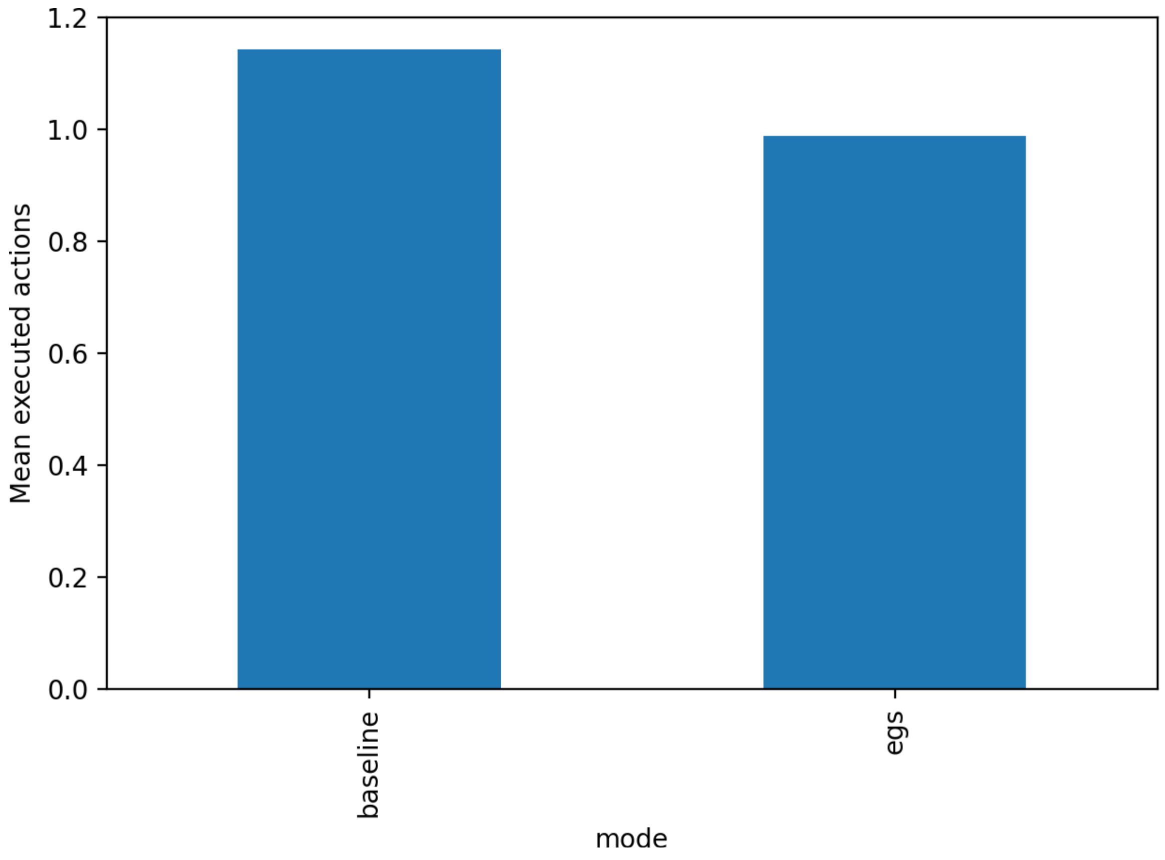

Fig. 4. Mean executed actions per episode (lower is better).

7. Discussion

EGS reduces operational search explosion by forcing evidence to be the currency for state-changing remediation. The most significant decreases are observed in cases where there are numerous plausible fixes (configuration and connectivity cases), and ungated loops are likely to thrash with restarts and redeploys. The extra read-only overhead is anticipated, and operationally desirable, since it substitutes risky changes with limited evidence acquisition. The slight decrease in success-rate with strict gating indicates a tuning knob: increase B_{read} and/or weaken evidence templates of low-risk actions and maintain stringent conditions of irreversible actions (rollback, access policy, network changes).

8. Conclusion and Future Work

In this paper, Evidence-Gated Search (EGS), a control-loop architecture, was presented, which views incident response as budgeted search through a limited primitive library, and prevents state-changing remediation until explicit evidence conditions are met. EGS minimizes the volume of actions, action fan-out, and budget-weighted search cost and does not significantly decrease recovery success by combining evidence gating with a risk-aware controller (top-k pruning, no-repeat, and cycle prevention). Future directions are (i) more evidence schemas with confidence and temporal decay, (ii) stronger baselines like human-in-the-loop approval of high-risk actions, (iii) extending the simulator to replay on real incident timelines with delayed telemetry, and (iv) learning organization-specific r(a) and c(a) based on the outcome of historical changes.

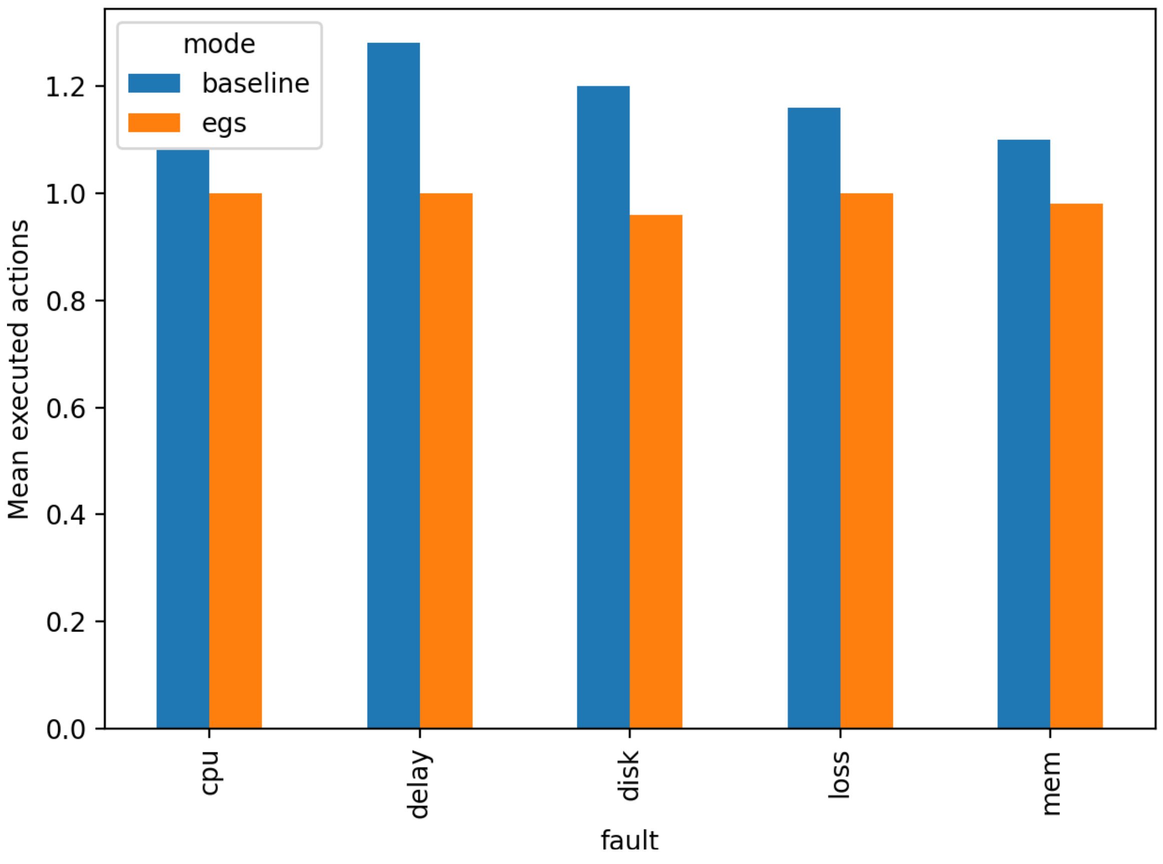

Fig. 5. Actions per episode by fault type (EGS vs baseline).

Funding: This research received no external funding
Conflicts of Interest: The authors declare no conflict of interest.
ORCID iD: 0009-0004-5810-0431
Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers.

References

1. Brandón, Á., Solé, M., Huélamo, M., Jiménez-García, D., Pérez, J.: Graph-based root cause analysis for service-oriented and microservice architectures. Journal of Systems and Software **159** (2020). https://doi.org/10.1016/j.jss.2019.110432
2. Du, M., Li, F., Zheng, G., Srikumar, V.: Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS) (2017). https://doi.org/10.1145/3133956.3134015
3. He, P., Zhu, J., Zheng, Z., Lyu, M.R.: Drain: An online log parsing approach with fixed depth tree. In: 2017 IEEE International Conference on Web Services (ICWS). pp. 33–40 (2017). https://doi.org/10.1109/ICWS.2017.13
4. Notaro, P., Albanese, L., De Simone, A., Nardelli, M.: A survey of aiops methods for failure management. ACM Computing Surveys (2021). https://doi.org/10.1145/3483424
5. 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 Web Conference (WWW Companion) (2025). https://doi.org/10.1145/3701716.3715290
6. Spring, J.M., Illari, P.: Review of human decision-making during computer security incident analysis. ACM Computing Surveys (2021). https://doi.org/10.1145/3427787

7. Wang, H., Wu, Z., Jiang, H., Huang, Y., Wang, J., Kopru, S., Xie, T.: Groot: An event-graph-based approach for root cause analysis in industrial settings. In: 2021 IEEE/ACM 36th International Conference on Automated Software Engineering (ASE) (2021). https://doi.org/10.1109/ASE51524.2021.9678708
8. Weng, J., Wang, J.H., Yang, J., Yang, Y.: Root cause analysis of anomalies of multitier services in public clouds. IEEE/ACM Transactions on Networking **26**(4), 1646–1659 (2018). https://doi.org/10.1109/TNET.2018.2843805
9. Wu, L., Tordsson, J., Elmroth, E., Kao, O.: Microrca: Root cause localization of performance issues in microservices. In: 2020 IEEE/IFIP Network Operations and Management Symposium (NOMS). pp. 1–9 (2020). https://doi.org/10.1109/NOMS47738.2020.9110353
10. Xie, Z., Xu, H., Chen, X., et al.: Microservice root cause analysis with limited observability. In: Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) (2024). https://doi.org/10.1145/3637528.3671530
11. Yao, Z., Pei, C., Pei, D., et al.: Chain-of-event: Interpretable root cause analysis for microservices through automatically learning weighted event causal graph. In: Proceedings of the ACM International Conference on the Foundations of Software Engineering (FSE) (2024). https://doi.org/10.1145/3663529.3663827
12. Zhang, S., Zhang, Y., Xu, H., et al.: Failure diagnosis in microservice systems. ACM Computing Surveys (2025). https://doi.org/10.1145/3715005
13. Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., Lyu, M.R.: Tools and benchmarks for automated log parsing. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). pp. 121–130 (2019). https://doi.org/10.1109/ICSE-SEIP.2019.00021
14. Zhu, Y., Wang, J., Li, B., Zhao, Y.: Microirc: Instance-level root cause localization for microservice systems. Journal of Systems