Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 09 , September - 2026

Automated Fault Remediation and Self-Healing in Kubernetes: A Survey of Failure Scope, LLM Roles, and Post-Remediation Verification

Arya Uday Singh
Department of Information Science and Engineering
RV Institute of Technology and Management
Bengaluru, India

Ananya P
Department of Information Science and Engineering
RV Institute of Technology and Management
Bengaluru, India

Subhashini B
Department of Information Science and Engineering
RV Institute of Technology and Management
Bengaluru, India

***Abstract*—Automated fault remediation and self-healing for Kubernetes-orchestrated microservice systems has moved rapidly from anomaly detection toward large language model (LLM) mediated diagnosis and action selection, yet the resulting body of work has not been assessed as a whole. This paper surveys 23 papers published between 2022 and 2026, separating seven core systems that execute a remediation action against a running Kubernetes or closely related cluster from sixteen framing sources that inform diagnosis, benchmarking, or taxonomy without themselves acting. Each core system is compared along thirteen dimensions spanning failure scope, autonomy level, the role assigned to the LLM, the action space, safety mechanisms, outcome verification, rollback, evaluation environment, and reproducibility. The corpus converges strongly on evaluation infrastructure: demo microservice applications on Kubernetes, injected resource and network faults, and Prometheus or OpenTelemetry based observability recur across a clear majority of the reviewed work. It diverges sharply, however, on what constitutes a verified recovery, with seven core systems applying seven largely incompatible definitions, and no core system reports a rollback rate measured over a population of remediation attempts. Reported accuracy for LLM-driven action selection on comparable Kubernetes settings differs by forty to sixty percentage points across three studies, a discrepancy this review leaves unresolved rather than adjudicated. The evidence indicates that the field's open problems concentrate after an action executes rather than in how the action is selected, and that further increases in autonomy will remain difficult to assess until post-remediation evaluation is standardized.**

***Index Terms*—automated remediation, fault tolerance, Kubernetes, large language models, root cause analysis, self-healing**

I. Introduction

Kubernetes has become the dominant orchestration layer for microservice deployment, and with that dominance has come a proportional growth in the operational burden of keeping deployed workloads healthy. A single cluster routinely hosts services owned by different teams, each with independent deployment cadences and failure modes, and an incident in one service can propagate to others through shared dependencies. An empirical study of Kubernetes operator bugs found that only 55% of examined defects were detectable by the best combination of existing tooling, and that 54% produced only silent failures with no immediate operator-visible signal [12]. Under these conditions, manual triage does not scale: the volume and heterogeneity of failure signatures across a large cluster exceeds what an on-call engineer can reliably diagnose within a service level objective, and the delay between fault onset and human intervention directly extends downtime.

This pressure has driven a shift from static, rule-based recovery toward systems that use large language models to interpret telemetry, propose a diagnosis, and in a growing number of cases select or generate a remediation action. A recent survey of AIOps research spanning 183 papers documents this shift at scale, and notes that its own taxonomy of automation levels contains no category for verifying or reversing an action once taken [1]. Independent recent work quantifies a further consequence of this shift directly: even where an LLM diagnoses a failure with near-perfect accuracy, the same model selects a valid recovery action in as few as 36.8% of cases, and in some fault categories the valid-action rate falls to zero [2]. Another recent system reports 96.67% overall recovery accuracy on its own Kubernetes fault set [15], while a third reports that no standalone general-purpose model it evaluated exceeded 50% accuracy even at the easiest tier of its benchmark [3]. These figures are not obviously reconcilable, and no existing review has attempted to place them side by side.

This paper makes the following contributions.

1) A cross-paper extraction matrix comparing seven core Kubernetes fault-remediation systems along thirteen dimensions, including failure scope, autonomy level, LLM role, action space, safety mechanism, outcome verifica-

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 09 , September - 2026

tion, and rollback.

2) An explicit synthesis of convergence and divergence across the corpus, including a documented forty to sixty percentage point spread in reported LLM action-selection accuracy across three studies evaluating comparable Kubernetes settings.
3) A set of seven derived gaps in post-remediation verification, safety enforcement, and reproducibility, each attached to an explicit evidence class rather than asserted from silence alone.
4) A consistent methodological distinction, applied throughout, between a capability a system is shown not to have and a capability that a paper simply does not report, which prior narrower comparisons have not maintained.

The remainder of this paper is organized as follows. Section II describes the review methodology, including corpus construction, inclusion and exclusion criteria, and the operational definition of each comparison dimension. Section III establishes background and taxonomy. Sections IV through IX analyze the corpus along failure scope, autonomy and human gating, the role of LLMs, safety verification and rollback, and evaluation and reproducibility practice, respectively. Section X synthesizes the core comparators into architectural families. Section XI consolidates consensus and open challenges. Section XII states threats to validity, including the corpus omissions declared in Section II. Section XIII concludes.

II. Review Methodology

A. Corpus Construction

The corpus was assembled through targeted and citation-chained collection rather than a systematic database query. Candidate papers were identified through direct submission by the authors, citation chasing from an initial seed of Kubernetes self-healing and LLM-for-operations papers, and cross-referencing among the retrieved papers themselves, several of which cite one another directly. No fixed search string was run against a bibliographic database, and this is treated as a limitation of the review rather than concealed; it is restated in Section XII. Retrieved papers span publication years 2022 through 2026, with the majority concentrated in 2024 through 2026.

Two categories of omission are declared here and detailed in Section XII. First, six papers were submitted for inclusion but had not been processed into the extraction matrix at the point the corpus was frozen; their titles are retained as candidates for a future revision rather than discarded. Second, eleven well-known works in this research area, including established agentic-cloud-operations benchmarks and an earlier full-autonomy system from an author group otherwise represented in the corpus, were identified as relevant during the review process but were never collected or extracted, and are therefore absent from every table, figure, and claim in this paper. Both omissions are named individually in Section XII rather than folded into the discussion sections, so that no analytical claim in Sections IV through XI rests on a paper this review did not actually examine.

TABLE I
Inclusion and Exclusion Criteria

Type	Rule
Inclusion	Addresses fault detection, diagnosis, or remediation in Kubernetes or a closely adjacent cloud-native or microservice setting.
Inclusion	Reports an evaluated system, an empirical study, a benchmark or dataset, or a survey providing taxonomy relevant to this scope.
Inclusion	Full text obtainable within the review window.
Exclusion	Full text not obtainable within the review window.
Exclusion	Substantive duplicate or subset of an already included report describing the same system and evaluation.

TABLE II
Screening Funnel

Stage	Count
Papers submitted for consideration	29
Excluded, not processed before corpus freeze	6
Included in frozen corpus	23
Classified core	7
Classified framing	16

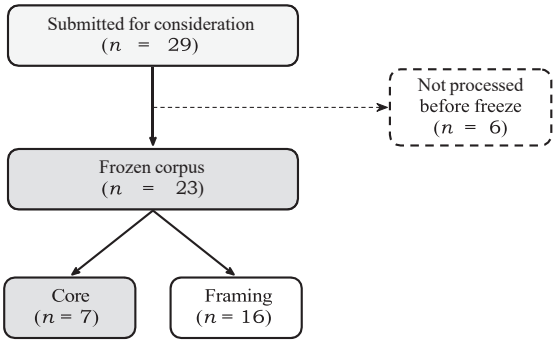

Fig. 1. Disposition of every paper submitted for consideration. Solid arrows denote inclusion, the dashed arrow denotes the six papers named in Section XII that were submitted but not extracted before the corpus was frozen. Counts correspond to Table II.

B. Inclusion and Exclusion Criteria

A candidate paper was included if it satisfied all criteria in Table I. Exclusion was applied only where a candidate's full text could not be obtained within the review window; no candidate was excluded on topical grounds once retrieved, since candidates were sourced under the same topical constraint.

C. Screening Funnel

Table II and Figure 1 report the disposition of every paper submitted for consideration, including the six not processed before corpus freeze.

D. Core and Framing Classification

Each included paper was assigned to exactly one of two tiers. A paper was classified *core* if it presented a system that

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 09 , September - 2026

executes, or directly generates for execution, a remediation action against a running Kubernetes cluster or a functionally equivalent orchestrated environment, and reported an evaluation of that system. A paper was classified *framing* if it did not meet this bar, including surveys, benchmarks and datasets, tool papers restricted to detection or advisory output, and empirical studies of failure characteristics that do not themselves remediate. Seven papers were classified core [3], [4], [8], [15], [16], [18], [19] and sixteen framing [1], [2], [5]–[7], [9]–[14], [17], [20]–[23].

E. Comparison Dimensions

Core comparators were compared along the following thirteen dimensions. Each definition below is operational, stating what was recorded rather than what was inferred.

1) **Failure scope**: the named classes of fault the system is designed to address, as stated by the source paper.
2) **Autonomy level**: a six-point rubric, L0 through L5, defined in Section III, coding how much of the detect-diagnose-act loop executes without human decision or approval.
3) **LLM role**: the specific function assigned to a large language model in the pipeline, if any, distinct from the system's overall autonomy level.
4) **Action space**: the set of operations the system is capable of issuing against the target environment, whether a fixed enumerated set or an unbounded generated artifact.
5) **Safety mechanisms**: any mechanism reported as constraining, validating, or gating an action before or during execution.
6) **Verifies outcome**: whether the paper reports an explicit check, after an action executes, that the targeted failure was resolved.
7) **Rollback**: whether the paper reports a mechanism to revert an executed action if the targeted failure was not resolved or worsened.
8) **Evaluation environment**: the physical or virtual infrastructure on which the system was evaluated.
9) **Workload**: the application or applications subjected to faults during evaluation.
10) **Fault injection**: the tool or method used to induce failures for evaluation.
11) **Metrics reported**: the quantitative outcomes stated in the paper.
12) **Code released**: whether source code was reported as publicly available.
13) **Data released**: whether evaluation data or benchmark artifacts were reported as publicly available.

F. Extraction Protocol

Extraction was performed by a single annotator across all included papers, recording each dimension directly against source text. Where a paper did not state a value for a dimension, the cell was recorded NR, not reported, rather than inferred as absent. Section V and Section VII apply this distinction explicitly: a system is described as lacking a capability only where the source paper states so directly or where the absence is structurally certain from the described architecture, and is otherwise described only as not reporting that capability. No second annotator reviewed the extraction, and this is treated as a limitation in Section XII.

III. Background and Taxonomy

A. Kubernetes Failure Modes

Failures in Kubernetes-orchestrated systems arise at multiple layers, including container-level crashes, resource exhaustion, misconfiguration, and network disruption between services. An empirical characterization of real Kubernetes operator defects found that 55% were detectable by the best combination of existing tooling and that 54% produced only silent failures, with 60% of root causes traced to incorrect observation of system state [12]. This characterization describes production defects directly, in contrast to the fault taxonomies used for evaluation across the corpus, which are examined in Section IV.

The fault classes most commonly used for evaluation across the corpus are resource exhaustion and network perturbation. The most complete taxonomy in the corpus enumerates four resource faults, two network faults, and five code-level faults [11]. CrashLoopBackOff, one of the most frequently operator-visible Kubernetes failure signatures, appears in the corpus only as a downstream manifestation rather than as a fault class scoped for evaluation in its own right: one core system states that all fifteen of its injected faults leave the pod unavailable in CrashLoopBackOff, FailedCreate, or Pending state, and treats return to a Running and traffic-ready state as its recovery criterion [15]. No paper in the corpus organizes its fault taxonomy around CrashLoopBackOff as a named class, and none reports results broken out for it separately.

B. A Six-Level Autonomy Scale

The systems reviewed in this paper do not share a common vocabulary for describing how much of the detection, diagnosis, and action process executes without human decision or approval. To allow comparison across systems on this dimension, this review employs a six-point scale, denoted L0 through L5, applied uniformly in Sections V through X. A related five-level automation taxonomy for AIOps has been proposed independently in prior work [1]; the scale used here is not identical to that taxonomy and is defined specifically around the detect-diagnose-act sequence relevant to Kubernetes remediation.

L0 The system displays raw telemetry only; a human performs detection, diagnosis, and action.
L1 The system detects an anomaly; a human performs diagnosis and action.
L2 The system diagnoses or ranks candidate root causes; a human decides on and executes an action.
L3 The system proposes a specific action; a human approves the action before it executes.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 09 , September - 2026

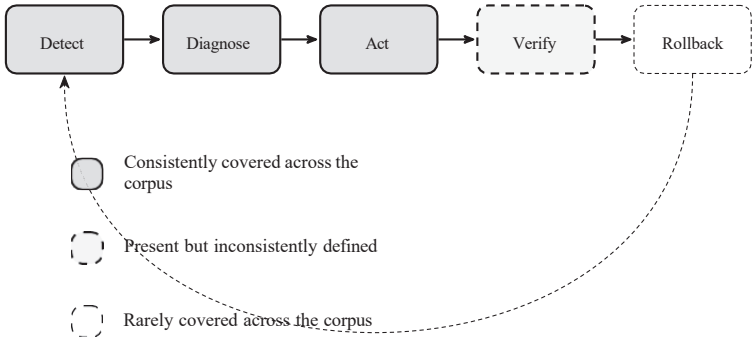

Fig. 2. Detect-diagnose-act-verify-rollback reference loop used as the comparison frame in this review. Shading indicates how consistently each stage is covered across the seven core comparators.

L4 The system selects and executes an action autonomously, constrained by pre-set limits established before execution.

L5 The system selects and executes an action autonomously with no constraint gate on the action taken.

This scale is descriptive rather than normative. It records how much decision authority a system retains for itself, and does not by itself indicate that a higher level is preferable to a lower one; Section V reports that the corpus is in fact divided on this question.

C. Reference Loop

The systems in this corpus descend, directly or indirectly, from the Monitor-Analyze-Plan-Execute over shared Knowledge (MAPE-K) control loop. One corpus paper instantiates an explicit MAPE-K variant for cloud-native resource management [6], and another implements a formal, dependency-aware planning loop for multi-cloud application recovery [7]. A separate corpus paper supplies a decomposition of recovery time into reaction time, recovery time, repair time, and outage time [8], a vocabulary reused in later sections of this review.

For comparison purposes, this review adopts a five-stage reference loop: detect, diagnose, act, verify, and rollback. Figure 2 depicts this loop and marks, for each stage, whether it is consistently present across the corpus, present but inconsistently defined, or rarely present. This reference loop extends the taxonomies already published in the corpus rather than adopting one of them directly, because none of the surveyed taxonomies, including the peer-reviewed AIOps survey spanning 183 papers, contains a category for verifying or reverting an executed action [1]. Sections V through X use this five-stage loop as the frame against which every core comparator is analyzed.

IV. Failure Scope

A. Convergence

Faults across the corpus are injected rather than observed; no paper derives its evaluation from naturally occurring incidents. Eleven corpus papers report an explicit fault injection method, including Chaos Mesh [2], [3], [14], stress-ng combined with traffic control [11], [20], Envoy combined with K6 [21], docker-stress [6], and bespoke injection mechanisms [7], [8], [13], [15]. This is the single strongest point of methodological agreement identified across the corpus.

A further convergence concerns the number of faults active during a single evaluation case. Six papers inject exactly one fault per case throughout [2], [3], [11], [14], [20], [21]. Three depart from this pattern. A trans-cloud application management system runs twenty-four hours of randomized, overlapping, and cascading failure injection [7]. A Kubernetes-native availability controller kills one to five active pods simultaneously in a dedicated experiment [8]. An LLM-bearing Kubernetes system runs the bulk of its evaluation one fault at a time but devotes a separate research question to concurrency, grouping non-interacting faults and injecting them in parallel [15]. Only the first of the three injects faults that interact with one another.

B. Divergence

The breadth of failure scope addressed by individual core systems varies widely, and this variation is empirical rather than definitional. One system addresses seven fault types plus configuration faults [3]; another injects fifteen individual pod-level faults drawn from two categories, ten resource-management and five scheduling faults [15]; a third addresses eight failure types through diagnosis and action selection [18]. At the narrow end, one system addresses a single class, stateful pod failure [8], and another is scoped to service level objective regressions and a noisy-neighbor scenario [4]. A sixth system evaluates against three datasets with mixed fault composition [19]. This spread means that a reported success rate from a narrow-scope system and a reported success rate from a broad-scope system answer substantially different questions, even where both are expressed as a single accuracy figure.

C. Silences

Two silences are identified in this dimension. The first concerns interacting failure specifically, and it is narrower than concurrency. Three corpus papers evaluate multiple simultaneous faults. A trans-cloud system injects overlapping and cascading failures over twenty-four hours [7]; a Kubernetes-native availability controller fails up to five active pods at once and reports that outage time grows for each successive pod because its controller processes failure events in first-in-first-out order [8]; and an LLM-bearing Kubernetes system groups its faults and injects them in parallel, reporting that mean time to recovery scales linearly with the number of concurrent faults [15]. The last of these is important because it removes concurrency itself from the list of things the LLM-bearing subset has not examined.

What remains unexamined is interaction. [15] states that the faults injected in parallel are those which do not influence each other, so its result describes throughput under independent load rather than recovery under a fault that propagates. The

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 09 , September - 2026

TABLE III
AUTONOMY PLACEMENT OF CORE COMPARATORS

Ref	System	Level
[3]	E2E-REME	L5
[15]	Wiesinger et al.	L5
[8]	HA State Controller	L4
[18]	LLM plus Observability	L4
[4]	ARBITER	L3
[16]	Sarda et al.	L2
[19]	GALR	L2

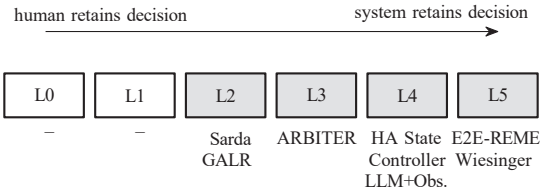

Fig. 3. Placement of the seven core comparators on the six-level autonomy scale defined in Section III. Shaded levels are occupied; L0 and L1 are empty, which follows from the inclusion criterion that a core comparator must execute or directly generate a remediation action. Corresponds to Table III.

only corpus paper that injects genuinely cascading failure is neither Kubernetes-native nor LLM-bearing [7]. No system in this corpus that uses an LLM has been evaluated against a failure that induces further failures, which is the condition under which a wrong remediation is most consequential, and this review records that as the structural silence rather than concurrency in general.

The second silence concerns the relationship between injected fault taxonomies and real operator defects. The corpus paper that characterizes real Kubernetes operator bugs reports that 54% of examined defects produce only silent failures [12], a failure signature that does not correspond cleanly to any of the injected fault classes used elsewhere in the corpus. Whether the systems evaluated against injected resource and network faults would detect or remediate this class of real-world silent failure is not addressed by any paper in the corpus, and this review treats that question as open rather than answered in either direction.

V. AUTONOMY AND HUMAN GATING

A. Convergence

Placed against the six-point scale defined in Section III, the seven core comparators occupy only four of its six levels, and none sits at L0 or L1. Table III lists the placement for every core comparator, and Figure 3 shows the same placement on the scale itself. Four of seven execute an action autonomously, at L4 or L5 [3], [8], [15], [18]; one proposes actions through an approval-required path, at L3 [4]; and two stop before execution, at L2 [16], [19]. The scale's lower positions, L0 and L1, are empty across the entire core corpus, which is expected given that inclusion as a core comparator required executing or directly generating a remediation action.

Within L5, [3] executes a generated Ansible playbook whose only stated constraint is a penalty applied during training rather than a check applied at the point of execution, though a bounded retry limit is applied during generation; whether that retry bound should be read as a partial runtime constraint is not settled by the source text, and the system is placed at L5 on the basis that no check gates the executed action itself. [15] applies no check of its own before an arbitrary generated YAML edit reaches the cluster: the executor submits the generated manifest directly to the Kubernetes API server, and the only validation in the path is the API server's own admission and schema checking, which occurs at the point of application rather than before it. A rejected manifest returns an error that the planner may then attempt to correct on a subsequent try. The source paper's own claim of complete autonomicity is consistent with this placement.

Within L4, [8] is the sole non-LLM core comparator; its single action, a deterministic label flip governed by a fixed state machine, is bounded by construction rather than by a separate validation layer, and its placement at L4 reflects deterministic scope rather than model-mediated constraint. [18] is placed at L4 because action selection is reported as gated by a confidence threshold before execution, with incidents below a stated threshold escalated to a human operator, and because a separate pre-execution safety layer, detailed in Section VII, further constrains which proposed actions may execute.

[4] is th orpus's sole L3 system and the most tightly bounded system in it. It enforces a schema check, an allowlist, a deny-list, four-axis budget caps, and quota, disruption-budget, and maintenance-window constraints evaluated on every action before execution. Its placement at L3 rather than L4 follows from the source paper's own account of what was evaluated: plans are created pending approval, and the paper states directly that all reported experiments used approval-required execution, with the experiment harness rather than a human operator granting approval for repeatability. The design does provide an auto-allow confidence threshold above which an action would execute without approval, and the paper names observe-only, canary-auto, and fully guarded autonomous modes as compatible policy settings, but records that none of these was enabled or evaluated. This review codes the configuration the paper reports rather than the configurations its architecture permits, and notes that under the auto-allow path the same system would sit at L4.

Within L2, [16] has a human operator author both the root-cause diagnosis and the specific remediation action as a natural-language instruction; the model's role is limited to translating that instruction into an Ansible playbook. The source paper's architecture describes its Generate and Act component as converting the plan into code and executing it, but the reported evaluation did not exercise that path: the authors state that they manually executed the generated playbooks and validated them through manual testing. An earlier reading of this paper by the present review first described execution as remaining with the operator, then over-corrected to describe it as automatic; neither is right on its own. The

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 09 , September - 2026

accurate position is that automated execution is specified in the architecture and not demonstrated in the evaluation, which leaves both the decision and the executed action in human hands throughout the reported experiments and places the system at L2 for what it actually shows. [19] produces a ranked recovery plan scored against a reference plan by textual overlap, but the plan is never executed by the system itself, placing it at L2 despite the sophistication of its retrieval-augmented construction.

B. Divergence

Whether autonomous execution is treated as the goal to pursue or as a hazard to be contained splits the corpus along lines that are largely normative rather than strictly empirical. Two core systems escalate toward fully autonomous execution [3], [15], while one core system deliberately stops short of it [19], a position reinforced outside the core comparator set by a production-deployed root cause analysis system that stops short of remediation by design after more than four years of deployment across more than thirty teams [17], and by a guardrail tool that stops at detection and a text remediation hint with no execution path at all [9]. [4] occupies a further distinct position, retaining an approval gate on every evaluated action despite having the most developed safety substrate in the corpus, which suggests that a strong deterministic validator is treated by its authors as a precondition for autonomy rather than a substitute for human approval. [16] restricts autonomy at the diagnosis-and-selection stage, with a human operator authoring both the root cause and the chosen action. The single clearest data point for this divide is that [16] and an earlier full-autonomy system from the same author lineage represent a documented move away from autonomous diagnosis and action selection toward an explicitly human-authored design, a regression in decision-making autonomy rather than in execution autonomy specifically. Restraint in this corpus correlates with production exposure across these two cases, though two cases are sufficient to observe this pattern and not sufficient to generalize it.

C. Silences

No paper in the corpus, including the core comparators and the production-deployed system referenced above [17], reports a human override rate, a handoff frequency, or any measure of operator trust in a system's autonomous decisions. This silence extends even to the two systems whose design premise is that a human retains the decision [16], [19]; despite building an architecture around human decision-making, neither paper reports how often, or under what conditions, the human overrides or declines the system's diagnosis. Whether this data exists but goes unreported, particularly plausible for [17] given its multi-year production deployment, or was never collected, cannot be determined from the corpus, and this review records it as a reporting silence rather than a structural one.

VI. Role of LLMs

A. Convergence

A majority of LLM-bearing core systems restrict the model to a bounded position in the loop rather than granting it direct execution authority. Four of the six LLM-bearing core systems confine the model's output to a role short of unconstrained execution: [4] treats the model purely as a planner whose proposal is separately validated before any action executes; [16] confines the model to translating an already human-decided instruction into executable code; [19] confines the model to producing plan text that the source paper never executes; [18] pairs diagnosis and action selection with a confidence gate and a separate pre-execution validation sequence before execution, despite generating actions from an open-ended action-type list rather than a fixed vocabulary. Two systems depart from this pattern, [3] and [15], both of which allow a generated artifact, an Ansible playbook or an arbitrary YAML edit respectively, to reach the cluster without a comparable runtime check. Read together, a clear majority of the corpus has already reached, in practice, a shared position that a model's output should not reach a running cluster unconstrained, even though no paper states this as an explicit design principle for the field as a whole.

Where the corpus reports a specific model or model family, the emphasis leans toward smaller or off-the-shelf models used through prompting and retrieval rather than toward a model fine-tuned specifically for the recovery task. [15] evaluates low-parameter LLMs specifically, as its title states, and reports Mistral-7B as its best-performing configuration among the models tested. [4] reports evaluation across three Claude model variants used as the planning component, with performance compared against a static rule-based baseline rather than against each other as separately fine-tuned systems, consistent with a prompting-based rather than fine-tuning-based construction. [19] constructs its recovery plans through retrieval-augmented generation over a graph-derived context rather than through a model fine-tuned for the recovery task specifically. [3] is the one core system whose description includes a training-time mechanism, a penalty applied during training, implying that some form of model training is part of its pipeline, though the frozen extraction does not specify which model family is trained or at which stage the penalty is applied. [16] evaluates two named models against each other, GPT-4 and LLaMa-2-70B, with the latter deployed on-premise for confidentiality-sensitive settings. [18] fine-tunes a named base model, Llama-2-13B, on operational incident data. [8] has no model component at all; it is the sole non-LLM core comparator, and this distinction is structural rather than a case of unreported model identity.

The corpus divides between single-shot model invocation and explicit multi-agent architecture. [3] is the only core system described as decomposing its pipeline into separate planner, executor, and verifier agents, an explicitly agentic design. The remaining core systems, where a model role is reported at all, apply the model to a single diagnosis, transla-

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 09 , September - 2026

tion, or generation step within a pipeline whose surrounding stages are handled by non-model components, for example the external validator in [4] or the graph-based localization stage feeding a single generation step in [19].

B. Divergence

Whether providing a model with more contextual information about a failure improves or degrades its performance divides the corpus in a way that appears contradictory on its surface. Within the core comparators, [4] reports that all three evaluated Claude models outperformed a static rule-based planner on a context the rules did not consult, using a bounded diagnosis context assembled from a four-layer causal graph, and [19] similarly builds its recovery-plan generation around retrieval-augmented context drawn from a graph representation. Outside the core comparator set, two framing sources report the opposite pattern: a production root cause analysis system deployed across more than thirty teams found that mixing additional context sources reduced its own accuracy from 0.766 to 0.440 in an ablation [17], and an empirical study of manifest generation found that prompt-only iterative refinement sometimes worsened results for smaller models, with the degradation traced to later refinement steps overwriting the output of earlier ones [10]. That same study, however, reports that replacing prompt-only refinement with tool-grounded refinement, feeding back real error messages from a Kubernetes dry-run, substantially improved outcomes and repaired most previously failing cases, so it is better read as distinguishing grounded from ungrounded feedback than as evidence that additional context is harmful in general. These findings are not drawn from directly comparable tasks or context-construction methods, so this review records the apparent contradiction rather than resolving it. A reading consistent with all four sources, though stated by none of them directly, is that context which is bounded, curated, or grounded in executable feedback, as in [4], [19] and in the second phase of [10], behaves differently from context that is broad and concatenated without equivalent curation, as in [17]; this reading is offered here as this review's own interpretation rather than as a finding any source paper states.

C. Silences

How each system constrains model output before it can affect the cluster is addressed in full in Section VII, since output constraint there is inseparable from the safety mechanisms examined in that section. No paper in the corpus reports a mechanism, a metric, or an evaluation specifically targeting hallucinated or fabricated diagnoses, as distinct from an incorrect but plausible one. The safety and gating mechanisms described above and in Section VII constrain what a model's output is permitted to do once produced, but none of the core papers reports testing whether the model's stated reasoning or claimed evidence for a diagnosis is itself accurate, as opposed to testing only whether the resulting action was valid or the resulting cluster state recovered. This review treats the absence of hallucination-specific evaluation as a reporting silence rather than a structural one, since a paper could in principle report such a check without foregrounding it, and the frozen extraction does not contain grounds to rule that out.

VII. Safety, Verification, and Rollback

A. Convergence

The one point of genuine agreement in this dimension is negative: verification and rollback are absent as standardized categories across the literature, not only across this corpus. A survey of 183 AIOps papers reports that its own taxonomy of automation levels contains no subtask for verifying or reversing an action once taken [1], establishing that this absence is a field-wide pattern rather than an artifact of how this corpus was assembled. Within the seven core comparators, this absence recurs directly: only one system, [4], subjects its safety mechanism to adversarial testing, and no core system reports a rollback rate measured over a population of remediation attempts rather than a small, individually described set of cases.

B. Divergence

The corpus splits, in a way that is more definitional than empirical, on where in the pipeline safety is enforced. [4] and [18] both enforce constraints at runtime, evaluated against every action before or during execution, through different mechanisms: [4] applies schema, allowlist, and budget checks, while [18] applies policy enforcement, simulated impact analysis, rate limiting, and dry-run validation in sequence before an action reaches the cluster. By contrast, [3] shapes safety at training time, through a penalty applied during model training rather than a check applied at inference, and [15] adds no check of its own at all, relying implicitly on the Kubernetes API server to reject malformed manifests at the moment they are submitted, which establishes that an artifact is well-formed but not that its effect is safe. These are not weaker and stronger versions of a single property; a training-time penalty and a runtime allowlist bind a system's behavior at different points and through different mechanisms, and the frozen extraction does not support ranking them on a single comparable scale of safety, only as differently constructed.

Placed against this dimension the seven core comparators yield seven largely incompatible definitions of what counts as evidence of recovery. [3] employs a dedicated status verification module that checks whether the specific injected fault was resolved. [4] applies a post-action monitor over an observation window and restores a pre-action snapshot if pod readiness degrades within that window, tying verification directly to an automatic reversal mechanism. [15] treats the absence of a newly detected issue as evidence the problem was fixed, a check that carries no stated time window or stability criterion; this review describes it as a weak verification rather than an absent one, following the source paper's own framing. [19] never executes its generated plan, and instead scores the plan by textual overlap against a reference action, which this review does not describe as outcome verification, since no outcome

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 09 , September - 2026

exists to verify. [18] verifies outcomes against three stated criteria, namely whether metrics return to normal ranges, whether logs indicate successful operation, and whether user-facing symptoms resolve, and ties this check to a progressive rollout mechanism that can halt or reverse a change if anomalies persist or worsen between stages. [8] performs no post-action check inside its control loop: on detecting a failed active pod it flips the standby pod's state label, and the newly active pod's entrypoint process resumes the service without any confirmation that the resumption succeeded. The source paper does define a recovery criterion, namely that the video has resumed streaming from the last checkpoint saved before the failure, and measures the time to reach it, but this is an experimental measurement performed by the authors rather than a check the running system performs. This review therefore records the absence of in-loop verification as structurally certain from the described architecture rather than as a statement the paper makes about itself; an earlier version of this review described it as explicitly stated, which the source text does not support. [16] scores success on the generated artifact rather than on its effect: its two headline metrics record whether the produced Ansible playbook passes unit tests and what proportion of its individual tasks execute without error, so a playbook that runs cleanly without resolving the underlying fault would score well, and no separate check that the targeted failure cleared is reported. Because these seven definitions measure different things at different points in the pipeline, and in one case measure a different object entirely, the success rates attached to each are not directly comparable to one another, though two of the seven, [4] and [18], tie verification to an automatic reversal mechanism. Figure 4 places the six criteria against the moment of execution.

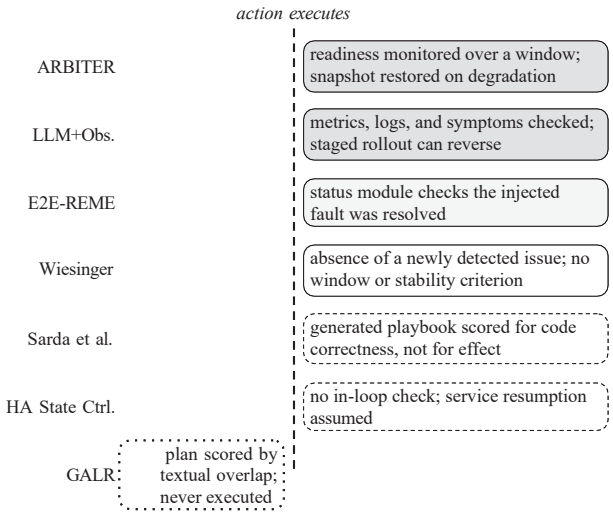

Fig. 4. What each core comparator treats as evidence of recovery, positioned relative to the moment an action executes. Darker boxes tie the check to an automatic reversal mechanism; dashed boxes denote systems that report no check on whether the targeted failure cleared, whether because none is performed or because success is scored on the generated artifact instead; the dotted box left of the line denotes a system whose plan is scored without ever being executed. The seven criteria measure different properties, and in one case a different object entirely, which is why the success rates attached to them are not mutually comparable.

C. Silences

Rollback is reported as a distinct mechanism in two core systems. [4] persists a pre-action snapshot and, for the subset of its action types that carry an implemented rollback monitor, observes pod readiness during the window and restores that snapshot if readiness deteriorates. Two qualifications matter. The monitor is implemented for two of its action types rather than all of them, and the source paper states directly that the remaining types have no rollback monitor. More consequentially, no reported experiment describes the monitor firing: the paper reports no case in which readiness degraded and a snapshot was restored, so the mechanism is specified and built but not exercised in the reported evaluation. A figure of five successes per flavour that appears in the same paper refers to the selection of a canary-rollback remediation action in deployment-regression runs, which is an action choice rather than an instance of the safety mechanism reverting an action this review would count as unsuccessful. [18] similarly ties rollback to its outcome verification, applying changes through a progressive, staged rollout and automatically halting or reversing the rollout if anomalies persist or worsen between stages, though the source paper reports this behavior descriptively rather than as a measured rate over a population of remediation attempts. Three core systems, [8], [15], and [19], report no rollback mechanism; this review makes that statement because each source paper's described architecture leaves no path for reversing an executed action once taken, treated here as structurally certain rather than as an inference from silence. [3] does not report a rollback mechanism, and here this review records the absence as not reported rather than as confirmed missing, though the source paper's own threats-to-validity section requests exactly this kind of runtime safeguard as future work, which indicates the paper itself treats the mechanism as absent at the time of writing without this review needing to infer that conclusion independently. Across the entire core comparator set, no paper reports a rollback rate computed over a population of remediation attempts; where rollback is measured or described at all, it is described qualitatively or measured as correctness on a small enumerated set of cases, and this review treats the absence of a population-level rollback rate as a structural silence, since measuring one would require executing enough remediation attempts, including unsuccessful ones, for a rate to be meaningful, a requirement none of the reported evaluation designs in this corpus appears built to satisfy.

Whether safety layers hold under conditions specifically constructed to defeat them, as opposed to conditions encountered during ordinary evaluation, is tested in only one core system. [4] is the sole paper that adversarially tests its own validator, reporting eleven of eleven adversarial inputs correctly rejected and seventy-seven of seventy-eight legitimate outputs correctly allowed, with one of seventy-eight correctly denied on budget grounds. [18] reports a different and weaker form

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 09 , September - 2026

of evidence: an ablation that removes safety validation entirely and states that doing so does not significantly change detection metrics but increases the risk of incorrect remediation actions, an effect the source paper itself states was not captured in its accuracy tables. This is evidence that the effect of removing the safety layer was assessed, not evidence that the layer was adversarially tested; [4] remains the only system in the corpus tested against inputs constructed to defeat its safety mechanism specifically. [3] shapes safety only at training time, so there is no runtime mechanism for this review to describe as tested or untested; the question of adversarial robustness does not apply to it in the same sense it applies to a runtime gate.

Whether a remediation counts as recovered is, across this corpus, answered several different ways rather than left undefined outright; every core system that reports verifying an outcome states some criterion for doing so, however minimal. The absence identified in this section is not that recovery goes undefined, but that no two definitions agree, that only two of the seven are tied to an automatic reversal mechanism, and that none is evaluated for sensitivity to the length of its stability window. A survey spanning 183 AIOps papers finds the same absence of a standardized verification category at the level of the broader field [1], and a Kubernetes-native guardrail tool restricted to detection and advisory output, outside the core comparator set, still produces a structured, auditable finding record without ever executing a remediation itself [9], a design choice available to the field that none of the seven core comparators adopts in combination with autonomous execution. The consequence carried forward to Sections X and XI is that verification and rollback are the dimension along which the corpus is least comparable to itself, more so than the accuracy figures examined in Section VIII, because accuracy figures at least share a numeric scale, while these seven verification definitions do not share a common unit of measurement at all.

D. A Common Form for Recovery Criteria

The six criteria above cannot be ranked against one another because they do not share a unit of measurement. They can, however, be written in a common form, and doing so makes visible which component each one omits. The formalization below is this review's own construction. It is not drawn from, nor attributed to, any paper in the corpus, and it is offered as a way of stating the gap precisely rather than as a standard against which the reviewed systems should be judged, none of which set out to satisfy it.

Let a denote a remediation action executed at time t_0 against a system exhibiting a targeted failure, and let $h(t) \in \{0, 1\}$ be a health predicate taking the value 1 when the targeted failure is not observable at time t. Let δ be the delay after execution at which observation begins and W the length of the observation window. Three successively stronger criteria follow.

The weakest evaluates the predicate once:

$$V_{\text{clear}}(a) = h(t_0 + \delta). \tag{1}$$

A stability criterion instead requires the predicate to hold throughout a window rather than at a single instant:

$$V_{\text{stable}}(a, \delta, W) = \prod_{t \in [t_0+\delta,\ t_0+\delta+W]} h(t). \tag{2}$$

Neither criterion establishes that the action caused the recovery. Let $\varnothing$ denote the null action, that is, the trajectory the same system would have followed had no remediation been applied. Over N incidents, an attributable recovery rate nets out the rate at which the system would have recovered unaided:

$$R_{\text{attr}} = \frac{1}{N}\sum_{i=1}^{N} V_{\text{stable}}(a_i, \delta, W) - \frac{1}{N}\sum_{i=1}^{N} V_{\text{stable}}(\varnothing_i, \delta, W). \tag{3}$$

Placed against this form, the corpus occupies a narrow band. Every core system that verifies an outcome at all reports a criterion of the shape of (1): [15] evaluates the predicate once with δ unstated and no window, [3] checks resolution of the injected fault at a point, and [18] evaluates three predicates rather than one but reports no window over which they must hold. [4] is the only core system whose criterion is evaluated over an observation window rather than at a single instant, and so the only one instantiating (2), though the sensitivity of its results to the choice of W is not reported. [8] evaluates no predicate inside its control loop, and [19] defines no h at all, since no action executes and therefore no post-action trajectory exists. [16] substitutes a different object for h altogether, scoring the generated artifact's syntactic and task-level correctness rather than any property of the system after the action lands.

The second term of (3) is estimated nowhere in the corpus for a fault class where it could be non-zero. One paper does include a no-controller comparison arm [4], but it does so for a bad-image deployment regression, a fault the platform's own mechanisms cannot resolve, so the baseline it measures is structurally zero and carries no information about attribution. Every other reviewed system reports the first term of (3) alone. This matters because Kubernetes restarts failed containers by default under an increasing backoff, so for crash-loop and transient resource faults the second term is neither zero nor known, and a reported recovery figure for those classes does not separate the effect of the remediation from the recovery behaviour of the platform beneath it. This is the precise sense in which Section XI treats recovery verification as unresolved: the field reports quantities of the form (1) and interprets them as though they were quantities of the form (3).

VIII. Evaluation Environments, Workloads, and Metrics

A. Convergence

A small number of demo microservice applications dominate as evaluation workloads across the corpus. Online Boutique is used by eight papers [2]–[4], [11], [15], [18], [20], [21], and Sock Shop by six [5], [6], [11], [16], [19], [20], with [11] and [20] evaluating against all three of the most common demo applications. Train Ticket, the largest of the

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 09 , September - 2026

three, appears in [3], [11], [18], [20]. Two further demo applications appear once each: [16] evaluates additionally against Robot Shop, and [4] uses DeathStarBench Social Network as its primary live workload, treating Online Boutique only as a portability check rather than as the source of its comparative claims. Three papers in the corpus evaluate against a workload outside these demo applications: [3] against industrial production traffic, [8] against a purpose-built stateful video-on-demand application chosen so that recovery could be observed directly as resumed streaming, and, outside the core comparator set, [17] against internal traffic from a production deployment spanning more than thirty teams.

Fault injection tooling shows the strongest single convergence identified anywhere in this review. Eleven corpus papers report an explicit injection method: Chaos Mesh [2], [3], [14], stress-ng combined with traffic control [11], [20], Envoy combined with K6 [21], docker-stress [6], and bespoke injection mechanisms specific to the system under evaluation [7], [8], [13], [15]. No paper in the corpus derives its evaluation from a naturally occurring incident rather than an injected one.

Telemetry collection converges on Prometheus or a Prometheus-compatible store. Five papers report Prometheus directly as their metrics substrate [5], [6], [11], [13], [20], and two further papers build on OpenTelemetry [4], [18], one of which stores the resulting metrics in a Prometheus-compatible time-series database [18]; [21] reports K6-derived metrics. The convergence is therefore on a small number of standard substrates rather than on Prometheus alone, and one paper argues explicitly for the shift, contending that OpenTelemetry's vendor-neutral schema and resource-attribute enrichment change the design space for controllers that earlier required benchmark-specific instrumentation [4]. No paper in the corpus argues for a substrate outside this set.

Where diagnosis quality is scored quantitatively, the corpus converges on top-k ranking metrics. [11] and [20] report AC@k and Avg@k, and [19] reports Top-k accuracy and mean reciprocal rank. [2] uses the same top-k shape to separate diagnosis accuracy from recovery-action validity, treating them as two distinct scores rather than a single combined figure. No remediation-side counterpart to these ranking metrics, such as a standardized action-validity score, appears anywhere in the corpus.

Benchmark design for agentic operations work is itself a subject of critique within the framing literature, independent of this corpus's own core comparators. A benchmark for training and evaluating AI site reliability engineering agents critiques two named prior benchmarks as too simple, and reports that its own mitigation oracle checks actual system state specifically to avoid rewarding an agent for suppressing an alert rather than resolving the underlying condition [22]. Neither of the two benchmarks this source critiques is itself part of this review's corpus, but the critique is relevant evidence that benchmark design for this class of system is recognized within the field as a nontrivial problem in its own right, a point this review returns to in Section XI.

B. Divergence

Reported LLM action-selection accuracy differs by roughly forty to sixty percentage points across three studies answering what is, on its face, the same question: whether an LLM can select a valid Kubernetes recovery action. One paper reports 96.67% overall accuracy using Mistral-7B across fifteen pod-level Kubernetes faults, with iterative retry permitted, at N=15 with four repetitions [15]. A second reports recovery-action validity for top LLMs ranging from 36.8% to 60.3% across 302 audited Kubernetes incidents on Online Boutique, with recovery validity falling to 0.00 for one fault category despite near-perfect diagnosis accuracy on the same cases [2]. A third reports that no standalone general-purpose LLM it evaluated exceeded 50% accuracy on its benchmark, a bound that held even at the benchmark's easiest of three difficulty tiers [3]. These three figures are not directly reconcilable from the information available in this corpus. The sample sizes differ by roughly a factor of twenty, the allowance for retry differs across the three evaluation designs, and each paper defines a correct or valid action differently, so no single confound can be isolated as the explanation without a harmonized evaluation protocol that does not exist in this literature. This review reports the three figures alongside one another and treats the discrepancy as an open question rather than adjudicating which figure is more representative of the field's actual capability.

This divergence compounds a divergence already established in Section VII: because seven core papers apply seven largely incompatible definitions of a verified recovery, a reported accuracy or MTTR figure encodes not only how well a system selected an action but also how permissively that paper defined success. A headline MTTR number and a headline accuracy number are therefore not comparable across papers even before accounting for differences in sample size, workload, or fault taxonomy, since the two papers being compared may not be measuring the same event at all. This review does not construct a ranked comparison of core system performance for this reason; doing so would imply a comparability the underlying evaluation designs do not support.

C. Silences

No LLM-bearing paper in the corpus decomposes its reported MTTR or recovery-time figure into model inference latency and cluster convergence time. [15] reports an MTTR of 345.27 seconds at maximum retry without attributing that duration to any specific stage of the pipeline, despite the paper's own premise being the use of low-parameter models chosen specifically for their footprint. The vocabulary needed for this decomposition already exists in the corpus: [8] separates recovery time into reaction time, recovery time, repair time, and outage time, but no LLM-bearing paper applies an equivalent decomposition to model inference specifically. This review treats the absence as a structural silence, since the practical effect of reporting such a decomposition would be to expose an unfavorable latency ratio for larger models, and no evaluation protocol in the corpus requires it.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 09 , September - 2026

TABLE IV
REPRODUCIBILITY STATUS OF CORE COMPARATORS. NR = NOT REPORTED.

Ref	ystem	Code	Data	Reproduced
[3]	2E-REME	Yes	Yes	No
[4]	RBITER	Yes	Yes	No
[8]	HA State Controller	NR	NR	No
[15]	Wiesinger et al.	NR	Yes	No
[16]	Sarda et al.	NR	NR	No
[18]	LLM plus Observability	NR	NR	No
[19]	GALR	NR	NR	No

A related silence concerns statistical rigor around reported results. [18] presents accuracy figures with no error bars and does not report a repository. [15] reports four repetitions at N=15. By contrast, several papers outside the LLM-bearing core subset report repeated trials as standard practice: ten runs with paired t-tests in [13], ten repetitions in [7], five repetitions in [20], and ten repetitions in [8]. This pattern, in which statistical rigor tracks closely with whether a system involves an LLM component, is treated here as a reporting silence rather than a structural one, since the underlying repeated-trial data may exist for the LLM-bearing systems without having been reported in the source text this review examined.

IX. REPRODUCIBILITY

A. Convergence

Across the full corpus of 23 papers, twelve report a publicly released code repository, dataset, or benchmark artifact [3]–[5], [7], [9], [11]–[13], [15], [20]–[22]. This release rate is not distributed evenly across paper type. Among papers classified as benchmarks, datasets, or methods, release is close to standard practice. Among the seven core remediation systems specifically, three release some artifact: [3] and [4] release full source code and full evaluation artifacts, while [15] releases its evaluation dataset publicly without reporting the release status of its implementation code. Table IV reports code release, data release, and independent reproduction status for each of the seven core comparators.

No core comparator in this corpus has been reproduced by any party outside its original author group. This applies even to [3] and [4], both of which release complete code and evaluation artifacts; for these two systems, the absence of independent reproduction is consistent with their recent publication and is not treated by this review as evidence against their reported results, only as an outstanding step the field has not yet taken. [16] is structurally irreproducible regardless of intent, since its evaluation runs on partner infrastructure not available outside the original study, a constraint the source paper states directly rather than one this review infers.

B. Divergence

The corpus does not divide into two comparably sized camps on this dimension. Reproducibility practice instead tracks paper type closely, with benchmarks, datasets, and methods releasing artifacts as a near-default and remediation systems specifically withholding them as a near-default, so this review does not treat reproducibility as a genuinely contested axis in the way Sections V through VIII treat autonomy, safety enforcement, and reported accuracy.

C. Silences

For five of the seven core comparators, code availability is recorded as not reported rather than as confirmed absent [8], [15], [16], [18], [19]. Data availability follows the same not-reported status for four of these five, [8], [16], [18], [19], while the fifth, [15], reports its evaluation dataset as publicly released even though its implementation code is not reported. This review cannot determine from the source text whether the remaining authors chose not to release an artifact or simply did not state a release status in the sections available for extraction, and treats this distinction as a reporting silence rather than a structural one. [18] was earlier treated in this review as a confirmed absence of both code and data; on direct review of the source text, no explicit statement of code or data availability, positive or negative, could be located, and this review now records both as not reported rather than confirmed absent, consistent with the treatment of the other four systems.

Whether the released artifacts outside the core comparator table have themselves been used by any subsequent, independent study is not addressed by any paper in this corpus, since a citing or reusing study would itself need to appear in the corpus for this review to identify it, and no such downstream reuse was identified during corpus construction. This review records that absence as a limit of the corpus's scope rather than as a finding about the artifacts' actual reuse, a distinction restated in Section XII.

X. COMPARATIVE SYNTHESIS

This review groups the seven core comparators into four architectural families, distinguished by three successive questions: whether the system executes an action against the cluster at all, whether an LLM selects or generates that action, and whether the action is validated at runtime before it reaches the cluster. Figure 5 shows the resulting partition. *F1, Guarded Agentic Remediation*, covers systems in which a model proposes an action that a separate runtime component validates before execution [4], [18]. *F2, Open Generative Remediation*, covers systems that generate an unbounded artifact and apply it without a comparable runtime check [3], [15]. *F3, Advisory Terminal*, covers systems that stop before executing [16], [19]. *F4, Deterministic Controller*, covers the single non-LLM system, whose action space is bounded by construction [8].

Table V places all seven core comparators against eight dimensions drawn directly from Sections IV through IX. A check mark indicates the source paper reports the capability present; a cross indicates the source paper states the capability is absent or the architecture structurally excludes it; Partial indicates a weak, soft, or qualitative form of the capability as described in the source paper; NR indicates the dimension is not reported in the extraction.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 09 , September - 2026

TABLE V
LANDSCAPE OF CORE COMPARATORS ACROSS EXTRACTION DIMENSIONS. ✓ = REPORTED PRESENT. × = REPORTED OR STRUCTURALLY ABSENT. PARTIAL = WEAK, SOFT, OR QUALITATIVE FORM. NR = NOT REPORTED. ACTION SPACE AND SCOPE REPORT DESCRIPTIVE VALUES TAKEN DIRECTLY FROM THE EXTRACTION RATHER THAN THIS FOUR-SYMBOL LEGEND.

Ref	System	Family	Autonomy	Action Space	Safety Gate	Verifies	Rollback	Scope
[3]	E2E-REME	F2	L5	Unbounded	Partial	✓	NR	Broad
[4]	ARBITER	F1	L3	Bounded (4)	✓	✓	✓	Narrow
[8]	HA State Controller	F4	L4	Bounded (1)	✓	×	×	Narrow
[15]	Wiesinger et al.	F2	L5	Unbounded	Partial	Partial	×	Broad
[16]	Sarda et al.	F3	L2	Unbounded	NR	×	×	NR
[18]	LLM plus Observability	F1	L4	Unbounded	✓	✓	✓	Broad
[19]	GALR	F3	L2	Bounded (templates)	NR	×	×	Moderate

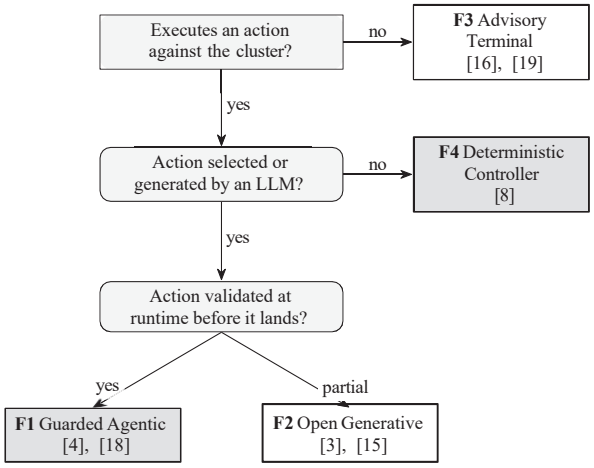

Fig. 5. The four architectural families used in the Family column of Table V, derived from three successive distinctions rather than assigned by inspection. Shaded families apply a constraint that is either deterministic by construction or enforced at runtime on every action.

Arranging all seven systems against the same set of columns surfaces a pattern that the dimension-by-dimension sections, each examined in isolation, do not make visible: the four architectural families identified in this review correspond to consistent bundles of table values rather than to independent design choices made along each dimension separately. The two Open Generative Remediation systems, [3] and [15], share L5 autonomy, an unbounded action space, and a Safety Gate value of Partial, despite otherwise differing in fault-scope breadth and in how that partial gate is constructed, a training-time penalty for one and reliance on the platform's own admission validation for the other. The two Advisory Terminal systems, [16] and [19], share L2 autonomy and a cross on both Verifies and Rollback, though for different underlying reasons: [19] never executes its generated plan at all, leaving no outcome to verify or revert, while [16] executed its generated playbooks manually during evaluation and reports no post-action check on the outcome and no mechanism to reverse it if the action fails. The Advisory Terminal label fits both for what they demonstrate, though [16] specifies an automated execution path its evaluation does not exercise; this review groups on demonstrated behavior and flags the label as a partial description of [16]'s actual behavior. This clustering by family across independently extracted dimensions is evidence that the four families identified in Section X's underlying synthesis describe genuine architectural commitments rather than an artifact of how this review chose to group the corpus.

A cross-dimension pattern connects autonomy level to the strength of a system's pre-execution safety gate. Both L5 systems carry a Partial safety gate, the weakest non-NR value in that column. Every system below L5 that executes at all, the two L4 systems [8], [18] and the single L3 system [4], carries a full check mark, despite implementing that gate through markedly different mechanisms: a deterministic label flip with no separate validation layer; a four-stage sequence of policy enforcement, simulated impact analysis, rate limiting, and dry-run validation; and a schema, allowlist, budget, and quota validator behind an approval path. On this dimension the corpus divides cleanly at the boundary between L5 and everything below it: the two systems that reserve no constraint on the executed action are also the two whose safety mechanism this review could not code as fully present.

A second candidate cross-dimension pattern would connect the breadth of a system's failure scope, established in Section IV, to the strength of its safety gate. The two narrowest-scope systems in the table, [4] and [8], do carry a full check mark on Safety Gate, and the two Open Generative Remediation systems, [3] and [15], both broad in scope, carry only Partial. [18], however, is also broad in scope and nonetheless carries a full check mark, which breaks this pattern rather than confirming it. This review therefore does not report scope breadth as associated with safety-gate strength across the full set of seven systems; the partial association visible among the remaining four systems should not be read as a corpus-wide finding.

XI. CONSENSUS AND OPEN CHALLENGES

A. What the Field Agrees On

The corpus exhibits substantial and largely unremarked agreement on evaluation infrastructure. Two demo microservice applications account for most workloads across the reviewed papers, with Online Boutique used by eight [2]–[4], [11], [15], [18], [20], [21] and Sock Shop by six [5], [6], [11], [16], [19], [20]. Eleven papers report an explicit fault injection method, and none derives its evaluation from a naturally occurring incident [2], [3], [6]–[8], [11], [13]–[15], [20], [21]. Five papers report Prometheus as their telemetry substrate

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 09 , September - 2026

[5], [6], [11], [13], [20] and two build on OpenTelemetry [4], [18], with no paper in the corpus reaching outside this small set of standard substrates. This agreement is real and it is infrastructural: the field has converged on how to stage a failure, not on how to judge a repair.

Agreement on fault taxonomy follows the same pattern. Resource exhaustion combined with network perturbation recurs as the canonical evaluation fault set across five papers [3], [11], [14], [15], [20], and six papers inject exactly one fault per evaluation case throughout [2], [3], [11], [14], [20], [21]. Three papers evaluate simultaneous faults [7], [8], [15], but only one of the three injects faults that interact with one another [7], and that system is neither Kubernetes-native nor LLM-bearing.

Two further agreements are tacit rather than stated, and are correspondingly more informative about the field's assumptions. Four of the six LLM-bearing core systems restrict the model to a role short of unconstrained execution [4], [16], [18], [19], indicating that the field already treats an unconstrained model executor as unacceptable in practice without having argued the position in print. Separately, every core system that defines success at all defines it as returning the cluster to a pre-existing desired state [3], [4], [8], [15], [16], [18], and no paper in the corpus considers that the desired state itself might be incorrect or that the appropriate response to a failure might be to adapt the target rather than restore it.

B. Open Challenges

Seven gaps follow from the convergences, divergences, and silences identified across Sections IV through X. They are stated below with the evidence supporting each and the reason each is difficult to close.

1) **Recovery verification has no shared definition and is not itself evaluated.** Seven core systems apply seven largely incompatible criteria for a verified recovery, ranging from a dedicated verification module that checks whether the specific injected fault was resolved [3], to a post-action monitor tied to snapshot restore [4], to the absence of a newly detected issue with no stated time window [15], to textual overlap against a reference plan that is never executed [19], to a three-criterion outcome check tied to a staged rollout mechanism [18], to code-level correctness of the generated artifact rather than any check on its effect [16], to no post-action check at all [8]. This is the best-attested gap in the review, because it does not rest on this corpus alone: a survey spanning 183 papers reports that its own taxonomy contains no verification subtask whatsoever [1], making the absence independently confirmed by a third party with no stake in these particular systems. Closing it is hard because genuine verification requires a counterfactual that a live cluster cannot supply, namely whether the system would have recovered without intervention. Symptom clearance is cheap to measure and insufficient as evidence, and a defensible alternative requires a stability criterion, an observation window, and a null model, none of which the field has standardized. Section VII states this gap in a common form and shows that the corpus reports quantities of the shape of (1) while interpreting them as though they were quantities of the shape of (3).
2) **Runtime gating and training-time safety shaping have never been compared.** One camp enforces constraints on every action at execution time, [4] and [18], while another shapes behavior through a training-time penalty [3] or adds no gate of its own at all, relying on the platform to reject malformed output [15]. The gap is acknowledged from within the second camp rather than asserted from outside it: the threats to validity section of [3] requests precisely the runtime safeguards its own design omits, which converts this from an inference drawn across papers into a position one of the papers states about itself. Closing it requires a single substrate capable of running both regimes with matched action spaces, and because the two approaches do not share an action representation, a fair comparison would require reimplementing one inside the other's formalism before any measurement could begin.
3) **The diagnosis-to-action gap is quantified but no gated system has been evaluated against it.** Diagnosis accuracy reaching 91 to 99.7 percent coexists with recovery-action validity of 36.8 to 60.3 percent across 302 audited Kubernetes incidents, with validity falling to 0.00 for one fault category despite near-perfect diagnosis on the same cases [2]. The gated system in the corpus reports results on ten covered contexts and one uncovered context [4], which is a demonstration rather than a measurement at the scale the diagnosis-to-action gap has been established. Closing it requires a labelled action-validity corpus coupled to a live executor, which means the validity labels must survive execution rather than only inspection, a substantially harder artifact to construct than either component alone.
4) **The cost of an incorrect remediation is unmeasured across the entire corpus.** Every core system reports whether its repair succeeded, and no core system reports what follows when it does not: no blast radius, no damage metric, and no time to detect that a remediation was itself wrong. The nearest approach quantifies action invalidity but stops before execution consequences [2]. This absence is structural rather than incidental, and the reason is visible in the corpus itself: measuring the cost of a wrong remediation requires deliberately executing known-bad actions against a running system, which is the exact outcome every safety mechanism in this literature exists to prevent [3], [13], [15]. Closing it needs both a disposable environment and an agreed damage metric, and the corpus contains neither.
5) **Model inference latency is excluded from remediation-time accounting.** No LLM-bearing paper in the corpus decomposes its reported recovery time into model inference and cluster convergence. One paper reports a mean time to recovery of 345.27 seconds at maximum retry without attributing that duration to

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 09 , September - 2026

any pipeline stage [15], despite the paper's premise being the use of low-parameter models selected for their footprint. The vocabulary for this decomposition already exists inside the corpus, since one system separates reaction, recovery, repair, and outage time [8], and the corpus demonstrates elsewhere that it can quantify an overhead precisely when it chooses to, reporting 180 percent additional CPU consumption as the price of eliminating recovery time [14]. The difficulty here is not technical. Reporting the decomposition would expose an unfavorable latency ratio for larger models, and no evaluation protocol in this literature requires it.

6) **Reported action-selection accuracy differs by forty to sixty percentage points across comparable settings, and the discrepancy is unexplained.** One study reports 96.67 percent overall accuracy at a sample size of fifteen with four repetitions and retry permitted [15], a second reports 36.8 to 60.3 percent recovery-action validity across 302 audited incidents [2], and a third reports the best general-purpose model evaluated at under 50 percent, even at its benchmark's easiest of three difficulty tiers [3]. All three address whether a model can select a valid Kubernetes recovery action, and this review presents the discrepancy as open rather than adjudicating it, because the three designs differ simultaneously in sample size by roughly a factor of twenty, in retry allowance, in fault taxonomy, and in what each defines as a correct action. No single confound can be isolated without a harmonized protocol, and constructing one requires deciding whose definition of a valid action prevails, which is a substantive commitment rather than a procedural one.
7) **Safety layers are asserted architecturally and rarely tested adversarially.** One core system subjects its own validator to adversarial input, reporting eleven of eleven adversarial inputs correctly rejected and seventy-seven of seventy-eight legitimate outputs correctly allowed [4]. A second reports an ablation that removes safety validation entirely and observes an increased risk of incorrect remediation actions, but this observation is qualitative and is not tested against inputs constructed adversarially to defeat the safety layer specifically [18]. A third shapes safety only at training time and therefore has no runtime mechanism available to test [3]. Adversarial testing of a planner-output validator is hard for a specific reason: it requires generating plausible but harmful remediation plans, which is a generation problem of roughly the same difficulty as the remediation problem the validator exists to guard.

Six of these seven gaps concern what happens at or after the moment an action executes. Only one, the unexplained spread in reported action-selection accuracy, concerns how an action is chosen. The distribution is itself a finding about where this literature has invested its attention, and it holds across a corpus whose members otherwise disagree about autonomy, safety enforcement, and the definition of recovery. A benchmark for agentic operations agents makes a compatible observation from a different direction, reporting that its mitigation oracle inspects actual system state specifically to avoid rewarding an agent for suppressing an alert rather than resolving the condition beneath it [22], and a detection-only guardrail tool demonstrates that structured, auditable finding records are available to systems that decline to execute at all [9].

XII. Threats to Validity

A. Selection Bias and Corpus Size

This review does not follow a systematic search protocol. The corpus was assembled through direct submission, citation chaining, and cross-referencing among retrieved papers, and no fixed query was run against a bibliographic database. Twenty-three papers is a small corpus for claims about a field, and several counts reported throughout Sections IV through X rest on single-digit numbers of papers, which limits how far any convergence claim generalizes beyond the specific systems examined here.

Two categories of known omission compound this. Six papers were submitted for inclusion but were not extracted before the corpus was frozen, and they appear in no table, figure, or claim. Separately, eleven well-known works in this area were identified as relevant during the review but were never collected or extracted. These include two agentic-cloud-operations benchmarks, AIOpsLab and ITBench, both of which are critiqued within the corpus by a benchmark that is included [22]; ADARMA, an earlier full-autonomy system from the same author lineage as one core comparator [16], whose omission directly weakens the autonomy-regression observation in Section V; and a pre-LLM survey of anomaly detection and root cause analysis in service-based cloud applications by Soldani and Brogi, whose inclusion would have strengthened the historical grounding of Section III. Each of these would plausibly have altered specific claims, and none should be treated as having been considered and rejected.

B. Language and Publication Status

Only English-language publications were considered. No search was conducted in any other language, and no assessment has been made of whether relevant work in this area is published outside English-language venues.

Two of the most load-bearing sources in this review are non-peer-reviewed preprints. The study establishing the diagnosis-to-action gap at scale [2] and the only system in the corpus with an adversarially tested validator and a specified but unexercised rollback mechanism [4] are both arXiv preprints. Several claims in Sections VII, VIII, and XI depend substantially on these two sources, and those claims inherit whatever uncertainty attaches to results that have not completed peer review.

C. Reliance on Author-Reported Results

Every quantitative figure in this review is reported by the paper that produced it. No result was independently verified, no system was executed, and no reported metric was

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 09 , September - 2026

recomputed. Section IX establishes that no core comparator in this corpus has been reproduced by any party outside its original author group, which means the accuracy, recovery-time, and safety-mechanism figures compared throughout this review rest entirely on self-reported evidence. Section VIII notes further that statistical rigor varies considerably across the corpus, with some papers reporting repeated trials and confidence intervals and others reporting single figures without error bars, so the reported values are not uniformly reliable even on their own terms.

D. Single-Annotator Extraction

All extraction was performed by one annotator. No second annotator independently coded any paper, no inter-rater agreement was computed, and no adjudication process existed for ambiguous cases. Several coding decisions reported in this review are genuinely contestable and were resolved by a single judgment. Whether a bounded retry limit constitutes a constraint gate determines whether one system is classified at L4 or L5 [3]. Whether the Kubernetes API server's own admission validation counts as a safety mechanism the system provides, as opposed to a property of the platform it runs on, determines whether another system's safety gate is coded Partial or absent [15]. Whether a system whose architecture specifies automated execution but whose evaluation executed every action manually should be coded on its design or on its demonstration determines the placement of a third [16]; this review codes demonstrated behavior throughout, which places it at L2. The same design-versus-demonstration question determines the placement of [4] at L3 rather than L4, since its architecture provides an auto-allow path above a confidence threshold while every reported experiment ran through approval-required execution. A reviewer who preferred to code architectural capability rather than evaluated configuration would move both systems up one level, which would restore the claim, made in an earlier version of this review, that no core system occupies L3. Each of these placements is defensible and each is arguable, and a second annotator might reasonably have coded them differently.

The extraction dimensions themselves originate in part from the architecture of an in-progress remediation prototype developed by the authors, described briefly in Section XIII. Dimensions chosen with a particular architecture in view may direct attention toward capabilities that architecture possesses and away from capabilities it does not, and this review cannot rule out that its emphasis on verification and rollback reflects that origin rather than the field's own priorities. Two considerations weigh against this reading without eliminating it. The verification gap is independently attested by a third-party survey of 183 papers with no connection to this work [1], and the central observation of Section V, that restraint correlates with production exposure, runs against the interest of an autonomous remediation system rather than supporting it.

E. Absence Versus Silence

This review distinguishes throughout between a capability a paper states or structurally excludes and a capability a paper does not report, and records the latter as not reported rather than as absent. This distinction is applied as consistently as the source material permits, but it cannot be applied perfectly. For five core comparators, code availability is recorded as not reported, and this review cannot determine whether those authors declined to release their implementation or simply did not state a release status in the text available for extraction [8], [15], [16], [18], [19]. Data availability carries the same ambiguity for four of these five, [8], [16], [18], [19], while the fifth, [15], releases its evaluation dataset even though its implementation code is not reported. Similar ambiguity affects rollback for one system [3].

Where this review characterizes an absence as structural rather than unreported, that judgment rests on the described architecture leaving no path for the capability in question, not on the capability going unmentioned. Readers who disagree with a specific structural judgment should treat the corresponding claim as weaker, since the difference between the two categories is a matter of interpretation applied to source text rather than a fact recoverable from the extraction alone.

F. Source Verification and Its Consequences

Bibliographic metadata initially flagged as missing for several entries in the reference list was resolved against publisher listings and preprint servers. A subsequent pass re-checked the coded dimensions of the core comparators against their full source texts, and this pass changed substantive claims rather than only citation detail, so its results are reported here in full.

All seven core comparators were re-read against full source text. For [4], the re-read moved the system from L4 to L3: the source states that all reported experiments used approval-required execution, with plans created pending approval and the experiment harness rather than a human granting approval for repeatability, and it names autonomous execution modes as available but not enabled. An earlier version of this review reported that no core system occupied L3; that claim does not survive the re-read and has been withdrawn. For [18], the re-read established a four-stage pre-execution safety gate, a three-criterion outcome check, and a rollback path tied to staged rollout, all of which an earlier extraction had understated, and established that its ablation assesses the effect of removing safety validation qualitatively rather than leaving the safety layer unmeasured. For [15], it established that the system adds no pre-execution check of its own, that its fifteen faults span two categories rather than constituting fifteen distinct types, and that its released artifact is an evaluation dataset rather than implementation code. For [16], it established that the architecture specifies automated execution but the reported evaluation executed every playbook manually, correcting both an original claim that execution rested with the operator and a subsequent over-correction in the opposite direction. For [19], it established a named model backbone, an action space bounded by predefined command templates, and that

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 09 , September - 2026

parameter-level safety constraints are proposed as future work rather than implemented. For [3], it confirmed the reported accuracy bound and added the qualification that the bound holds at the benchmark's easiest difficulty tier.

[8] was also re-read against full source text, and the re-read corrected two claims. Its evaluation includes a dedicated experiment failing up to five active pods simultaneously, which contradicts an earlier statement in this review that only one non-Kubernetes-native paper evaluated concurrent failure; the concurrency silence is real but holds for the LLM-bearing subset rather than for the corpus. Separately, this review had described its absence of post-action verification as explicitly stated by the source, which the text does not support: the absence is certain from the described control loop, but the paper makes no statement about it, and it does define and measure a recovery criterion in its evaluation. Verification against full source text was not performed for the sixteen framing citations, with the exception of two whose full texts were available during the verification pass; the single-annotator limitation described above therefore applies with full force to the remaining framing entries.

That several substantive codings changed on contact with full source text, after those codings had already survived one round of metadata verification, is itself a finding about this review's method. Extraction performed at summary depth was sufficient to identify which systems were relevant and roughly what they did, but was not reliable at the resolution the comparative claims in Sections V through XI require. A reader should treat any dimension in this review that has not been re-derived from full source text as carrying materially more uncertainty than the dimensions that have.

G. Candidate Gaps Considered and Rejected

Seven further gaps were considered during synthesis and dropped for lack of evidence. They are recorded here so that their absence from Section XI is visible as a decision rather than an oversight.

1) **Multi-cluster and federated remediation.** No paper in the corpus targets this or names it as future work, and the nearest relevant system is neither Kubernetes-native nor LLM-bearing [7]. Asserting a gap here would mean inferring a field-level absence from an absence of stated interest.
2) **Remediation of attack-induced failures.** Only one paper scopes this, and does so explicitly as future work in a cyber-physical security context [13]. A single paper's stated future work does not establish a corpus-level gap, and the topic drifts into a different problem domain.
3) **Stateful workload remediation.** Directly contradicted by a core comparator that addresses exactly this failure class [8]. Narrowing the claim to the LLM-bearing subset would require asserting absence from silence across five papers.
4) **Explainability and audit quality of model decisions.** Structured audit records appear in the corpus [4], [9], [13], and although no paper measures explanation quality, this review could not separate genuine field silence from limits in the extraction, and the concern substantially overlaps the adversarial-testing gap already stated.
5) **Absence of production deployment.** Refuted by the corpus. One core system reports results on industrial production traffic [3], and one framing source is production-deployed across more than thirty teams [17].
6) **Unquantified monitoring overhead.** Refuted by the corpus. Overhead figures are reported by several papers [5], [6], [13], [14], so no silence exists to report.
7) **Cost-benefit analysis of autonomy level.** Tempting given the divide documented in Section V, but the restrained camp justifies its position normatively rather than through measurement, so there is no existing yardstick whose absence could be reported without this review inventing it. The measurable portion of this concern is absorbed into the gap concerning the unmeasured cost of an incorrect remediation.

H. Currency

This is a fast-moving research area. The corpus spans publication years 2022 through 2026 with the majority concentrated in the final three years, and several core comparators were published within months of this review being assembled. Claims of the form that no system does something are claims about the corpus at the time of freezing and carry a short shelf life. The absence of independent reproduction for the two core systems that released complete artifacts [3], [4] is particularly likely to be a function of recency rather than of any property of those systems, and should not be read as a durable finding.

XIII. Conclusion

Research on automated fault remediation in Kubernetes has converged on a common evaluation substrate. Demo microservice applications, injected resource and network faults, and Prometheus or OpenTelemetry telemetry recur across the corpus with sufficient consistency that reported results appear, on their surface, to be measuring the same thing [11]. They are not. Seven core systems apply seven largely incompatible definitions of a verified recovery, rollback is measured as a population-level capability in none of them, and reported action-selection accuracy on comparable Kubernetes settings ranges from no standalone model exceeding 50 percent [3] to 96.67 percent [15] without the discrepancy having been explained. A study of 302 audited incidents establishes that near-perfect diagnosis coexists with recovery-action validity as low as zero for some fault categories [2], which locates the field's central difficulty after the diagnosis rather than within it.

The consequence is that shared infrastructure has produced the appearance of comparability without its substance. A survey of 183 papers finds no verification subtask anywhere in its taxonomy [1], confirming that this absence is a property of the field rather than of any particular system, and the one core

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 09 , September - 2026

comparator that ties post-action monitoring to automatic snapshot restore while also subjecting its validator to adversarial testing [4] remains an exception rather than a template. Six of the seven open gaps identified in this review concern what happens at or after the moment an action executes. Further increases in autonomy will not be meaningfully assessable until post-action evaluation is standardized, because a higher autonomy level measured against an undefined success criterion produces a larger number rather than a better system.

Future work following from this review divides into two efforts. The first is definitional and collective: a shared specification of what constitutes a verified recovery, including a stability criterion, an observation window with reported sensitivity, and a null model against which spontaneous recovery can be distinguished from remediated recovery, none of which the literature currently provides. The common form given in Section VII is offered as one starting point for that specification rather than as a finished standard. The second is empirical: a harmonized protocol under which runtime action gating and training-time safety shaping can be compared on one substrate with matched action spaces, and under which the cost of an incorrect remediation can be measured rather than prevented by construction. SAGE-K8s, an in-progress Kubernetes operator prototype developed by the authors, treats snapshot capture, bounded action validation, post-action verification, and automatic rollback as first-class and separately measurable stages of the remediation loop, and is intended to serve as one vehicle for the second of these efforts once its evaluation is complete.

REFERENCES

[1] L. Zhang, T. Jia, M. Jia, Y. Wu, A. Liu, Y. Yang, Z. Wu, X. Hu, P. S. Yu and Y. Li, "A Survey of AIOps in the Era of Large Language Models," *ACM Computing Surveys*, 2025. doi: 10.1145/3746635.

[2] J. Qi, Z. Luan, H. Zhang, S. Huang, C. J. Fung, Y. Tong, H. Yang, and D. Qian, "Can LLMs Really Recover Microservice Failures? A Recovery-Aware Evaluation of Diagnosis-to-Action Reasoning," arXiv preprint arXiv:2607.04623, Jul. 2026.

[3] L. Zhang, Y. Zhai, T. Jia, M. He, C. Duan, Z. Liu, B. Ding, and Y. Li, "E2E-REME: Towards End-to-End Microservices Auto-Remediation via Experience-Simulation Reinforcement Fine-Tuning," in *FSE Companion '26*, Montreal, QC, Canada, 2026. doi: 10.1145/3803437.3805206.

[4] P. Habibi and A. Leon-Garcia, "ARBITER: Guarded Agentic Control for SLO-Oriented Kubernetes Remediation," arXiv preprint arXiv:2607.19182, Jul. 2026.

[5] J. Kosinska and M. Tobiasz, "Detection of Cluster Anomalies With ML Techniques," *IEEE Access*, vol. 10, pp. 110742–110751, 2022.

[6] J. Kosinska and K. Zielinski, "Experimental Evaluation of Rule-Based Autonomic Computing Management Framework for Cloud-Native Applications," *IEEE Transactions on Services Computing*, vol. 16, no. 2, pp. 1172–1183, 2023.

[7] A. Brogi, J. Carrasco, F. Duran, E. Pimentel, and J. Soldani, "Self-Healing Trans-Cloud Applications," *Computing*, vol. 104, pp. 809–833, 2022.

[8] L. Abdollahi Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, "A Kubernetes Controller for Managing the Availability of Elastic Microservice Based Stateful Applications," *Journal of Systems and Software*, vol. 175, art. 110924, 2021. Extended journal version of an IEEE QRS 2019 paper.

[9] C. Flynn, B.-Y. Toh, and Z. Li, "KubeRule: A declarative, guardrail-driven runtime management tool for kubernetes automation," *SoftwareX*, vol. 35, art. 102833, 2026.

[10] N. Beuter, A. Drews, and N. Kratzke, "Prompt-Driven and Kubernetes Error Report-Aware Container Orchestration," *Future Internet*, vol. 17, no. 9, art. 416, 2025. doi: 10.3390/fi17090416.

[11] L. Pham, H. Zhang, H. Ha, F. Salim, and X. Zhang, "RCAEval: A Benchmark for Root Cause Analysis of Microservice Systems with Telemetry Data," in *WWW Companion '25*, Sydney, NSW, Australia, 2025. doi: 10.1145/3701716.3715290.

[12] J. Xu, Z. Gao, and Y. Wei, "An Empirical Study on Kubernetes Operator Bugs," in *ISSTA '24*, 2024.

[13] O. Johnphill, A. S. Sadiq, O. Kaiwartya, and M. A. Taheir, "Cross: a cloud-native approach to automated remediation and self-healing in cyber-physical systems," *Cybersecurity*, vol. 9, art. 124, 2026. doi: 10.1186/s42400-026-00549-8.

[14] Y. Bouizem, D. Dib, N. Parlavantzas, and C. Morin, "Integrating request replication into FaaS platforms: an experimental evaluation," *Journal of Cloud Computing*, vol. 12, art. 94, 2023. doi: 10.1186/s13677-023-00457-z.

[15] Y. Wiesinger, M. Engelhardt, and R. Laas, "Leveraging Low-Parameter LLMs for Self-Healing in Kubernetes-Based Container Orchestration," in *SEAMS '26*, Rio de Janeiro, Brazil, 2026, pp. 194–206. doi: 10.1145/3788550.3794882.

[16] K. Sarda, Z. Namrud, M. Litoiu, L. Shwartz, and I. Watts, "Leveraging Large Language Models for the Auto-remediation of Microservice Applications: An Experimental Study," in *FSE Companion '24*, 2024.

[17] Y. Chen, H. Xie, M. Ma, Y. Kang, X. Gao, L. Shi, Y. Cao, X. Gao, H. Fan M. Wen, J. Zeng, S. Ghosh, X. Zhang, C. Zhang, Q. Lin, S. Rajmohan, D. Zhang, and T. Xu, "Automatic Root Cause Analysis via Large Language Models for Cloud Incidents," in *EuroSys '24*, Athens, Greece, 2024, pp. 674–688. doi: 10.1145/3627703.3629553.

[18] C. Wang, T. Yuan, C. Hua, L. Chang, X. Yang, and Z. Qiu, "Integrating Large Language Models with Cloud-Native Observability for Automated Root Cause Analysis and Remediation," in *AISNS '25*, 2025, pp. 327–334. doi: 10.1145/3797161.3797213.

[19] W. Zhang, Z. Yang, F. Peng, L. Zhang, Y. Chen, and R. Chen, "GALR: Graph-Based Root Cause Localization and LLM-Assisted Recovery for Microservice Systems," *Electronics*, vol. 15, no. 1, art. 243, 2026. doi: 10.3390/electronics15010243.

[20] L. Pham, H. Ha, and H. Zhang, "BARO: Robust Root Cause Analysis for Microservices via Multivariate Bayesian Online Change Point Detection," *Proceedings of the ACM on Software Engineering (FSE)*, vol. 1, art. 98, 2024. doi: 10.1145/3660805.

[21] C. M. Aderaldo and N. C. Mendonca, "ResilienceBench-Operator: A Kubernetes Extension for Orchestrating Resilience Experiments on Microservice Applications," in *SBES '25*, Recife, PE, Brazil, 2025.

[22] J. Clark, Y. Su, S. M. Rafid Pial, L. Gniedziejko, and T. Xu, "SREGym: A Live Training Ground for AI SRE Agents with High-Fidelity Failure Drills," in *CAIS '26*, 2026. doi: 10.1145/3786335.3813208.

[23] J.-B. Kim, J.-B. Choi, and E.-S. Jung, "Design and Implementation of an Automated Disaster-Recovery System for a Kubernetes Cluster Using LSTM," *Applied Sciences*, vol. 14, no. 9, art. 3914, 2024. doi: 10.3390/app14093914.