Computer Science Review 63 (2027) 101075

Contents lists available at ScienceDirect

Computer Science Review

journal homepage: www.elsevier.com/locate/cosrev

Large language models for agentic NetOps and AIOps: Architectures, evaluation, and safety

Muhammad Bilal [a,*], Jon Crowcroft [b], Ruizhi Wang [c], Xiaolong Xu [c], Schahram Dustdar [d,e]

[a] School of Computing and Communications, Lancaster University, Lancaster, United Kingdom
[b] Department of Computer Science and Technology, University of Cambridge, Cambridge, United Kingdom
[c] School of Software, Nanjing University of Information Science and Technology, Nanjing, China
[d] Distributed Systems Group, TU Wien, Vienna, Austria
[e] ICREA, Barcelona, Spain

HIGHLIGHTS

- Reviews LLM-based agents for NetOps and AIOps using a structured, evidence-stratified corpus.
- Organises operational autonomy around tools, evidence, gates, budgets, rollout, rollback, and audit.
- Separates directly evaluated practices, transferred operational principles, and author-proposed evaluation templates.
- Shows that the qualitative synthesis indicates a capability–assurance gap for write-capable operational agents.

ARTICLE INFO

Keywords:
Agentic AI
AIOps
Autonomous networks
Closed-loop control
Large language models
LLM agents
Network automation
Network operations
Operational assurance
Reliability
Tool-augmented agents
Root-cause analysis
Closed-loop control
Safety

ABSTRACT

Large language models (LLMs) are increasingly being used in network operations (NetOps) and artificial intelligence for IT operations (AIOps) for tasks ranging from telemetry retrieval and incident diagnosis to configuration planning and bounded remediation. As these systems acquire greater access to operational tools, a central question is whether operational assurance develops commensurately with the authority granted to them. This survey examines that question through a structured, evidence-stratified review of agentic NetOps and AIOps, spanning directly evaluated LLM systems, operational mechanisms, transferred evidence, and normative guidance. We organise the field around autonomy, tool scope, evidence traces, assurance controls, evaluation, security, and governance, and introduce an operational assurance contract that links each autonomy level to permitted tools, required evidence, independent gates, execution budgets, rollout and rollback duties, and audit requirements. The synthesis indicates a capability–assurance gap: evidence is comparatively strong for read-oriented assistance and tool-grounded diagnosis, but becomes substantially less complete as systems approach configuration change, bounded execution, and closed-loop operation. We therefore develop a workflow-level evaluation framework covering evidence quality, tool use, policy and invariant compliance, staged execution, recovery, calibration, cost, and human intervention, together with a threat model for prompt-borne attacks, compromised operational evidence, excessive agency, privilege boundaries, and auditability. The resulting view treats agentic NetOps and AIOps as constrained operational control, in which increasing autonomy requires correspondingly stronger evidence, independently enforced safeguards, and recovery mechanisms.

* Corresponding author.
Email addresses: m.bilal8@lancaster.ac.uk (M. Bilal), jon.crowcroft@cl.cam.ac.uk (J. Crowcroft), rzwdm3ky@163.com (R. Wang), xlxu@nuist.edu.cn (X. Xu), dustdar@dsg.tuwien.ac.at (S. Dustdar).

https://doi.org/10.1016/j.cosrev.2026.101075
Received 5 July 2026; Received in revised form 30 August 2026; Accepted 26 September 2026
1574-0137/© 2026 Elsevier Inc. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

1. Introduction

Modern networks and cloud services are operated through a dense layer of telemetry streams, tickets, runbooks, dashboards, and configuration artefacts. Over the last decade, this layer has grown faster than the teams that maintain it, largely because service graphs, deployment pipelines, and network policies now change continuously. In parallel, operators have accumulated powerful tools for reasoning about configuration and reachability (for example, offline analysis of routing and policy intent, and data-plane invariant checking) [1–3]. What is new is not that we automate operations, but that large language models (LLMs) can sit across this entire surface and turn fragmented operational work into a single loop that reads, reasons, and acts.

This operational burden is being intensified by three related trends: the *breadth* of the operational tool surface (more systems to query and more sources of truth), the *velocity* of change (deployments, policy edits, and topology churn), and the *volume* of interruption (alerts and incident handovers). In the field of SRE (Site Reliability Engineering), organizations aim to minimise detection and recovery times amidst increasing change frequency, making evidence collection and change security critical constraints [4,5]. The main advantage of LLMs lies in their ability to handle both procedural and linguistic tasks, both of which can be aided by tools. This perspective places agentic NetOps and AIOps within the context of autonomic computing. Kephart and Chess described autonomous systems as self-managing systems focused on configuration, remediation, and protection. LLM-based operational agents revisit this goal with modern tools, including natural-language interfaces, planning, and code generation. However, these tools also introduce new failure types, such as incorrect diagnoses, outdated evidence, unsafe tool use, prompt manipulation, and unrestrained execution [6].

LLMs transform the user interface by introducing natural language as the control layer for state queries and tool invocations. In early deployments, these models were primarily used for event aggregation, post-event analysis, or query building. As systems have evolved into agents that can plan multiple steps, call APIs or command-line tools, and change configurations directly, this shift has become more significant. Recent research demonstrates that by combining language inference with operational tools, language models can support event workflows, including monitoring and diagnostics [7–10]. This survey focuses on this shift and its impact on architecture, assessment, and security. However, LLMs should not be considered the only way to achieve intelligent operations. In various NetOps and AIOps scenarios, causal reasoning and dependency graph methods have provided interpretable and less computationally intensive fault location and hypothesis generation mechanisms; therefore, a core design question is: how should LLM agents complement rather than mask or replace these causal mechanisms?

Consider an on-call incident where a latency SLO is breached for a microservice after a routine deployment. An agent must (i) retrieve recent changes and ownership context, (ii) issue a small number of discriminating log/metric/trace queries, (iii) propose a mitigation such as a feature-flag rollback or traffic shift, and (iv) route any write action through policy checks, invariant validation, and canary rollout with rollback triggers. The same example later illustrates the tool loop, the verification wall, the evaluation ladder, and the threat model.

The literature relevant to agentic operations is scattered across networking, systems, software engineering, and security. Networking venues emphasise correctness and invariants under policy and topology dynamics [1–3,11]. Systems and SRE practice emphasise incident response latency, operational load, and the brittleness that comes from complex dependencies [4]. Software engineering venues increasingly study how LLMs help operators express intent and retrieve the right operational evidence [7,8]. Meanwhile, foundation-model research has produced patterns for tool use and iterative reasoning that are directly relevant to operations, but are rarely discussed with operational risk in mind [12–14]. The survey is timely because, although researchers are exploring agentic workflows, the field lacks shared terms for autonomy, a stable set of evaluation tasks, and a clear threat model that matches real operational tool surfaces.

Review questions and evidence discipline. The review asks an overarching question: *does operational assurance increase commensurately with the authority granted to an agent?* Three questions operationalise it: (RQ1) which operational tasks and tool interfaces are currently supported by LLM-based NetOps and AIOps systems; (RQ2) what evidence and independently enforced controls support claims at different autonomy levels; and (RQ3) how should workflow-level evaluation expose diagnosis quality, action safety, recovery, cost, and human intervention? Throughout the article, we distinguish four corpus statuses: *direct LLM-facing evidence* from evaluated operational components or agents; *domain-operational technical evidence* from NetOps, AIOps, cloud, SRE, and security-operations mechanisms; *cross-domain transferred evidence* from agent, verification, safety, and human–automation research outside those operational domains; and *normative infrastructure* from standards and official guidance. Author synthesis, including the contract schema and proposed metrics, is labelled separately and is not counted as an evidential status. This discipline prevents a transferred design principle from being reported as though it had already been validated in a write-capable LLM deployment.

1.1. Why agentic operations now

The structure of a running task is just as important as the functionality of the underlying model. In most cases, tasks involve a chain of queries and actions, often with missing details. An operator may need to check the blast radius, look for evidence in logs or counters, suggest ways to fix issues, make changes, and then watch for any problems that return. These steps follow a clear procedure. Much of the important context appears in work orders, manuals, commit messages, or handover notes. LLMs (Large Language Models) fit well here because they can work with both the process and the language found in these records.

(1) Running tasks require extensive retrieval. Much of the work in these settings comes down to searching for evidence and correlating information from several tools. Retrieval-enhanced designs address this need by joining model reasoning with dependable access to documents and system data [12]. In practice, the information we need is often present, but it can be hard to find. This is usually because of fragmented tools, changes over time, inconsistent names, or scattered records from past events.

(2) Tool invocation translates language into an executable plan. Modern LLM agents can select tools, maintain state across steps, and modify the plan as new evidence emerges [13,14]. This approach is similar to the workflow of an experienced operator, but agents can repeat the process more quickly on different interfaces. In incident management, this helps in developing systems capable of recommending investigative queries, summarising monitoring signals, and identifying potential root causes [7,8,10].

(3) Networks and clouds already have a correctness tradition. NetOps has mature ways to check whether a change violates invariants, breaks reachability, or conflicts with policy intent [1–3]. This tradition introduces a beneficial constraint on agent-based systems. An agent may propose action plans, but the entire system must check critical properties before any action is taken and keep changes within safe limits. As network operations have shown, finding new configurations addresses only one aspect of the problem. The real challenge lies in deploying these changes without causing temporary violations or disruptions [11]. However, these advantages also introduce new types of failures. Agent-based systems integrate interpretation and execution into a single operational flow. Misunderstandings of dashboards, unclear operating manuals, or imprecise objectives can all lead to changes that impact

system availability. Once languages become the programming means of operating tools, the previous assumption that languages were merely harmless interfaces no longer holds true.

Practical question: measurable operational objectives. Alongside the authority–assurance question, this article examines the following practical design question: *How should we design, evaluate, and manage NetOps and AIOps agents to maintain or even maximise reliability and security while reducing operational burden?* From an operational perspective, the goal is to improve outcomes such as diagnostic and mitigation times while meeting explicit security constraints. These constraints include policy compliance, limiting the scope of impact, and ensuring rollback readiness [4,5]. For this reason, the later discussion on evaluation focuses on workflow performance as the primary subject, rather than on static question answering.

1.2. Scope and definitions

We use **NetOps** to mean operational workflows specific to communication networks, including intent, configuration, troubleshooting, traffic engineering, and safe rollout [15–17]. We use **AIOps** to mean AI-supported IT operations, including log and metric anomaly detection, incident triage, diagnosis, and remediation [18]. We use **agentic** to describe a system that maintains state across steps, selects tools or actions, and executes multi-step plans, possibly with human approval at selected stages [19–21].

Terminology and evidential boundary. The scope is represented on two orthogonal axes. The first axis contains six analytical pillars: *operational capability and autonomy*, *architecture and tool grounding*, *assurance and operational control*, *evaluation and benchmarking*, *security, privacy, and adversarial robustness*, and *human factors, governance, and standards*. These pillars define which part of an LLM-enabled NetOps or AIOps system a record informs. Each record receives one primary pillar for reproducible distribution counts and may receive secondary pillar tags when its contribution spans concerns.

The second axis records evidential status. *Direct LLM-facing evidence* evaluates an LLM-based component or agent on a NetOps or AIOps task. *Domain-operational technical evidence* evaluates a relevant mechanism inside networking, cloud operations, AIOps, SRE, or security operations without necessarily using an LLM. *Cross-domain transferred evidence* supplies a mechanism from general agent systems, machine learning, software engineering, security, or human–automation research. *Normative infrastructure* defines interfaces, duties, or controls through a standard or official framework. The axes are intentionally independent. A classical network verifier can justify an assurance gate without demonstrating that an LLM agent has used that gate successfully, while an LLM benchmark can demonstrate task performance without establishing deployment safety.

We use *LLM-based operational agent* as the umbrella term. *NetOps agent* and *AIOps agent* identify the operational domain; *operational agent* is used only when a claim applies to both. An *agentic NetOps/AIOps system* is the surrounding system that combines the agent with evidence sources, tools, gates, rollout, rollback, and audit mechanisms. An *autonomy rung* describes the authority granted to that system. The *operational assurance contract*, shortened to *assurance contract* when unambiguous, is the review's rung-specific schema for permitted tools, required evidence, gates, budgets, rollout, rollback, and audit. An *operational contract* is a deployment-specific instance of that schema. An *agent card* is a reporting artefact that can document these obligations, but does not itself enforce them.

Autonomy ladder as an operational object. We model autonomy as a ladder of rungs. Each rung is characterised by which tools the agent may use and which gates are mandatory:

$$\mathcal{A}_k = \left(\mathcal{T}_k^{\text{read}},\ \mathcal{T}_k^{\text{write}},\ \mathcal{G}_k\right), \tag{1}$$

where $\mathcal{T}_k^{\text{read}}$ are read-only tools (queries and retrieval), $\mathcal{T}_k^{\text{write}}$ are write-capable tools, and $\mathcal{G}_k$ are non-bypassable gates (policy checks, invariants, approvals, canary and rollback rules) [3,11,22]. For clarity, write authority is separated in prose into proposal and execution capabilities, so a system may draft a diff or change request without being allowed to apply it. The formal tool partition is introduced once, with the assurance contract, in Section 4. Later sections use $\mathcal{A}_k$ to state what is being evaluated and what safety contract must hold.

Evidence traces for audit and scoring. The agent's interaction with tools is treated as a first-class output. To avoid repeating the same definition, the evidence trace is formalised once in Section 3, where it is used for budget, provenance, and scoring analyses. This makes the concept of "tool grounding" more concrete: the evaluation can determine whether the tracing results are discriminative, whether the cost is limited, and whether it is sufficient to justify the proposed operation [19–21].

Therefore, a practical approach to interpreting the term "agentic" in this survey is to view it as a ladder of autonomy. At the bottom is a read-only assistant responsible for aggregating and retrieving information. Mid-level systems can recommend queries, hypotheses, and manual procedures. Systems with higher autonomy can develop mitigation measures, generate configuration diffs, and run security checks. At the top is a closed-loop controller that continuously submits changes and verifies results. See Jiang et al. [8], Chen et al. [10] and Tian et al. [22]. Fig. 6 outlines the main roles, the tools each role uses, and the safeguards required at different levels.

We focus on the *operations tool interface*: monitoring backends, log storage, tracing systems, configuration repositories, CI/CD pipelines, network controllers, and ticketing systems. This boundary matters because security and governance depend on the tools an agent can invoke, the permissions attached to those tools, and the checks enforced before any action [8,10,18].

1.3. Survey method and corpus construction

We conducted a structured integrative review to make the survey scope, evidence roles, and analytical procedure explicit. It is an explanatory synthesis across several research traditions. The aim was not to count every paper mentioning LLMs, NetOps, or AIOps, but to identify work that explains how agentic operational systems observe evidence, propose actions, pass checks, execute changes, and remain auditable near network and cloud control surfaces. This sets a functional boundary: we include work that changes how operational agents are designed, evaluated, constrained, or governed, rather than work that only applies LLMs to operational text.

The historical search log records searches across 16 source families: Scopus, Web of Science Core Collection, ACM Digital Library, IEEE Xplore, Elsevier ScienceDirect, Springer Nature Link, Wiley Online Library, USENIX proceedings, ACL Anthology, arXiv, OpenReview/ICLR proceedings, MLSys proceedings, Google Scholar, IETF/RFC Editor sources, NIST publications, and official project or specification pages. The search covered work published from January 2020 through 28 August 2026, since most LLM-agent and tool-use work appears in this period. The principal search was completed through 31 May 2026, after which a targeted literature-status, gap, and forward-search update extended the final boundary to 28 August 2026. The supplementary update log records the refreshed query families, eligible additions, and publication-status checks. Earlier work was retained where it supplied a necessary foundation for the survey, including network verification, configuration analysis, safe network updates, distributed tracing, autonomic computing, SRE practice, and classical accounts of automation risk [1–4,6,11,23,24]. This earlier material is needed because agentic operations inherit many of their safety constraints from older systems work, even when the user interface is now a language model.

Searches were organised into six strands. The NetOps strand covered intent-based networking, configuration synthesis, network verification, safe updates, reachability, isolation, SDN, controllers, and telemetry

interfaces. The AIOps and SRE strand covered incident management, observability, anomaly detection, root-cause analysis, triage, remediation, rollback, and post-incident learning. The LLM-agent strand covered retrieval, tool use, structured outputs, planning, API calling, and multi-step operational workflows. The evaluation strand covered tool-use benchmarks, IT automation tasks, log and trace datasets, sandbox evaluation, trace replay, and workflow-level scoring. The safety and security strand covered prompt injection, indirect prompt injection, retrieval poisoning, telemetry manipulation, excessive agency, privacy, audit, and least privilege. The standards and practice strand covered OpenTelemetry, gNMI, NETCONF, YANG, intent-based networking definitions, telemetry frameworks, policy-as-code, and change governance. Search expressions were adapted to the syntax of each source, since academic databases, standards repositories, and Google Scholar do not expose the same query model.

Search-query families and reproducibility boundary. The historical search was not implemented as one fixed Boolean string applied unchanged to every source. Source-specific and targeted expressions were used across heterogeneous databases, publisher platforms, repositories, standards sources, and citation-search routes. To make the central LLM-era operational search logic repeatable, Supplementary Search Record and Reproducibility Strategy S1 provides two common concept blocks and six *consolidated reproducibility queries* aligned with the final analytical pillars: Q1 operational capability and autonomy, Q2 architecture and tool grounding, Q3 assurance and operational control, Q4 evaluation and benchmarking, Q5 security, privacy, and adversarial robustness, and Q6 human factors, governance, and standards. These consolidated queries summarise the recurring concepts used across the review; they are not claimed to be verbatim strings executed unchanged on every source and are not used to reconstruct the historical hit counts. The supplement separately describes the additional domain-operational, cross-domain, foundational, normative, citation-chaining, and venue-specific discovery lanes required to recover the broader 185-record evidence base, and it preserves the exact source-specific routes used in the final 28 August update.

Candidate records were screened in two stages. First, titles, abstracts, venues, and keywords were checked for relevance to at least one search strand. Second, the full text was checked against the survey question. A record was included if it met one or more of the following conditions: it studied a NetOps, AIOps, SRE, cloud operations, or security operations task; it proposed or evaluated a tool-using or retrieval-grounded LLM system for operational work; it defined a benchmark, dataset, or metric relevant to operational agents; it provided a verification, rollout, rollback, or governance mechanism that can serve as a gate for agentic systems; or it supplied a named mechanism used in a specific later claim about operational autonomy or assurance. Generic relevance to computing, networking, or AI was not sufficient. Records were excluded when they addressed generic chatbots, general prompting, or broad machine-learning methods without a clear operational task, tool surface, evaluation setting, or safety implication. Duplicate records and superseded versions were intended to be removed; the revision audit identified and removed one exact duplicate that had remained in the earlier reference set. Where both a preprint and a peer-reviewed version existed, the peer-reviewed version was preferred unless the preprint contained later system or benchmark details needed for this survey.

Screening and coding responsibilities. The review was conducted within the five-author team. Screening and coding were not independently duplicated, and the project did not preserve a per-record assignment log that would support a reliable count of independent screeners or coders. Borderline eligibility and coding decisions were discussed within the author team until agreement. No inter-rater coefficient was calculated, and no review protocol was prospectively registered.

The historical PRISMA search record identifies 4886 records across the 16 source families. Deduplication removed 1972 records, leaving 2914 unique records. Title and abstract screening excluded 2319 records, so 595 advanced for further assessment. Ten reports could not be retrieved with sufficient full-text detail, leaving 585 full texts assessed for eligibility. Full-text assessment excluded 394 records. The resulting bibliography contains 191 records, of which 185 form the claim-linked evidence base used for quantitative corpus analysis and six are contextual comparison surveys outside that denominator. Fig. 1 reports the identification, duplicate-removal, screening, eligibility, and inclusion stages.

The revision-stage reconciliation is distinct from the full search flow. The June 2026 version used a coded corpus of 181 records. A later claim-source audit removed 14 records, including one exact duplicate, leaving an audited base of 167. The final targeted update through 28 August 2026 added 18 eligible records, producing the final 185-record claim-linked evidence base. At the cut-off, these 185 records comprise 162 peer-reviewed conference or journal papers, 10 preprints, and 13 standards, specifications, books, technical reports, or official guidance documents. The public dataset supplies the 185-row evidence table, the 14 exclusions, the 18 update records, the operational-system subset, derived distributions, a data dictionary, and the update log [25].

Peer-reviewed work is used as the main evidence base. Grey literature, standards, and project specifications are used more narrowly, where they define an operational interface, control surface, benchmark, or governance requirement that is difficult to recover from academic papers alone. This treatment is important for agentic NetOps and AIOps because several relevant control surfaces are specified outside ordinary research articles, including telemetry standards, network-management interfaces, model-context protocols, and risk-management guidance [26–32].

Evidence roles and permissible claims. Each retained source is assigned one of four evidential statuses. *Direct LLM-facing* records evaluate an LLM-based operational component or agent on a NetOps or AIOps task. *Domain-operational technical* records evaluate a mechanism within networking, cloud operations, AIOps, SRE, or security operations. *Cross-domain transfer* records evaluate a relevant mechanism outside those operational domains. *Normative* records specify an interface, governance expectation, or security practice. A direct record can support a task-specific empirical claim. A domain-operational record can support feasibility of an operational mechanism within its evaluated setting. A cross-domain record can motivate a transfer hypothesis or design requirement but cannot demonstrate that an LLM NetOps or AIOps agent satisfies it. A normative record can define an obligation but cannot establish effectiveness. Proposed contract clauses, metric templates, and research priorities are identified as author synthesis unless a passage explicitly reports evaluated practice.

For source-type attribution, the direct LLM-facing evidence is further divided by the object evaluated: *operational systems* exercise an agent within a NetOps or AIOps workflow; *enabling frameworks* contribute an adaptable model, architecture, or orchestration method; *benchmarks* define tasks, interfaces, scenarios, or scoring oracles; and *safety evaluations* probe attacks, failure modes, or controls. Operational systems establish observed workflow behaviour, frameworks establish architectural feasibility, benchmarks support comparative measurement, and safety evaluations establish bounded risk or control findings. None of these objects alone demonstrates broad closed-loop autonomy.

Each included record was coded along eight dimensions: domain, operational task, autonomy role, tool surface, evidence role, evaluation setting, safety control, and contribution type. The domain codes were NetOps, AIOps, SRE, cloud operations, security operations, and general agent infrastructure. The task codes included triage, telemetry query generation, incident summarisation, root-cause analysis, configuration synthesis, change planning, remediation, rollback, and post-incident learning. The autonomy codes followed the ladder used in this survey: read-only copilot, tool-grounded analyst, write-limited planner, and closed-loop controller. Tool surfaces were coded as logs, metrics,

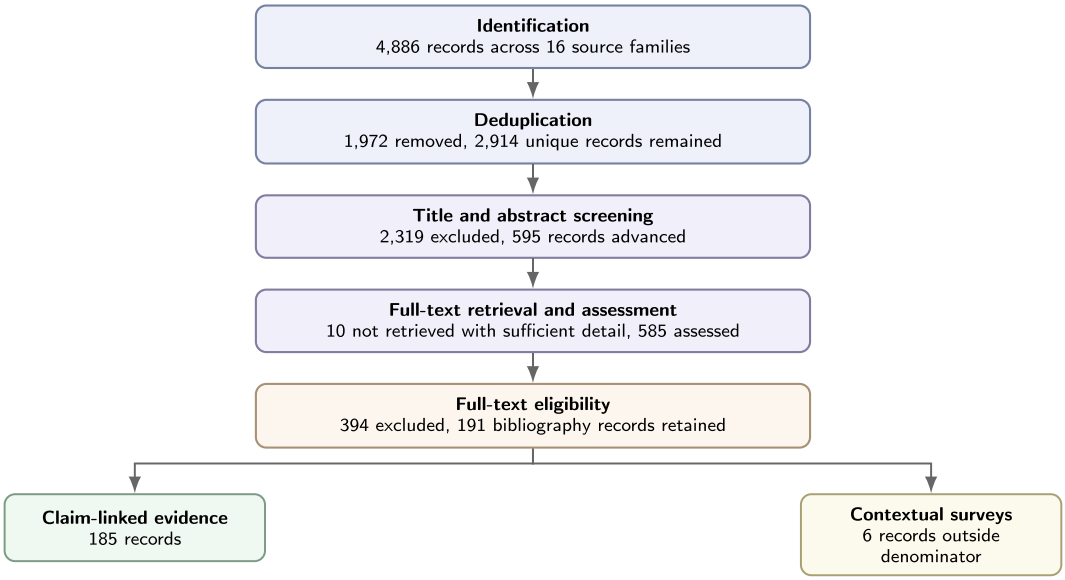


Fig. 1. PRISMA-style search and inclusion flow. The lower box shows the later corpus reconciliation separately from the historical identification and screening stages.

traces, tickets, runbooks, configuration repositories, topology stores, controllers, CI/CD systems, policy engines, validators, simulators, and action APIs. Evaluation settings were coded as static question answering, offline classification, trace replay, sandbox execution, live testbed, human study, production report, or benchmark suite. Safety controls were coded as least privilege, provenance tracking, freshness checks, schema validation, approval gates, policy checks, invariant verification, canary rollout, rollback readiness, audit logging, and red-team testing. Each record was also assigned one primary analytical pillar for non-overlapping corpus distributions and any applicable secondary pillar tags for cross-cutting interpretation. Autonomy, write capability, evaluation setting, and safety-control distributions are calculated only for direct operational systems. Frameworks, benchmarks, safety evaluations, domain-operational mechanisms, cross-domain foundations, and normative sources are marked not applicable rather than being forced into operational-agent categories.

This coding scheme supports the contract-centred taxonomy used later in the paper. For each work, we record what an agent may observe, which tools it may call, what evidence it must collect, which gates it must pass, how changes are rolled out or undone, and what budget or stopping rule limits the workflow. Model family and AIOps task name are useful labels, but they do not show whether a system is safe near an operational write boundary. For this survey, a read-only RCA assistant and a rollback-capable controller are different operational objects, even if both use an LLM.

1.4. Corpus characteristics and evidence distribution

The final claim-linked evidence base contains 185 records. By publication form of the screened artefact, 162 are peer-reviewed conference or journal papers (87.6%), 10 are preprints (5.4%), and 13 are standards, specifications, books, technical reports, or official guidance documents (7.0%). The 162 peer-reviewed records comprise 130 conference papers and 32 journal articles, using the publication types recorded in the accompanying machine-readable bibliography. By evidential status, 28 records are direct LLM-facing evidence (15.1%), 95 are domain-operational technical evidence (51.4%), 53 are cross-domain transferred evidence (28.6%), and 9 are normative infrastructure (4.9%). Publication form and evidential status are orthogonal. A peer-reviewed network verifier, for example, remains domain-operational evidence rather than direct evidence of an LLM agent. Every retained row records its first manuscript location, claim context, inclusion rationale, and permissible claim boundary.

Year and primary-domain distributions. Within the contemporary search window, the year counts are 7 records from 2020, 6 from 2021, 8 from 2022, 23 from 2023, 44 from 2024, 38 from 2025, and 5 from 2026 through the 28 August cut-off. The 54 earlier records are retained foundations from 1975–2019 and are not counted as direct contemporary LLM-agent studies. Using one primary-domain assignment per retained record, the distribution is NetOps 51, AIOps 28, cloud operations 14, SRE or software operations 15, security operations or agent security 19, general agent infrastructure 33, and transferred general infrastructure or general ML evaluation 25. These domain labels describe the role of each record in the synthesis and do not imply that every record evaluates an operational LLM system.

The six primary analytical pillars describe the review object more accurately than does a division between a small LLM core and a large adjacent periphery. Operational capability and autonomy contains 48 records (25.9%). Architecture and tool grounding contains 21 records (11.4%). Assurance and operational control contains 39 records (21.1%). Evaluation and benchmarking contains 31 records (16.8%). Security, privacy, and adversarial robustness contains 27 records (14.6%). Human factors, governance, and standards contains 19 records (10.3%). Percentages use one non-overlapping primary assignment per record and may not sum to exactly 100% because of rounding. Secondary pillar tags remain available in the supplement for records that contribute across several concerns.

The direct LLM-facing subset contains 28 records with distinct evidentiary objects: 13 operational systems, 4 enabling frameworks or models, 8 benchmarks or task-evaluation studies, and 3 safety or assurance evaluations. Autonomy and evaluation-setting distributions are derived only for the 13 operational systems: one read-only copilot, seven tool-grounded analysts, and five write-limited planners or bounded executors. No retained system demonstrates broad, independently assured closed-loop control. The direct records are also unevenly distributed across the pillars, with 12 in operational capability, 4 in architecture, 5 in assurance, 6 in evaluation, 1 in security, and none whose primary contribution is a longitudinal human-factors evaluation. These small and heterogeneous counts do not justify evidence-base-wide prevalence claims. They support the more cautious conclusion that our qualitative synthesis indicates a capability–assurance gap as authority approaches operational write access.

Comparable operational-system profile. For a limited descriptive cross-tabulation, the 13 operational systems were grouped by the evaluation

setting reported by the source. Seven use production reports, production incidents, deployment reports, or operational case evidence. Five use controlled benchmark, testbed, sandbox, experimental, or fault-injection settings. One is an early live operational prototype. The single R1 system is production-connected. Among the seven R2 systems, five are production-connected, one is evaluated in a controlled setting, and one is the early live prototype. Among the five R3 systems, one uses an operational case and four use controlled settings. Eight systems remain read-side. All five R3 systems demonstrate write-limited proposal or bounded execution, and four exercise bounded execution in an operational case, testbed, sandbox, or experimental environment. None evaluates a complete chain of proposal, independent verification, execution, and end-to-end rollback, and none demonstrates an independently assessed rollback mechanism. Reported controls are source-specific and overlapping, including read-only or scoped tools, schema or DSL validation, container or testbed isolation, guide or SOP bounding, and explicit stopping logic. We therefore report raw counts for the comparable authority and setting fields, but do not estimate corpus-wide safety-control prevalence from incompatible evidence objects.

The 28 August 2026 update adds 18 eligible records to the earlier 167-row audit and integrates them into the final denominator. The update contributes recent NetOps monitoring and intent systems, AIOps diagnosis and remediation agents, domain benchmarks, calibrated-confidence work, an operational-trust pipeline, and a cross-domain agent-security evaluation [7,33–49]. The complete 185-row evidence audit, 14-row exclusion audit, 28-record direct-evidence classification, 13-system operational coding, formula-derived pillar and evidential-status distributions, six contextual-review records, and final update log are deposited in the associated version-specific Mendeley Data record [25].

Backward and forward snowballing were then applied to the screened corpus. Backward snowballing identified foundational work on network correctness, tracing, incident practice, and benchmark construction. Forward snowballing identified newer work on LLM agents, operational assistants, tool-use evaluation, and prompt-injection risks. Snowballing stopped when an additional pass did not add a new operational task, tool surface, evaluation type, or safety control. This ties the review to saturation of the coding scheme rather than an open-ended citation search.

The method has two limits. Some operational agent systems are deployed inside companies and are described only through engineering reports, product documents, or not at all. In addition, the LLM-agent literature is changing quickly, especially in benchmark design and agent security. We therefore separate stable claims from current observations. Stable claims are grounded in older work on verification, SRE, tracing, least privilege, rollback, and audit. Current observations describe the present state of LLM agents for NetOps and AIOps, where the evidence base is still thinner and less even. This distinction matters because operational reliability is a slow discipline. The model interface may change quickly, but the need for evidence, gates, rollback, and accountability does not.

1.5. Positioning against prior surveys

The survey is related to several mature bodies of review work, but it is not a replacement for any one of them. Existing surveys mainly organise the literature around AIOps tasks, failure-management pipelines, network automation, LLM-agent architectures, retrieval methods, or tool-use benchmarks. Those views are useful. They do not, however, answer the central question of this survey: what must be true before an LLM-based operational agent is allowed to move from evidence gathering to proposed or executed change?

Table 1 summarises the distinction. Earlier AIOps surveys define the operational problem space well. Notaro et al. survey AIOps methods for failure management and organise work around failure-related tasks such as detection, diagnosis, and remediation [18]. Cheng et al. review AIOps on cloud platforms and give a useful account of incident detection, failure prediction, root-cause analysis, and automated actions [50]. More recently, Zhang et al. provide a broad LLM4AIOps survey, covering data sources, AIOps tasks, LLM-based methods, and evaluation practice in the LLM era [51]. These studies are closest to the AIOps side of this paper. Their main unit of analysis is the AIOps task or the LLM method. Our unit of analysis is the operational contract: which evidence is required, which tools may be invoked, which gates must be passed, and which actions may be proposed or executed.

Network automation surveys form the second related group. Leivadeas and Falkner survey intent-based networking, including intent expression, translation, assurance, and closed-loop network management [15]. Xu and Zhou survey systems approaches to configuration errors, including detection, diagnosis, testing, and prevention [52]. These works are essential for the NetOps side of the paper because they show that network operations already have a strong correctness tradition. The missing link is the agentic one. Existing network-automation surveys do not study LLM-mediated tool use, prompt-borne risk, provenance-aware retrieval, or rollback-aware evaluation for operational agents.

A third group surveys LLM agents and tool use. Wang et al. and Xi et al. review LLM-based autonomous agents from broad architectural and application perspectives [53,54]. Qin et al. survey tool learning with foundation models, while Li reviews prominent LLM-agent paradigms around tool use, planning, and feedback learning [55,56]. These surveys are valuable for agent construction. Yet they usually treat tools as capability extensions. In NetOps and AIOps, tools are also trust boundaries. A log query, a configuration diff, a controller API call, and a rollback command cannot be treated as equivalent actions simply because they are all tool calls.

A fourth group covers retrieval and evaluation. RAG surveys explain how external knowledge can improve factual grounding and freshness [57]. Agent benchmarks such as AgentBench, WebArena, API-Bank, OSWorld, SafeToolBench, and ITBench show how agent behaviour can be tested beyond single-turn question answering [19,20,58–61]. These works help frame the evaluation problem. Still, operational evaluation has extra structure. In an incident workflow, success is not only about whether an answer is correct. It also includes whether the evidence trace is adequate, whether tool use stayed within budget, whether the proposal passed independent checks, and whether rollout and rollback behaviour was scored.

The position of this survey is therefore narrower than a general LLM-agent survey and broader than an LLM4AIOps task review. It treats agentic NetOps and AIOps as constrained operational control. The survey asks how an agent observes, proposes, verifies, executes, rolls back, and leaves an audit trail. This contract-centred view is the main difference from prior work.

Relation to adjacent assurance artefacts. The assurance contract is not a replacement for an assurance case, safety case, access-control profile, policy specification, model card, or agent card. An assurance or safety case argues why a system is acceptably safe; a policy or access-control profile determines which operations are permitted; and a model or agent card reports characteristics and limitations. The contract schema instead supplies a compact, rung-indexed interface among these artefacts: it binds permitted tools to required evidence, independent gates, budgets, rollout and rollback duties, and an auditable trace. Its analytical consequence is the *capability–assurance gap*: our qualitative synthesis indicates that support is strongest for read-side querying and diagnosis, while evidence becomes markedly less complete as claims approach autonomous execution, canary management, and recovery.

1.6. Contributions of this survey

This paper makes the following contributions.

1. **Workflow-grounded framing:** We frame agentic NetOps and AIOps as operational workflows that move from observation and

Table 1
Positioning of this survey against prior surveys and adjacent reviews.

Prior survey line	Main focus	What it contributes	Remaining gap addressed here
AIOps failure-management surveys [18]	Failure detection, diagnosis, and remediation	Defines AIOps failure-management tasks and organises earlier ML-based methods	Does not treat LLM agents as tool-mediated operational actors with explicit write boundaries
Cloud AIOps reviews [50]	AIOps on cloud platforms	Covers incident detection, failure prediction, RCA, and automated actions in cloud settings	Does not give a contract view of evidence, permissions, verification gates, rollout, and rollback
LLM4AIOps surveys [51]	LLMs applied to AIOps	Surveys LLM-based data sources, AIOps tasks, methods, and evaluation	Centred on AIOps rather than joint NetOps and AIOps control surfaces. The write-side safety contract remains secondary
Intent-based networking surveys [15]	Intent expression, translation, assurance, and closed-loop networking	Provides the NetOps foundation for intent, policy, assurance, and automation	Does not address LLM-mediated reasoning, tool calls, prompt-borne risk, or agent audit trails
Configuration-error surveys [52]	Configuration errors in complex systems	Explains detection, diagnosis, prevention, and operational causes of misconfiguration	Pre-agentic. It does not cover LLM-generated diffs, tool-grounded planning, or gated execution
LLM-agent and tool-use surveys [53–56]	Agent architectures, tool use, planning, memory, and feedback	Gives the general agentic design vocabulary needed for LLM-based agents	Usually treats tools as capabilities. In operations, tools are also permissions, trust boundaries, and audit objects
RAG surveys [57]	Retrieval, generation, augmentation, and grounding	Explains how external evidence can reduce staleness and improve factual grounding	Does not model operational provenance, freshness, authority level, or write-side evidence obligations
Agent benchmarks and safety benchmarks [19,20,58–61]	Task completion, tool use, web interaction, desktop interaction, IT automation, and tool safety	Shows how agent evaluation can move beyond static question answering	Operational evaluation also needs trace quality, budgeted tool use, verification outcomes, canary behaviour, rollback, and audit scoring
This survey	Agentic NetOps and AIOps as constrained operational control	Organises the field around autonomy rungs, tool surfaces, evidence traces, gates, rollout, rollback, budgets, and audit	Provides a contract-centred taxonomy for agents placed near network and cloud control surfaces

diagnosis to decision-making and action. This shifts the unit of analysis from model architecture alone to the operational role an agent plays, the tools it may use, and the failure modes it may introduce.

2. **Contract-centred taxonomy:** We develop a review-informed, theory-led integrative taxonomy for agentic NetOps and AIOps, organised around autonomy rungs, tool surfaces, evidence obligations, safety gates, rollout constraints, and audit requirements.
3. **Architectural patterns and guardrails:** We extract recurring design patterns from recent systems, including retrieval and evidence management [12], tool-use loops [13,14], and operational assistants for monitoring, query recommendation, incident analysis, and network troubleshooting [7–10]. We map these patterns to guardrails such as verification, privilege minimisation, staging, and human approval.
4. **Evaluation practice and benchmark guidance:** We synthesise evaluation requirements into proposed, benchmark-instantiable task and metric templates that reflect operational work, including event classification, diagnosis under partial observability, safe change proposal, rollout, and rollback. We also discuss how benchmarks can measure end-to-end utility rather than isolated language quality, drawing on emerging agent evaluation practice [62].
5. **Security, governance, and compliance analysis:** We analyse risks specific to agentic operations, including unsafe tool invocation, privilege boundaries, audit integrity, and the tension between autonomy and accountability. We show how existing network verification and update abstractions can be repurposed as enforceable safety constraints [1–3,11].

Fig. 2 summarises the survey structure and its contract-centred progression from operational context to open problems.

2. Operational background: NetOps and AIOps

Operations is best understood as a control problem carried out under uncertainty and organisational constraints. Most incidents are not solved by one brilliant query. They are solved by repeatedly turning partial signals into safe, reversible actions while keeping the blast radius small. That is why "helpful text" is only a small part of the story. Key challenges include the collection of evidence, the proposal and review of changes, and the documentation of decisions for subsequent audit and organisational learning. Early research on Internet operations demonstrated that even minor configuration errors or policy mistakes can result in unexpected and widespread failures [63]. As systems increase in scale and software updates occur more frequently, the likelihood of such issues escalates.

2.1. A unified evidence-to-action control loop

Both NetOps and AIOps can be modelled as a partially observed control loop. There is a latent system state x_t (network and service state), operators and automation observe it only through tool-mediated observations o_t (logs, metrics, traces, reachability checks), and they apply actions a_t (changes, mitigations) that alter future state [1,8,10,18]. A compact abstraction is:

$$x_{t+1} \sim P(x_{t+1} \mid x_t, a_t), \qquad o_t \sim O(o_t \mid x_t), \qquad a_t \in \mathcal{A}_k. \tag{2}$$

The key operational point is the rightmost term. The agent does not act in an unconstrained action space. It acts within an autonomy rung k, which is defined by the tool permissions and the mandatory gates that must be satisfied before any write-side action is executed [3,11]. Following the introduction, we treat an autonomy rung as an operational object:

$$\mathcal{A}_k = \left(\mathcal{T}_k^{\text{read}}, \mathcal{T}_k^{\text{write}}, \mathcal{G}_k\right), \tag{3}$$

where $\mathcal{T}_k^{\text{read}}$ are read-only tools (queries and retrieval), $\mathcal{T}_k^{\text{write}}$ are write-capable tools, and $\mathcal{G}_k$ are non-bypassable gates (policy checks, invariants, approvals, rollout constraints) [3,11]. This makes "tools as a boundary of trust" concrete: the trust boundary is precisely the interface where a proposal becomes an executed action, and where $\mathcal{G}_k$ must hold.

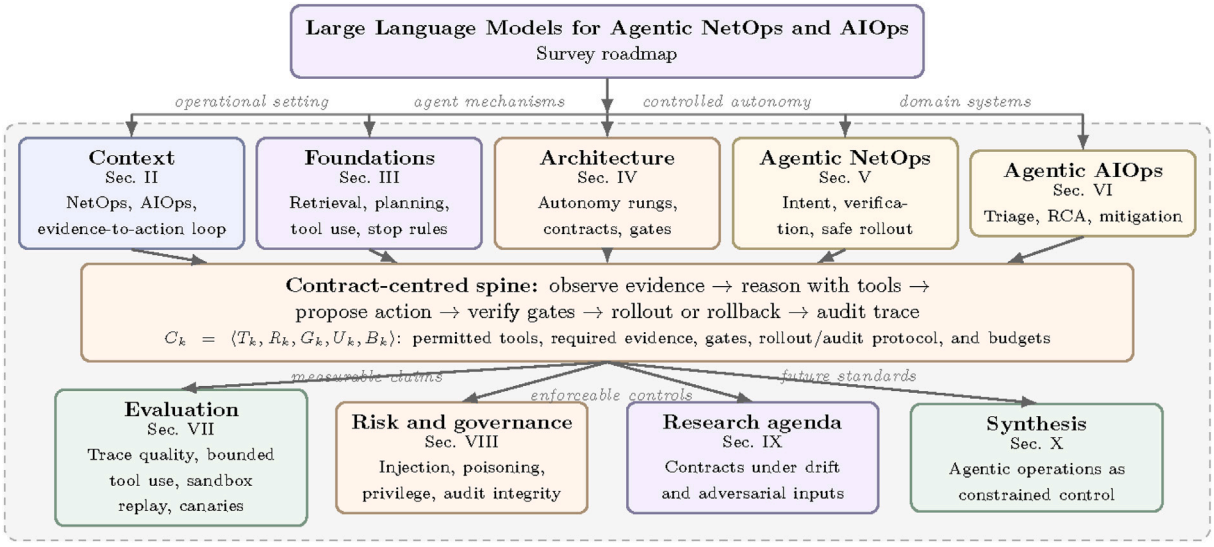


Fig. 2. Survey roadmap. Unlike the later control-loop, architecture, autonomy, and human-review figures, this figure shows only the article's argumentative progression from evidence and operational context to contracts, evaluation, governance, and synthesis.

A second shared primitive is mismatch between what is intended and what is actually realised [1,15]. Let x_t^{desired} denote the declared desired state (intent, policy, change request), and let x_t^{realised} denote the realised operational state (what the system actually does, as evidenced by measurements and derived models) [1,15]. We can express mismatch as:

$$\Delta_t = d\left(x_t^{\text{desired}}, x_t^{\text{realised}}\right), \tag{4}$$

where $d(\cdot,\cdot)$ is a domain-appropriate distance (for example, a binary violation indicator for an invariant, a count of violated intent clauses, or a normalised diff magnitude for a configuration slice). This is useful later because Δ_t is an evaluation axis: sound operational practice reduces Δ_t safely, or refuses to act when uncertainty is too high to reduce it without undue risk [3,11].

2.2. NetOps workflows and artefacts

NetOps encompasses the operational lifecycle of a communication network, and its workflow typically consists of five stages: capturing intents, integrating device or controller changes, validating against invariants and operational policies, executing phased deployment, and conducting post-change monitoring with rollback as necessary. In programmable networks, the controller has a central role in this process. This change began with Software-Defined Networking (SDN), which separated the control plane from the data plane. This separation gave operators a clearer way to manage the control plane [64]. Network operations (NetOps) depend on artefacts at multiple levels: desired states, such as intents and policies; rendered states, including configurations and controller rules; and realised states, encompassing forwarding behaviour, counters, and alarms. Issues frequently occur when these layers are misaligned. In notation (4), x_t^{desired} represents intents and policies, whereas x_t^{realised} denotes the resulting data-plane and control-plane behaviour. Intents are typically too abstract for simple verification, while vendor-specific configurations, although precise, are often difficult to interpret. Intent-based networking tackles this challenge by employing closed-loop transformation, execution, and monitoring, rather than relying on a single translation step [15].

Why configuration is operationally hard. Configuration is not simply parameter setting. It is a distributed process across heterogeneous devices, with asynchronous convergence and emergent behaviour. Errors arise from local mistakes, policy conflicts, drift, or fragile change sequences. Empirical work on BGP configuration errors shows that even minor policy mistakes can propagate and cause system-wide effects, which remains a warning for automation [63]. Even when the final state is correct, an incorrect update order may create unsafe intermediate states. This has motivated consistent update protocols and phased deployment designs for managing transient inconsistencies [65].

Causal diagnosis as a NetOps primitive. Not all NetOps diagnoses are suited to language reasoning or pattern recognition. A complementary approach uses causal inference, dependency models, and event structures to explain observed symptoms [66–68]. This approach is attractive in network operations because faults propagate through known relationships, including topology, protocol dependencies, control-plane interactions, service dependencies, and time-ordered event sequences [66,68]. Causal approaches can construct compact, interpretable fault graphs. When the underlying causal or dependency structure is known, the graph can reduce repeated resource-intensive inference over raw telemetry and operational manuals. For network operations, causal reasoning can therefore serve as a primary tool for $\mathcal{T}_k^{\text{read}}$. An LLM can query or summarise the model, while the causal graph makes the assumptions and evidence relationships explicit. This structured division can improve practical efficiency [66–68].

Validation and verification in everyday NetOps. In network operations, a change is normally checked before acceptance. Even a small configuration edit may alter reachability, isolation, waypointing, or policy. This need has produced tools that turn configurations into network models and check important properties before rollout. Batfish extracts the intended data plane from candidate configurations and supports pre-deployment what-if analysis [1]. Minesweeper broadens configuration verification by checking richer properties at scale while retaining operator-facing workflows [69]. In dynamic, controller-managed networks, verification must run at the write boundary, not only offline. VeriFlow inserts a layer between the controller and devices to block unsafe rules before they reach the data plane [3]. NoD adds a related lesson: operator assumptions about reachability, path selection, and service access must also be checked, since mismatches can expose policy gaps and security risks [70]. NetOps governance defines who may review, approve, or apply changes through configuration records, CI checks, controller interfaces, and ticketing systems. For agentic systems, this layer sets the permitted action boundary.

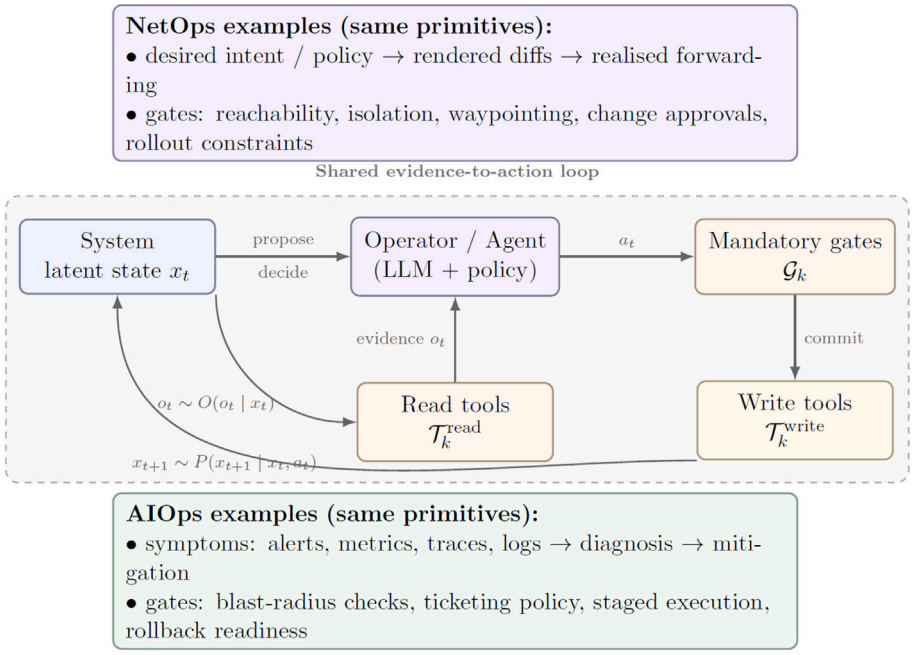


Fig. 3. Cross-domain analytical model. This figure aligns NetOps and AIOps as partially observed evidence-to-action loops: tools expose o_t, rung $\mathcal{A}_k$ bounds actions a_t, and mandatory gates $\mathcal{G}_k$ mediate write-side execution. It is distinct from the domain-specific workflow figures, which show task sequences.

2.3. AIOps workflows and artefacts

AIOps turns telemetry into incident evidence and response options. Its core workflow covers detection, triage, diagnosis, mitigation, and post-incident learning, often in a non-linear order.

Signals to decisions. AIOps begins with noisy and incomplete alerts. Triage filters common causes, while diagnosis links symptoms to plausible failures. In a sound process, tests rule out weaker explanations instead of supporting a merely convincing narrative.

Early systems work on problem determination in large services already stressed that diagnosis must be grounded in measured behaviour, since modern services are dynamic and failures often have several contributing causes [71]. As systems have grown in complexity, diagnostic practice has expanded to include provenance and the context of recent changes. Orca illustrates how operational debugging can be improved by tying incidents to code and deployment provenance, so that suspect changes can be ranked and inspected [72].

Artefacts: incident timelines, runbooks, and postmortems. Compared with NetOps, AIOps produces a broader and more detailed set of narrative artefacts [18,73]. Incidents generate timelines, chat transcripts, dashboards, on-call notes, runbooks, and postmortems [73,74]. These artefacts capture both policy and practice. They show what the organisation treats as safe, which actions are permitted, and what evidence must be gathered before mitigation is attempted [73,74].

They are also a source of systematic bias. Runbooks may be stale, postmortems may omit sensitive details, and tickets may contain ambiguous or adversarial text [73,75]. For agentic systems, these artefacts are both a knowledge base and an attack surface. They should therefore be treated as governed inputs rather than informal documentation.

From anomaly detection to actionable RCA. Early research in AIOps largely focused on incident detection. However, the true operational benefits lie in the actual mitigation of incidents [18,73]. This early focus led researchers to pay more attention to root cause analysis, often combining different types of data to support actual fault localisation [76,77]. Recent work in automated software engineering argues for multi-modal RCA at finer granularity, combining metrics with other signals to localise causes more precisely in microservice settings [76]. This reinforces a practical point: operations data is incomplete, and single-modality answers are often too fragile [75–77]. Fig. 3 follows the standard incident-response loop used in large-scale services and observability stacks, informed by classic service diagnosis and tracing systems [71,73,78,79].

2.4. Operational artefacts as data types: trust and freshness

Agentic operations systems inevitably consume and produce artefacts [18,73]. Treating these artefacts as data types, instead of as “just text”, makes the trust boundary visible. It also reduces accidental over-trust during incidents, when people and systems are already working under pressure. Fig. 4 summarises the two axes used in this subsection: trust level, from advisory to authoritative, and freshness, from fresh to stale. The figure also shows why the same artefact can change its operational status over time, which matters when retrieved evidence is used near the write boundary.

Trust level. Some artefacts are authoritative, such as policy-as-code, signed configuration baselines, and CMDB entries with ownership. Some are advisory, such as runbooks, postmortems, and tickets. Other inputs are considered untrusted, including free-form chat logs, externally provided documents, and user-influenced log strings. The operating rule is simple: untrusted artefacts can provide information for hypotheses, but without independent checks in $\mathcal{G}_k$, they must not be used to drive write operations.

Freshness and staleness risk. Operational knowledge ages [73,74]. Runbooks lag behind deployments, ownership rotates, and instrumentation coverage changes [74,75]. Freshness should therefore be tracked at the artefact level through timestamps, versions, and provenance. When a write action is under consideration, retrieval should favour recent and authoritative sources.

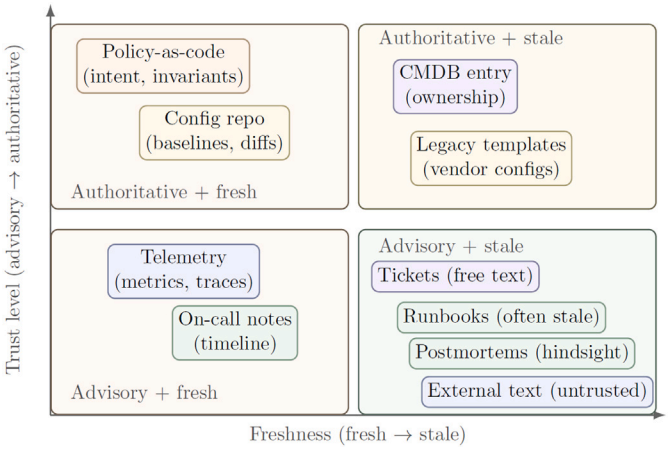


Fig. 4. Operational artefacts by trust level and staleness risk. The same artefact can move over time. For example, a runbook can become stale after a logging change. Retrieval therefore needs provenance and freshness signals when actions approach the write boundary.

Table 2
Operational artefacts as data types, with typical trust and freshness concerns.

Artefact type	Typical source	Trust class	Common staleness or failure mode
Policy/intent	policy-as-code, change requests [86,87]	Authoritative	Policy drift, incomplete translation into device or service configurations [86,87].
Config baselines /diffs	repositories, controllers [65,88]	Authoritative	Drift from live state, unsafe intermediate states during rollout [65,88].
Telemetry: logs/metrics/traces	observability stack [71,75,78,79]	Mixed	Missing coverage, misleading signals, sampling bias [18,75].
Tickets/runbooks	ITSM, wikis [73,74]	Advisory	Stale steps, unclear intent, injection through free-form text [73,74].
Postmortems /timelines	incident documents [73,89]	Advisory	Incomplete detail, hindsight bias, assumptions that no longer hold [73,89].

2.5. Observability modalities and tool surface

Agentic operations must connect different kinds of evidence, from live telemetry to standing policy, to the tools that expose or act on them. Table 2 summarises these artefacts by trust level and common failure mode. Each modality is a partial sensor for a distributed system. Each tool is either a read-side interface that produces observations o_t, or a write-side actuator that proposes or executes actions a_t, thereby triggering the gates $\mathcal{G}_k$.

Logs: rich context, weak structure. Logs carry useful local context, but they are often unstructured, noisy, and uneven across components. Log analysis therefore usually starts with parsing and template extraction, since later correlation depends on shared structure. Drain is a widely used example of online log parsing for streaming environments [80]. DeepLog shows that systems can learn usual log patterns and flag departures from them, although reliability still depends on timing robustness and parser accuracy [81]. Loghub supports evaluation across varied real-world logs and helps avoid overfitting to one system's logging habits [82].

Metrics: regular, scalable, and ambiguous. Metrics remain common in alerting because they are easy to collect, combine, and compare. Their weakness is ambiguity. Different failures can leave much the same numerical trace. Pipelines therefore attach labels and dimensions for service layout, deployment state, and other operating details. Standards such as OpenTelemetry aim to align these conventions, reducing the translation needed before diagnosis can begin [26].

Traces: causal hints with coverage gaps. Distributed tracing is valuable because it exposes request paths and latency contributors. By linking events through request context, tracing lets engineers ask causal questions rather than search through raw logs alone [78]. At scale, traceability remains both a systems problem and a data-management problem. Canopy addresses this through end-to-end traceability and analysis for high-volume production settings [79]. The limit is coverage. Traces are sampled, instrumentation is incomplete, and context propagation can fail across service boundaries.

Tickets and change records: the policy layer in disguise. Tickets and change records capture intent, approval, and accountability [73,83]. They are not telemetry, but they often hold the constraints that determine whether an action is safe [73,74]. They also help define operational ground truth, including whether mitigation was accepted, whether a change was rolled back, and whether a similar incident later recurred [83,84]. For agentic systems, incident-management and change-control interfaces should therefore be treated as first-class tool surfaces [73,85].

Tools as a boundary of trust. Operational tools determine what an agent can observe and what it can safely do. Query tools expose evidence. Change tools alter the system. This asymmetry is one reason operations differ from many general agent benchmarks. Read-only assistants may suit moderate-risk settings, while write-enabled agents require least privilege, staged deployment, and independent verification. Without these controls, the agent may simply automate failure, which is efficiency of the least helpful kind. In the unified model (2), this is the distinction between tools in $\mathcal{T}_k^{\text{read}}$ and tools in $\mathcal{T}_k^{\text{write}}$. It is also where the gate $\mathcal{G}_k$ must sit. Work on network verification and secure updates provides concrete precedents for this trust boundary [1,3,65,69]. Since tools cross these boundaries, the artefacts they consume and produce must be weighted according to whether they are authoritative or advisory, as shown in Table 2.

3. LLM foundations for operations

LLMs matter in NetOps and AIOps because they can coordinate evidence, hypotheses, tool calls, and change proposals across fragmented operational artefacts. They do not remove the older constraints of operations. Queries, dashboards, validators, runbooks, ticketing systems, change windows, approvals, and audit trails still define what can be safely done [4,5,73,88,90,91]. For this reason, the useful foundation is not a general account of prompting or language modelling. It is the interface between the model and the operational system: what the model may observe, which tools it may call, what evidence it must collect, which checks it must pass, and how unsafe or uncertain actions are stopped [7–9,32,85,92]. Fig. 5 sketches the resulting stack. The LLM sits between operational evidence and controlled action. Our synthesis suggests that operational reliability depends critically on the control system surrounding the model, especially as authority approaches write-capable interfaces. Typed tool interfaces, provenance-aware retrieval, explicit budgets, independent gates, and rollback-aware execution are therefore treated here as assurance mechanisms alongside model capability [12,13,55,93,94]. This section therefore keeps only the primitives needed by later sections: evidence traces, budgeted planning, freshness and provenance, controlled tool interfaces, the verification wall, human approval, and rollback.

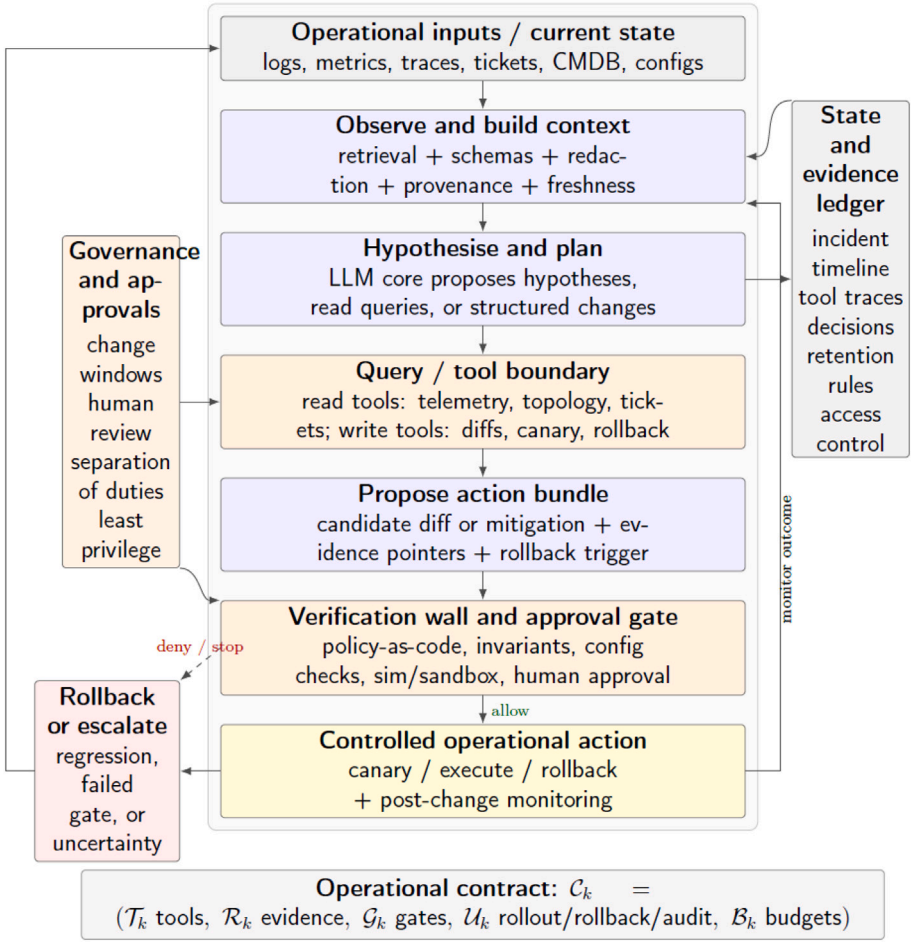


Fig. 5. Reference component stack. The model coordinates evidence, hypotheses, and proposals; typed tools, provenance logging, verification gates, approval paths, and rollback hooks define the enforceable system boundary. This component view is distinct from the autonomy taxonomy and task-specific workflow figures.

Status of the mathematical notation. The equations in this review provide a formal conceptual vocabulary for comparing operational agents; they are not theorem claims or evidence that a particular deployment satisfies a bound. The autonomy object, trace, budgets, gates, and score components make assumptions and reporting obligations explicit. Benchmark-specific quantities—including risk estimates, calibration bins, cost weights, stopping thresholds, and aggregation rules—must be instantiated and calibrated against a declared task distribution, oracle, horizon, and consequence model before numerical comparison. Where the article introduces a score or contract component rather than reporting an established metric, it is labelled as a proposed template. A further notation audit also removed repeated displayed definitions and re-statements that were not needed by later analysis. The remaining displayed equations are retained only where the same object is reused across the contract, evaluation, security, or research-agenda sections.

3.1. Model-level synthesis and attribution

Model choice matters, but operational results belong to the compound system unless an ablation isolates the model from retrieval, prompting, adaptation, tools, and control logic. Table 4 therefore retains the operational comparison as Panel A and adds a continuation, Panel B, recording the reported backbone and scale, prompting or agent strategy, retrieval-augmented generation, fine-tuning or domain adaptation, tool interface, and benchmark or evaluation setting. Missing implementation details are reported as not stated rather than inferred. This separation answers model-level comparison questions without attributing an agent's workflow result to its backbone alone. NetLLM is classified separately as an enabling framework because it evaluates networking-oriented model adaptation without, by itself, demonstrating operational authority or production assurance [95]. The update sharpens this attribution boundary. OWL adapts a LLaMA2-13B backbone with operations-specific instruction data, long-context processing, and a mixture of LoRA adapters, then evaluates the resulting model on Owl-Bench and log-analysis tasks [42]. OFCnetLLM reports an early multi-agent monitoring prototype on the OFC conference network, which supports system-feasibility and architecture claims but does not provide a mature comparative evaluation [33]. Network Meets ChatGPT proposes an intent-oriented management architecture, while Confucius provides production-facing evidence for multi-agent network management by representing operational workflows as DAGs, integrating LLM agents with existing management tools and retrieval, and coupling critical operations to validation mechanisms [36,96]. AIOpsLab and OpenRCA complement these systems with controlled evaluations of compound agents, operational tasks, and telemetry interfaces [43,44]. These records support model, architecture, system, or benchmark claims at their evaluated level. They do not by themselves establish independently assured production authority.

3.2. Evidence traces, budgets, and stop rules

An operational agent should be evaluated through its trace, not only through its final answer. We write the evidence trace as

$$E = \big((\tau_1, y_1), \ldots, (\tau_n, y_n)\big), \tag{5}$$

where τ_i is a typed tool call, including its arguments, and y_i is the corresponding output. This makes the agent's reasoning inspectable: a reviewer can see which logs, metrics, traces, tickets, configurations, validators, or simulators were queried, and whether the resulting evidence was sufficient for the proposed diagnosis or action [26,78,79,97,98].

Operational loops are also budgeted. Budgets are not only cost-control devices. They prevent open-ended tool churn during incidents and provide a mechanical point at which an agent must stop, escalate, or roll back [58,61,99]. We use

$$B = (B^{\text{tool}}, B^{\text{tok}}, B^{\text{time}}, B^{\text{risk}}), \tag{6}$$

where B^{tool} bounds tool calls, B^{tok} bounds context and generation, B^{time} bounds latency, and B^{risk} bounds the risk of committing a violating write action.

A compact planning constraint is

$$\sum_{i=1}^{n} c(\tau_i) \le B^{\text{tool}}, \qquad \sum_{i=1}^{n} \ell_i \le B^{\text{tok}}, \qquad \sum_{i=1}^{n} \Delta t_i \le B^{\text{time}}, \tag{7}$$

where $c(\tau_i)$ is a per-call cost, ℓ_i is token consumption at step i, and Δt_i is wall-clock time. For write-side risk, we use

$$\Pr\big(V_{\Pi,\mathcal{I}}(a_t, E) = 1\big) \le \epsilon, \qquad \epsilon \doteq B^{\text{risk}}, \tag{8}$$

where a_t is a candidate action, usually a diff plus rollout plan, and $V_{\Pi,\mathcal{I}}$ is a violation indicator under the relevant policy Π and invariants $\mathcal{I}$. This makes refusal and escalation measurable. A system can report when it stopped because evidence was missing, the budget was exhausted, the gate failed, or the estimated risk exceeded ϵ [31,32,100–103]. In a benchmark, the probability in Eq. (8) must be estimated over a declared incident and perturbation distribution using an independent violation oracle. The threshold ϵ is consequently a calibrated deployment parameter, not a universal constant supplied by the LLM.

MonitorAssistant provides complementary operational evidence from cloud-service monitoring. It uses LLMs to assist monitoring configuration, knowledge reuse, and alarm interpretation within an operational workflow [7]. The study is useful here because it places the language model inside an established monitoring process rather than treating model output as an independent operational decision. Its evidence supports assisted monitoring and diagnosis, but not autonomous remediation or independently verified write-capable control.

Stop rules should be explicit and mechanically enforceable. Common examples are budget exhaustion, missing required evidence, a failed gate, repeated contradictory observations, non-diagnosable ambiguity, and any request that would move the system outside the permitted autonomy rung. In operations, knowing when to stop is part of the safety case [89,104,105].

3.3. Tool-first evidence and controlled interfaces

Two operational facts explain why fluent text is a weak success criterion. First, evidence is produced by tools. In incident response, the system status is best determined by evidence provided by tools, such as logs, metrics, topology status, ticket records, and validators. Recent operational assistants therefore collect and structure diagnostic artefacts before asking the model to draw conclusions [7–10,106]. Second, actions must be reviewable and reversible. A diagnosis often implies a change, but a change must be expressed as a structured proposal, checked outside the model, and applied with limited permissions [86–88,107,108]. Work on router configuration synthesis shows the same pattern. Direct LLM output is brittle when used alone. Correctness comes from verifiers, repair loops, and explicit checks around the generator [93,109,110].

Operational retrieval is useful only when information remains fresh, attributable, and authoritative. Runbooks change after incidents, ownership shifts, and baselines drift. Stale instructions can be worse than none when they guide operational changes. The retrieval layer should therefore record source, validity period, trust level, and the presence of newer evidence [12,111,112].

Sources should be classified as authoritative, such as policies as code, CMDB entries, verified topologies, and configuration baselines; consultative, such as operating manuals, tickets, and post-event analyses; or untrusted, such as chat logs and externally supplied text. The class determines how the evidence may be used [100,101,113–115].

Operational tools should be typed, versioned, access-controlled, and scope-limited. Broad shell access should not be the default. Privileged tasks should use restricted APIs for validation, tracing, change requests, sandbox tests, canary deployment, and rollback. Tool failures should be expected, including poor tool choice, malformed parameters, dependence on output wording, and loops that continue after the task is complete [19–21,58,116,117].

Structured output links language to operational control. Typed assumptions, action proposals, configuration discrepancies, approval packages, and rollback plans can be parsed, replayed, checked against policy, and reused in regression tests from past incidents. This is important in network operations, where ambiguous intent may conceal conflicts in reachability, isolation, or priority. Ambiguities in configuration synthesis should therefore be resolved or rejected before a change is generated.

Algorithm 1 is normative pseudocode for the proposed contract, not an implementation evaluated in the retained studies. It identifies where independent policy, budget, and gate checks must occur. An implementation must also define concurrency, retry semantics, partial tool failure, idempotence, and transaction or compensation behaviour.

Termination is guaranteed only if all relevant budget counters are monotone, finite bounds are imposed, and the stop rules fire when those bounds or a finite retry limit is reached. If a planner may emit an empty or repeated plan without advancing a counter or invoking a stop rule, Algorithm 1 gives no termination guarantee.

3.4. Verification wall and gated planning

The verification wall is the non-bypassable boundary between proposal and execution. It turns agentic operation from free-form text generation into a controlled change process. Let $\mathcal{A}_k = (\mathcal{T}_k^{\text{read}}, \mathcal{T}_k^{\text{write}}, \mathcal{G}_k)$. Every candidate write action (a), or equivalently every tool call $\tau \in \mathcal{T}_k^{\text{write}}$, must pass a gate

$$g : (a, E, \Pi, \mathcal{I}) \mapsto \{\texttt{allow}, \texttt{deny}\}, \qquad g \in \mathcal{G}_k, \tag{9}$$

Here, E is the evidence trace, Π is policy, and $\mathcal{I}$ is the set of invariants. If g is sound for its declared model and checked property set, an allowed write satisfies only those modelled policies and invariants; this does not guarantee correctness under unmodelled state, stale evidence, checker defects, or actuator failure [1–3,69,118]. Planning must consider both tool limits and gate rules. Algorithm 1 shows this loop, which we use throughout the survey.

Safety risks. Safety matters because agentic operations connect untrusted text and telemetry to privileged tools. Prompt injection may enter through tickets, runbooks, chat logs, postmortems, or customer-provided text, so retrieved content should remain separate from system instructions, with narrow permissions, provenance records, and independent checks before action [101,102,115]. Telemetry manipulation can also mislead the agent by corrupting logs, metrics, traces, alerts, or topology. This is an evidence-integrity failure, not only a reasoning error. Defences include cross-checking telemetry, validating formats, tracking provenance, and refusing action when evidence is too weak [31,32,103]. The main operational safeguards are least privilege, independent verification, approval gates, canary rollout, and rollback plans [88,91,107,119]. An agent may identify the right cause and still fail operationally if it cannot show evidence, pass checks, prepare rollback, or leave an audit trail [61,92,105].

Algorithm 1 Normative planner–executor procedure under operational gates.

Input: incident signal s, autonomy rung $\mathcal{A}_k = (\mathcal{T}_k^{\text{read}}, \mathcal{T}_k^{\text{write}}, \mathcal{G}_k)$, policy Π, invariants $\mathcal{I}$, required evidence $\mathcal{R}$, budgets B, stop rules $\mathcal{S}$
$E \leftarrow \emptyset$, $H \leftarrow \emptyset$, $b \leftarrow 0$ evidence, hypotheses, budget counters
while not resolved **do**
  if $\mathcal{S}$ triggers on (E, H, s, B) **then**
    return escalate or rollback with evidence trace E
  end if
  $p \leftarrow \textsc{Plan}(s, H, E, \Pi, \mathcal{I}, B)$
  if not $\textsc{PolicyCheck}(p, \Pi)$ **then**
    return escalate with E
  end if
  for each $(\tau_i, args_i)$ in p **do**
    if $\tau_i \notin \mathcal{T}_k^{\text{read}} \cup \mathcal{T}_k^{\text{write}}$ **then**
      return deny action and escalate with E
    end if
    update budget counters b
    if b exceeds B **then**
      return stop and escalate with E
    end if
    if $\tau_i \in \mathcal{T}_k^{\text{write}}$ and not $\textsc{GateCheck}(\tau_i, args_i, E, \Pi, \mathcal{I}, \mathcal{G}_k, B^{\text{risk}})$ **then**
      return reject action, propose safe alternatives, and escalate with E
    end if
    $y_i \leftarrow \tau_i(args_i)$
    append (τ_i, y_i) to E
    $H \leftarrow \textsc{UpdateHypotheses}(H, E)$
    $s \leftarrow \textsc{UpdateSignal}(s, E)$
  end for
end while
return diagnosis, next safe actions, and evidence trace E

4. Agentic architecture patterns for operations

Agentic NetOps and AIOps systems differ mainly in their permitted authority [73,85,94]. We describe this through a ladder of autonomy, where each rung adds capability and risk. Lower rungs read, filter, and organise evidence. Higher rungs propose or execute operational changes. As autonomy increases, permissions, verification checks, approval paths, and rollback plans must be part of the architecture from the start [7–9,94].

Fig. 6 shows the role-level taxonomy used here. It separates read-only copilots, tool-based analysts, write-limited planner-executors, and closed-loop controllers by tool surface and guardrails. Table 3 then makes the same ladder operational by writing each rung as a contract covering allowed tools, required evidence, gates, rollout and audit obligations, and budgets or success criteria. The figure explains the roles, while Table 3 defines the controls that make those roles safe enough to evaluate.

4.1. Autonomy rungs as operational assurance contracts

We make the autonomy rung operational by treating it as a contract between the agent and the surrounding operational system. At rung k, the agent is defined not only by the model it uses, but also by what it may observe, which tools it may call, what evidence it must collect, which checks it must pass, how changes are rolled out or undone, and which budgets limit the workflow. We write this rung-indexed assurance contract as

$$C_k = (\mathcal{T}_k, \mathcal{R}_k, \mathcal{G}_k, \mathcal{U}_k, \mathcal{B}_k), \tag{10}$$

where $\mathcal{T}_k$ is the permitted tool surface, $\mathcal{R}_k$ is the required evidence to collect before a conclusion or action, $\mathcal{G}_k$ is the set of non-bypassable gates, $\mathcal{U}_k$ is the rollout, rollback, and audit protocol, and $\mathcal{B}_k$ denotes budgets for tool cost, latency, uncertainty, and action risk. The permitted tool surface is partitioned as

$$\mathcal{T}_k = \mathcal{T}_k^{\text{read}} \cup \mathcal{T}_k^{\text{propose}} \cup \mathcal{T}_k^{\text{execute}}, \tag{11}$$

where read tools retrieve or query evidence, propose tools generate reports, hypotheses, plans, or configuration diffs, and execute tools apply changes to an operational system.

The evidence trace remains a first-class output:

$$E = \big((\tau_1, y_1), \ldots, (\tau_n, y_n)\big), \tag{12}$$

where τ_i is a typed tool call and y_i is its output. For any candidate action a, gating is modelled as a non-bypassable decision function

$$g_k(a, E, \Pi, \mathcal{I}) \in \{0, 1\}, \tag{13}$$

which returns 1 only when the action is permitted under policy Π, is supported by the required evidence, satisfies the relevant invariants $\mathcal{I}$, and passes the gate conditions in $\mathcal{G}_k$ [1,3,87,120,121]. The shift from advisory operation to operational autonomy is therefore marked by the regime $\mathcal{T}_k^{\text{execute}} \neq \emptyset$, where verification, rollout, rollback, and audit become binding constraints rather than optional safeguards [32,92,119].

Contract satisfaction is a trace property. For a trace τ, we write $\tau \vDash C_k$ when the trace contains the mandated evidence, respects the relevant budgets, never bypasses the gate for executed actions, and follows the rollout and audit protocol:

$$\mathsf{Evidence}(\tau) \supseteq \mathcal{R}_k \quad \wedge \quad \mathsf{Budget}(\tau) \leq \mathcal{B}_k, \tag{14}$$

$$\wedge\ \forall a \in \tau \cap \mathcal{T}_k^{\text{execute}} : \ g_k(a, E, \Pi, \mathcal{I}) = 1, \tag{15}$$

$$\wedge\ \mathsf{Rollout}(\tau),\ \mathsf{Rollback}(\tau),\ \mathsf{Audit}(\tau) \text{ satisfy } \mathcal{U}_k. \tag{16}$$

This lets later sections compare agentic NetOps and AIOps systems by their operational contract, rather than by model family alone. Eqs. (10)–(16) are a proposed reporting and design template. They do not imply that every component is presently implemented or empirically validated at every rung. A benchmark must specify the concrete evidence oracle, gate implementation, budget units, rollout horizon, audit predicate, and failure handling before reporting contract satisfaction.

4.2. Ladder of autonomy with explicit tools, gates, artefacts, and success

Table 3 summarises the same four autonomy levels as operational contracts. For each rung, it lists the allowed tools $\mathcal{T}_k$, required evidence $\mathcal{R}_k$, gates $\mathcal{G}_k$, rollout and audit obligations $\mathcal{U}_k$, and budgets or success criteria $\mathcal{B}_k$. The purpose is to ensure that later architecture and evaluation sections refer to the same contract objects, rather than restating autonomy in prose.

4.3. Reference architecture and evaluation observables

Fig. 5 gives the reference architecture used throughout this survey. It condenses the safe-autonomy commitments that recur across incident assistants, tool-grounded diagnosis systems, and verification-heavy NetOps workflows [1,3,7–10,32,65,69,85,92,94].

The practical value of this architecture is that it makes later evaluation concrete. It exposes observables that can be measured across tasks, traces, and deployment stages [61,99,127]. Examples include:

- **Evidence quality:** completeness and correctness of E, including missing tool calls, incorrect citations, and stale artefacts [111,112].

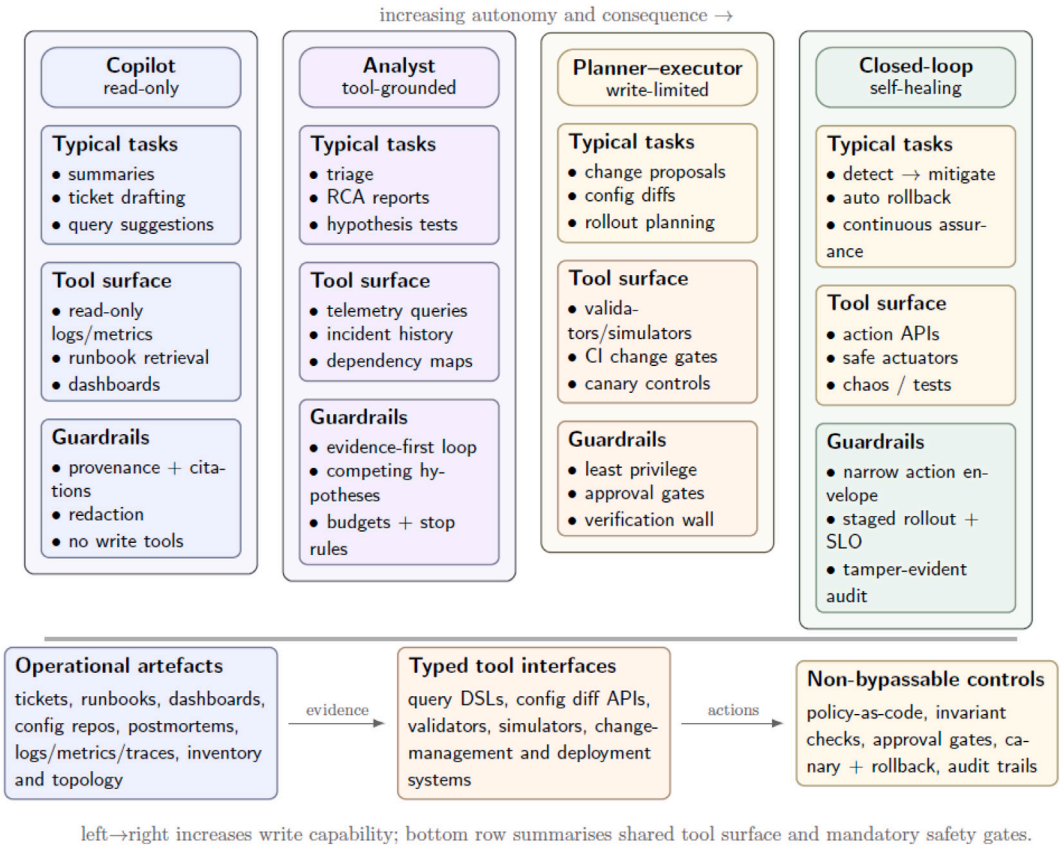


Fig. 6. Authority taxonomy for LLM-based operational agents. The figure classifies roles by read, proposal, and execution authority; Table 3 separately specifies the evidence and assurance obligations associated with each role.

Table 3
A ladder of autonomy for agentic NetOps/AIOps written as operational contracts. Higher autonomy shifts the engineering burden from answer quality to evidence obligations, gated action, least privilege, rollout, rollback, and audit [91,107,108,119,122].

Rung k	$\mathcal{T}_k$: allowed tools	$\mathcal{R}_k$: required evidence	$\mathcal{G}_k$: gates	$\mathcal{U}_k$: rollout and audit	$\mathcal{B}_k$: budgets and success criteria
Copilot (read-only)	Search, retrieval, summarisation, and query suggestion; no propose or execute tools [7,8,85]	Provenance-linked observations; cited source snippets; freshness markers where available [7,8,26]	Redaction, provenance rules, citation requirement, and rate limits [100,112,115]	Evidence trace E; incident summary; query candidates; no rollout obligation	Faster evidence location; fewer missed signals; low false citations; bounded tool calls [7,8,99].
Analyst (read-mostly)	Read tools plus diagnosis tools, including causal graphs, dependency models, event-correlation engines, and telemetry queries [9,10,106]	Evidence trace E; hypothesis set H; RCA evidence pointers; contradiction checks [10,76,77,123]	Citation to tool outputs; stop rules; escalation policy; no unsupervised write execution [10,85,92]	RCA report; audit-ready trace of queries and discarded hypotheses; human handoff where uncertainty remains	Correct top-k localisation; low tool churn; calibrated uncertainty; bounded diagnostic latency [61,76,77,124].
Planner–executor (write-limited)	Read tools, propose tools for diff synthesis, and execute tools limited to approved safe actuators [93,94,110]	Evidence trace E; candidate change diff Δ; preconditions; risk estimate; rollback evidence [88,107,108,122]	Non-bypassable verification wall; approvals; canary constraints; rollback triggers [1,3,91,107,120,121]	Staged rollout plan; rollback plan; signed change record; post-change observation window	No invariant violation in the checked domain; safe rollout; quick rollback on regression; bounded risk and time budget [1,3,69,108,123].
Closed-loop (self-healing)	Continuous read plus execute tools inside a narrow action envelope [107,108,125]	Continuous evidence trace E; monitor state; trigger condition; recovery evidence; periodic review artefacts [26,28,126]	Strict action envelope; automated rollback; continuous monitors; periodic audits [32,92,119,126]	Signed action log; recovery certificate where possible; rollback record; scheduled audit of repeated actions	Improved MTTR with bounded regressions; stable long-run drift Δ_t; bounded action frequency and risk [5,107,108,125].

- **Gate behaviour:** pass and fail rates of $g_k(\cdot)$, together with the reasons for rejection, such as policy violation, invariant failure, missing approval, or budget exhaustion [1,3,32,120,121].
- **Proposal quality:** validity of diffs Δ, satisfaction of preconditions, and behaviour under staged rollout [93,108,110,122].
- **Operational outcomes:** time to mitigate, rollback frequency, and post-change drift or mismatch metrics, for example Δ_t from Section 2 [5,73,83,123].

4.4. Human review as a concrete object

In guarded planner–executor systems, the human does not review a chat transcript. They review a *bundle* that can be scored, archived, and replayed [88,92,105,122]. We model the review bundle as

$$\mathcal{R} = \big(\Delta,\ \mathsf{Pre},\ \mathsf{Checks},\ \mathsf{Canary},\ \mathsf{Rollback},\ \mathsf{Ptrs}(E)\big), \tag{17}$$

Table 4
Representative evidence and model-level attribution mapped to the review schema. Panel A distinguishes operational LLM systems, enabling analytical frameworks, and benchmark components; Panel B reports backbone/scale, prompting or agent strategy, retrieval, adaptation, and tools without attributing compound-system results to the model alone [18,61,73].

System	Domain/task	Evidence object/autonomy	Tool surface	Evaluation type	Safety controls	Core survey use
DeepLog [81,147]	AIOps/log anomaly detection	Enabling analytical framework/read-only component	log streams, parsers, anomaly scores	offline classification; log-sequence prediction	thresholding, drift checks, operator review	Establishes a baseline for log evidence extraction. Realism depends on parser quality, drift handling, and whether alerts support later diagnosis or mitigation [147–149].
Xpert [8,73]	AIOps/triage and diagnosis	Operational LLM system/tool-grounded analyst	telemetry queries, DSLs, incident context	tool-use workflow; query recommendation	bounded tool budget, schema-constrained queries, feedback loops	Shifts evaluation from answer generation to evidence acquisition. Quality depends on the DSL, retrieved evidence, and reproducibility of the query trail [8,99].
RCACopilot [10,112]	AIOps/RCA	Operational LLM system/tool-driven analyst	diagnostic data collection, logs, traces, category prediction	workflow RCA; explanation quality	evidence linking, provenance, auditability	Shows that RCA should be evaluated as a workflow, not a single answer. Explanations are useful only when grounded in collected evidence [10,92].
RCAgent [61,106]	AIOps/RCA agents	Operational LLM system/tool-augmented analyst or planner	multi-step tools, telemetry, hypothesis refinement	agent trajectory; task completion; tool use	termination rules, tool budgets, unsafe-action penalties	Shows the need to score trajectories, not only final diagnoses. Evaluation should include safety requirements, cost accounting, and failure handling under partial observability [32,99,106].
TraceRCA [77,146]	AIOps/trace-based RCA	Benchmark and analytical framework/evidence construction	distributed traces, causal or temporal paths	offline localisation; benchmark comparison	causal consistency, supported edges, uncertainty checks	Supports causal structure and localisation. Higher autonomy evaluation still needs agent traces, counterfactual tests, and action outcomes [144,145].
TADBench [75,156]	AIOps/trace anomaly detection	Benchmark/escalation evidence	traces, anomaly scores, service dependencies	benchmark suite; anomaly detection under drift	drift testing, escalation thresholds, partial-observability checks	Helps assess when anomalies justify further investigation. Benchmarks should include drift, partial observability, and penalties for unsafe or unsupported escalation [99,156].
DeepLog	LSTM sequence model; not an LLM	Next-event prediction over parsed log sequences	Parsed log templates and execution paths	Trained on normal-operation sequences	Log parser and anomaly scorer	Analytical baseline; it isolates log modelling from language-agent orchestration.
Xpert	GPT-3.5 and GPT-4 variants; parameter counts not disclosed	Few-shot in-context KQL generation with post-rectification	Historical incidents and associated KQL records	Prompt-based main system; fine-tuned code models are comparators	KQL generation and validation pipeline	Quality reflects model, demonstrations, retrieval corpus, DSL constraints, and rectification.
RCACopilot	GPT-3.5-turbo and GPT-4 with an 8 K context in the reported study	Few-shot chain-of-thought category prediction after diagnostic summarisation	Predefined incident handlers collect diagnostic data; similar historical cases supply demonstrations	In-context main system; a fine-tuned GPT-3.5 comparator is reported	Incident handlers and diagnostic collectors	The evaluated effect belongs to the handler–retrieval–prompt–model pipeline, not GPT-4 alone.
RCAgent	Locally deployed Vicuna-13B-v1.5-16 K	ReAct-style controller, expert agents, JSON repair, and trajectory self-consistency	Tool observations, snapshot-key memory, and RAG over log chunks	No task-specific backbone fine-tuning reported; domain knowledge enters through tools and prompts	Cloud RCA diagnostic tools	Ablations are needed to separate backbone capacity from tool design, context management, and stabilisation.
OWL [42]	LLaMA2-13B base; OWL-13B after adaptation	Operations-specific instruction following with Homogeneous Markov Context Extension	Owl-Instruct and operations-domain corpora; no retrieval layer evaluated	Supervised tuning with a mixture of LoRA adapters across domains and tasks	No live operational tools; evaluated as a domain model	Results on Owl-Bench, log parsing, and anomaly detection establish model adaptation, not operational authority.
AIOpsLab [43]	GPT-4 and GPT-3.5 shell agents plus ReAct and FLASH variants; parameter counts not disclosed	Interactive agents plan across detection, localisation, RCA, and mitigation levels	Logs, metrics, traces, and live cloud state exposed through the benchmark interface	No benchmark-specific backbone fine-tuning reported	Shell, Kubernetes, telemetry APIs, fault injection, and workload control	Reported behaviour belongs to each agent–interface–scenario combination; the framework also evaluates tool cost and invalid API use.
OpenRCA [44]	Claude 3.5 Sonnet, GPT-4o, Gemini 1.5 Pro, Mistral Large 2, Command R+, and Llama 3.1	Sampling baselines and a ReAct-style controller–executor RCA agent with code generation	More than 68 GB of logs, metrics, and traces from 335 failures across three systems	No task-specific fine-tuning reported	Python execution over benchmark telemetry	Low complete-case accuracy shows that model choice, telemetry access, generated code, and error recovery jointly determine performance.

where Δ is the proposed change (diff or action plan), Pre are preconditions, Checks are required validations (policy-as-code, invariants, sandbox), Canary is the staged rollout plan, Rollback defines triggers and steps, and $\mathrm{Ptrs}(E)$ are evidence pointers into the trace [1,3,107,108,122].

Fig. 7 sketches this review flow, which is the practical interface for accountability [32,92].

4.5. Failure modes by rung

Autonomy changes which failures dominate [32,61,105]. At lower rungs, the main damage is wasted time, unsupported advice, and misleading explanations. At higher rungs, the main damage is unsafe change, delayed rollback, and hard-to-audit action [92,119].

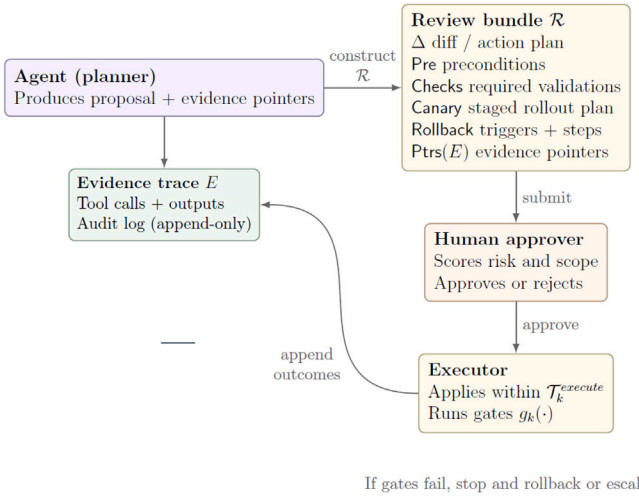


Fig. 7. Human decision interface for a guarded planner–executor. The review bundle $\mathcal{R}$ contains the diff, preconditions, completed checks, canary plan, rollback triggers, and evidence pointers into E; it does not imply that approval quality is automatic.

Design rule: autonomy raises failure severity As an agent moves from copilot to analyst, planner–executor, and closed-loop controller, its capability set changes from read-only evidence support to proposal generation and, eventually, bounded execution. Hallucination or story-mode reasoning can affect every rung, but tool misuse, prompt injection through operational artefacts, drift between verified and realised states, and rollback failure become more serious once the agent is allowed to propose or execute actions [83,101,103,115,122]. Higher autonomy is therefore acceptable only when the operational contract strengthens the tool boundary, evidence obligations, gates, rollout controls, rollback triggers, budgets, and audit record.

The design rule above is qualitative rather than an empirical measurement. It follows from each rung's capability set $\mathcal{T}_k = \mathcal{T}_k^{\text{read}} \cup \mathcal{T}_k^{\text{propose}} \cup \mathcal{T}_k^{\text{execute}}$ and gate strength $\mathcal{G}_k$. Risks driven by write capability increase when $\mathcal{T}_k^{\text{execute}} \neq \emptyset$, while strong, non-bypassable gates reduce the likelihood that these risks propagate into executed actions [1,3,119–121]. Table 3 operationalises this autonomy–risk coupling by specifying, for each rung, the allowed tools, required evidence, gates, rollout and audit duties, and budgets. The qualitative takeaway is that gating and verification are not optional extras at higher rungs. They are binding constraints once $\mathcal{T}_k^{\text{execute}}$ is non-empty [32,92].

4.6. Pattern summaries tied back to the contract

The following patterns are designed to align with the contract object $(\mathcal{T}_k, \mathcal{G}_k, E, \Pi, \mathcal{I})$.

Pattern A: Co-pilot (Read-only). The Co-pilot pattern improves operator throughput by compressing and navigating evidence. Its security depends primarily on the source, editing, and faithful referencing of the tool's output. The measure of success should be faster discovery of relevant evidence and fewer missed signals, rather than a more polished incident summary [7,8,85].

Pattern B: Analyst (tool-grounded diagnosis). The Analyst agent runs read-only queries, keeps several hypotheses in view, and produces a structured root cause analysis (RCA) report with traceable evidence clues. Its main failure mode is narrative drift: a coherent explanation that is only lightly attached to the system under investigation. The design should therefore require tool-based citations, checks against contrary evidence, and clear stopping conditions. If uncertainty remains, the agent should escalate rather than close the case on a tidy but weak explanation [9,10,76,77,106,123].

Pattern C: Planner–executor (write-limited). This is the first rung where safe autonomy becomes a property of the whole system. The agent proposes diffs and plans, while execution is constrained by least privilege and by a verification wall that checks policies and invariants [1,3,69,94,120,121]. Success is not only about whether the change was applied. It is about whether the change was applied without violating checked properties, and whether the system could return quickly to a known-good state if reality objected [108,122,123].

Pattern D: Closed-loop (self-healing). A closed-loop system aims to integrate detection, diagnosis, mitigation, and recovery verification into a continuous process. At this rung, since the system is part of the control loop, the range of action must be very limited, and rollback needs to happen automatically. Thus, the main evaluation metric should be the system's long-term stability in the face of faults and changes, rather than the outcome of any single event [32,107,108,125].

5. Agentic NetOps

NetOps is a natural target for agentic systems because many operational tasks already follow a repeatable loop: (i) gather evidence from measurement and control-plane state, (ii) form a hypothesis about a failure or a policy violation, (iii) propose a change, and (iv) validate the change before rollout [15,16,29,73,88]. At the same time, NetOps has a higher safety bar than many AIOps settings because a single incorrect change can propagate quickly, trigger large blast-radius outages, or create hard-to-debug transient behaviours during convergence [11,65,128,129]. Empirical studies show that operational failures in networks often arise from simple causes, including component faults, misconfigurations, and side effects from other configuration changes. Although these causes may appear simple, they are difficult to predict without systematic testing [52,63,83,128,129].

This combination of structured workflows and high consequence makes NetOps a useful stress test for agentic LLM design. The LLM is rarely the correct place to "decide truth" about network state. Instead, the LLM is most valuable as a controller of *workflow*: it translates intent into queries, routes evidence through verifiers, proposes minimally invasive diffs, and documents preconditions and rollback [9,93,94]. Fig. 8 summarises this workflow view: evidence is gathered through tools, intent is refined before planning, and every write-side action is routed through verification, staged rollout, rollback triggers, and assurance.

In other words, NetOps rewards agentic systems that are tool-first and verification-first, rather than conversationally fluent [1,3,121,130]. This separation also leaves room for specialised, network-native control mechanisms such as NOS [131], where fast adaptation is handled by the control layer rather than by the language model itself. For example, in O-RAN scheduling, the lower-layer manages latency tails and spectrum targets. Meanwhile, higher-level systems interpret intent, select policies, and oversee auditable changes [132]. This positioning is consistent with practical NetOps verification and safe-change practice, where the hard part is not proposing a change but justifying and checking it before it is trusted [1,3,69,120,133].

5.1. NetOps property definitions (targets for the verification wall)

To make "verification" concrete, we fix a compact model and a small palette of properties that recur across the NetOps literature [1,2,69,120]. Let the network be a directed graph $G = (V, E)$. Let x denote the realised control/forwarding state (routing adjacencies, FIB entries, ACL tables, tunnel state, etc.). For a traffic class c (e.g., 5-tuple predicate, VRF, DSCP class), let $F_x(v, c)$ denote the next-hop relation induced by x.

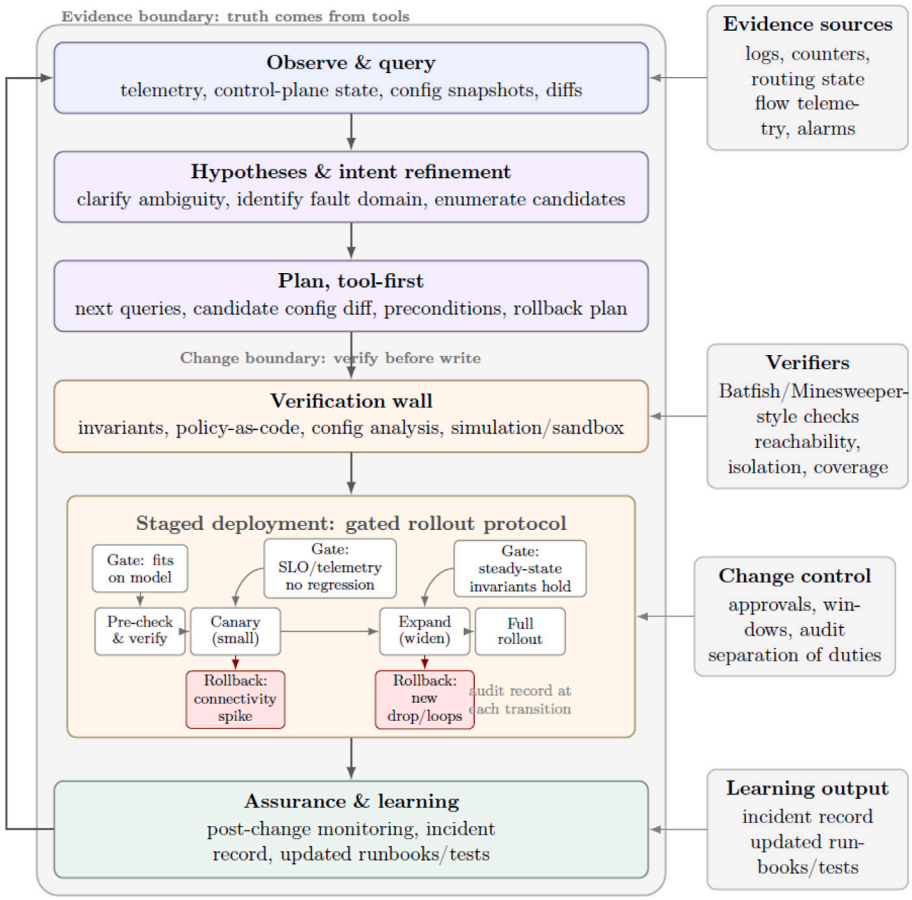


Fig. 8. Agentic NetOps loop specialised to high-consequence change. The LLM is useful as a workflow controller: it gathers evidence, clarifies intent, proposes minimally invasive diffs, and routes every write through an explicit verification wall, staged rollout, and rollback path.

This induces a (possibly empty) forwarding path $\mathsf{Path}_x(s,c)$ obtained by iterating F_x from source s until termination (deliver, drop) or a loop. These abstractions match the intent of data-plane and control-plane analysis systems that operationalise reachability and policy reasoning from configurations [1,2,70,120].

We use the following predicates as reusable invariants, illustrated in Fig. 9:

$$\mathsf{Reach}_x(s,t,c) \equiv \mathsf{Path}_x(s,c) \text{ terminates at } t \text{ without drop or loop,} \tag{18}$$

$$\mathsf{Isolate}_x(s,t,c) \equiv \neg\mathsf{Reach}_x(s,t,c), \tag{19}$$

$$\mathsf{Waypoint}_x(s,t,w,c) \equiv \mathsf{Reach}_x(s,t,c) \wedge w \in \mathsf{Path}_x(s,c), \tag{20}$$

$$\mathsf{LoopFree}_x(s,c) \equiv \mathsf{Path}_x(s,c) \text{ contains no repeated node} \tag{21}$$

These cover the most common operational questions: "can A reach B", "must A never reach B", "must traffic pass through a firewall/proxy", and "are we creating transient loops". Fig. 9 also makes explicit why rollout safety is stricter than final-state correctness: the invariants should hold not only after the final configuration is installed, but also across intermediate states during canary, expansion, and full rollout. They align with what configuration analysers and invariant checkers are typically asked to prove or refute at scale [1,3,17,69,121,133].

5.2. Intent, ambiguity, and when clarification is mandatory

Natural language intent is typically underspecified, even when expressed by experienced operators [15,16,29,134]. Intent often omits operational constraints that matter for safety, such as acceptable blast radius, maintenance window, traffic class priorities, failure-domain boundaries, or whether a waypoint is "preferred" versus "mandatory". Intent-based networking is typically conceptualised as a closed-loop system that provides continuous assurance, rather than as a single compilation step [15,16,29]. Prior to the advent of LLMs, research approached configuration as a synthesis problem defined by explicit correctness criteria, which typically focused on translating high level policies into device-level configurations, ensuring invariants are maintained, and protocol semantics are respected [17,135–137].

For agentic NetOps, this suggests a safe division of labour: the LLM handles ambiguity resolution and produces structured candidates, while synthesis and checking back-ends provide the authority on correctness [93,110,134].

Recent domain evidence supports this division without eliminating its limits. Network Meets ChatGPT organises intent translation, autonomous management, and network control as a layered architecture rather than a free-form command path [36]. The LUMI pipeline in Jacobs et al. uses operator confirmation, grammar-constrained intent assembly, ambiguity detection, and post-deployment monitoring to make natural-language intent more trustworthy, although its evaluated information-extraction component is not an LLM [46]. A YANG-grounded LLM agent further shows that schema grounding can constrain candidate configurations on a multi-vendor OpenROADM testbed [34]. Together, these studies support typed intent records, clarification, and independent validation. They do not establish unrestricted autonomous configuration of production networks.

Recent work makes ambiguity explicit and studies how an LLM-driven assistant should ask questions *before* generating configurations

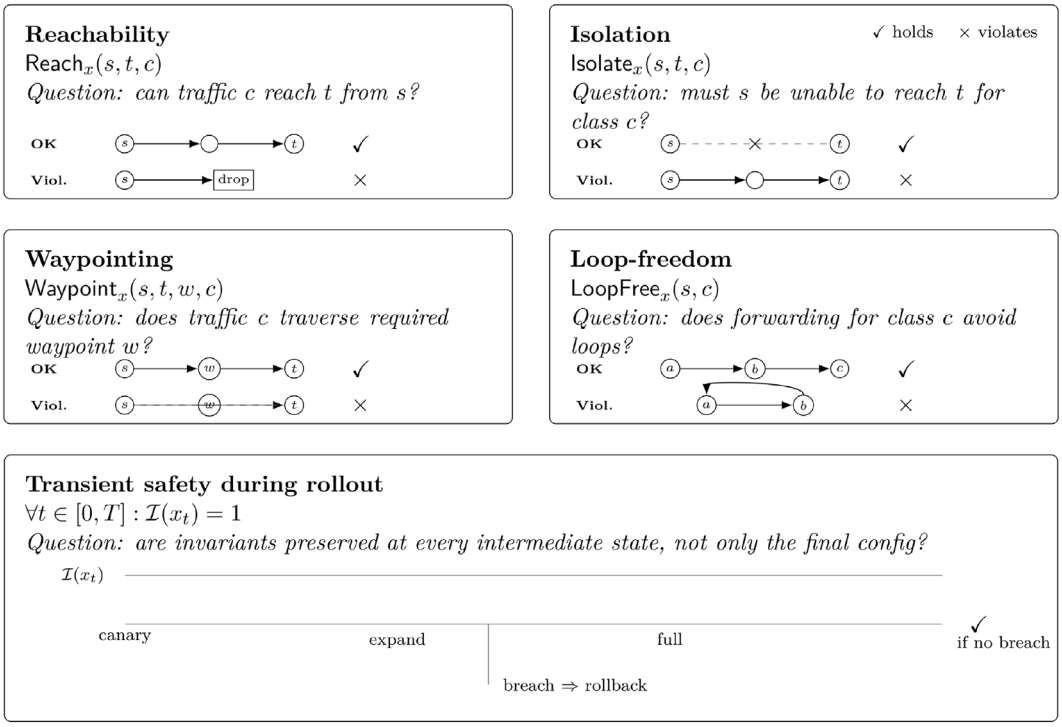


Fig. 9. Common NetOps invariants that can form the verification wall. The point is not the notation; it is to make the targets explicit and reusable across synthesis, rollout safety, and autonomy gating.

[93,134]. A practical and checkable condition for *mandatory clarification* is that there exist multiple distinct diffs that satisfy the current checks but induce different forwarding semantics for some traffic class. Let $\mathcal{S}$ be the set of diffs Δ that pass the current policy and invariant checks. We have an ambiguity witness when:

$$\exists\, \Delta_1, \Delta_2 \in \mathcal{S},\ \Delta_1 \neq \Delta_2 :\quad \exists (s,t,c) \text{ s.t} \\ \mathsf{Path}_{x \oplus \Delta_1}(s,c) \neq \mathsf{Path}_{x \oplus \Delta_2}(s,c). \tag{22}$$

In practice, the most common sources of such witnesses are rule overlap and priority conflicts, where an operator's intent does not specify precedence even though the configuration must [17,134,137]. Under these conditions, the agent should not "pick a reasonable default". It should ask a small typed set of disambiguating questions, commit the answers to the intent record, and only then generate diffs [93,134].

Finally, synthesis is not only about producing configs but also about supporting review and post-incident learning. Work on producing localised explanations for synthesised configurations supports this direction and fits naturally with agentic workflows where explanations are linked to specific lines, policies, and checker outputs [109,110].

5.3. Update safety: correctness during rollout, not only at rest

Update safety is not only about the final configuration being correct. Even correct end states can be reached through unsafe intermediate states, such as loops, black holes, or policy violations during rollout, because updates are applied under conditions of asynchronous convergence and partial deployment [11,64,65]. Agentic NetOps should treat "apply change" as a protocol, not a single tool call [11,88,91].

Let x_t denote the realised network state during rollout, as partial deployments and convergence events take effect. Transient safety requires every intermediate state to satisfy the relevant invariants:

$$\forall t \in [0,T] :\ \mathcal{I}(x_t) = 1, \tag{23}$$

where $\mathcal{I}$ combines the required reachability, isolation, waypointing, and loop-freedom checks. This separates final-state correctness from rollout correctness, which is the key lesson of consistent updates and staged operations [11,91,138].

5.4. Change risk scoring and autonomy gating

NetOps practitioners assess risk using criteria that align with autonomy decisions, such as the number of affected devices, novelty of the change, the adequacy of verification procedures, and the history of similar changes causing incidents [83,88,91]. Agentic systems benefit from making that reasoning explicit because it links verification coverage to permissioning and gating, rather than treating coverage as an afterthought [32,139].

Let Δ be a candidate diff (or change bundle). A minimal risk score that is easy to compute and easy to review is:

$$\mathsf{Risk}(\Delta) = \alpha\, \mathsf{Blast}(\Delta) + \beta\, \mathsf{Novelty}(\Delta) + \gamma\, \big(1 - \mathsf{Coverage}(\Delta)\big). \tag{24}$$

where $\mathsf{Blast}(\Delta)$ approximates potential impact, $\mathsf{Novelty}(\Delta)$ measures distance from known-safe patterns, and $\mathsf{Coverage}(\Delta)$ measures how well tests and checks exercise the configuration lines and behaviours touched by Δ. Recent coverage work shows why this is operationally important: "we have tests" is not the same as "we have assurance for the lines we have just changed" [121,139]. In practice, when coverage is low for the touched lines, autonomy should be reduced rather than expanded [91,139].

This connects directly to the earlier ladder-of-autonomy and gate definitions. A simple execution gate can include a risk threshold:

$$g_k(\Delta, E, \Pi, \mathcal{I}) = \mathbf{1}\Big[\ \mathsf{PolicyCheck}(\Delta,\Pi) = 1 \\ \wedge\ \mathsf{Verify}(\Delta,\mathcal{I}) = 1 \wedge\ \mathsf{Risk}(\Delta) \le \theta_k\Big]. \tag{25}$$

This makes "validation barriers" an important principle. The scope of coverage and the risks decide whether an agent can carry out a plan, not just whether they can have a major effect [1,3,32,120,139].

5.5. Troubleshooting: from symptoms to hypotheses with reproducible traces

Network troubleshooting is typically a search over an incomplete set of evidence. This is because signals are often noisy, telemetry data may be missing, and symptoms can deviate from the root cause due to retries, buffering, and control plane convergence. Therefore, practical application relies on a standardized workflow: collecting discriminative observations to narrow down the fault, and then validating hypotheses through targeted testing [140]. The same idea carries over to agentic NetOps. The agent should be rewarded for producing a reproducible evidence trail E, not for writing a smooth account of events. Recent dialogue-driven diagnosis in data centre networks points in the same direction: success depends on workflow discipline, tool sequencing, and explicit escalation when confidence remains low [9].

Cross-layer cases are still the most difficult. Application symptoms may originate from routing policy, transport behaviour, middlebox policy, or transient forwarding states during updates. Agentic NetOps systems should therefore represent hypotheses at several layers and test them with independent signals. A practical rule is to maintain competing hypotheses and require at least one disconfirming test before committing to any change with a broad blast radius.

5.6. Causal inference for explainable NetOps diagnosis

Causal inference provides a useful counterweight to purely LLM-centred NetOps diagnosis. Many network failures are not isolated events; they propagate through topology, protocol state, control-plane dependencies, service dependencies, and time-ordered event streams [66,68]. A causal diagnosis method aims to recover this propagation structure, so that the output is not only a ranked root cause but also an explanation path linking symptoms to candidate causes [67,68].

Earlier NetOps work already reflects this idea. G-RCA models service-quality problems in large IP networks using dependency relationships, temporal and spatial event correlation, and reasoning logic, showing that operator-facing RCA benefits from explicit dependency models rather than opaque prediction alone [66]. Work on mining causes of network events from log data uses causal inference to go beyond co-occurrence and recover likely causal relationships among network events [67]. Later work further combines causal inference with protocol-layer and topology knowledge, reducing spurious causal edges and producing information that is more useful for troubleshooting [68].

A closely related cellular-network line of work studies causal inference for estimating or explaining the operational effects of radio-access parameter and software changes. For example, causal models have been used to estimate the impact of handover-parameter adjustments and other cellular-network configuration interventions, including transmission-power and cell-offset changes, on subsequent service and performance time series [141–143]. This is especially relevant to agentic NetOps because it shows that some root-cause and change-impact questions can be answered through structured causal models before an LLM is asked to reason over large telemetry and runbook contexts.

For agentic NetOps, these methods should be treated as diagnostic back-ends rather than competitors to the agentic framing. A useful division of labour is: causal inference constructs or updates a fault graph; the LLM turns operator intent into queries, requests additional evidence, explains the causal path, and prepares a reviewable mitigation proposal [66–68]. This hybrid design has two advantages. First, it improves explainability, because the diagnosis is grounded in a causal path rather than a free-form narrative. Second, it can reduce repeated reasoning over raw telemetry and runbook context, because the causal graph narrows the candidate space before the LLM is asked to summarise, compare, or prepare an action proposal [67,68].

5.7. Repair as an iterative workflow

There is growing interest in combining localisation, repair, and validation into an integrated loop. Automatic Configuration Repair is illustrative: it frames misconfiguration handling as localise–fix–validate, which matches how operators actually de-risk changes during incidents [110]. This maps well to agentic design because it encourages iterative, checkable steps and makes rollback and validation central rather than decorative.

5.8. Putting it together: why NetOps is the sharpest testbed

NetOps makes agent-safety claims concrete because it offers: (i) checkable invariants, (ii) a known failure mode of transient rollout unsafety, and (iii) a principled link between test coverage and autonomy. A conservative autonomy policy then follows, with broad read access, narrow write access, and closed-loop behaviour only for low-impact actions with strong verification support. This is consistent with the NetOps literature on outage causes and the limits of informal reasoning in complex control planes [52,128–130].

6. Agentic AIOps

AIOps has a richer public literature than NetOps because software services produce abundant observability data and incident work is already mediated through telemetry stores, dashboards, incident histories, and runbooks [26,73,75]. What changes with agentic LLMs is not simply that a system can describe an incident; it is that the system can conduct the investigation as a structured workflow: selecting the next discriminating query, interpreting partial results, maintaining competing hypotheses, and drafting an action proposal tied to evidence, approvals, and rollback [8,10,85,106].

Two background facts from the systems and SRE literature explain why AIOps is a natural home for agents. First, modern production diagnosis is fundamentally query-driven, and good teams treat incident response as repeated hypothesis testing under partial observability [4,98,104]. Second, tracing and analysis stacks in large organisations are designed to support rapid, iterative diagnosis at scale, because static dashboards are not enough [75,78,79,97]. Fig. 6 shows the main roles, their tools, and needed safeguards. Agentic AIOps is practical only when it follows operational routines: tool-driven investigation, not open-ended explanation. Fig. 10 maps the AIOps lifecycle and marks where tool-based RCA and mitigation planning must tie to clear gates and rollback [8,10,32,73,79,98,106].

AIOps invariants. AIOps can be related to NetOps verification by treating mitigation safety as formal conditions over the current service state. Let s_t denote the observed state at time t, including SLOs, error budget, dependency health, replica status, rollout state, and tenant impact. A candidate action a_t is admissible only when the relevant service conditions hold:

$$\phi_{\mathrm{SLO}}(s_t, a_t) \quad \Pr[L_{p99}(s_{t+1}) \le L_{\max}] \ge 1 - \epsilon, \tag{26}$$

$$\phi_{\mathrm{budget}}(s_t, a_t) \quad B_{\mathrm{err}}(t+1) \ge B_{\min}, \tag{27}$$

$$\phi_{\mathrm{blast}}(s_t, a_t) \quad |\mathcal{U}_{\mathrm{affected}}(a_t)| \le \beta_{\max}, \tag{28}$$

$$\phi_{\mathrm{dep}}(s_t, a_t) \quad \mathrm{Reachable}\left(G_{\mathrm{dep}}^{t+1}, S_{\mathrm{critical}}\right) = \mathrm{true}. \tag{29}$$

These conditions cover latency, error budget, blast radius, and dependency health. They are proposed examples of gate clauses, not claims that all reviewed AIOps systems implement probabilistic SLO checking. A benchmark must define the forecast horizon, estimator, uncertainty interval, and threshold for ϕ_{SLO}; deterministic limits remain preferable where available. After an agent suggests an action, the surrounding system checks the applicable service-level requirements before proceeding.

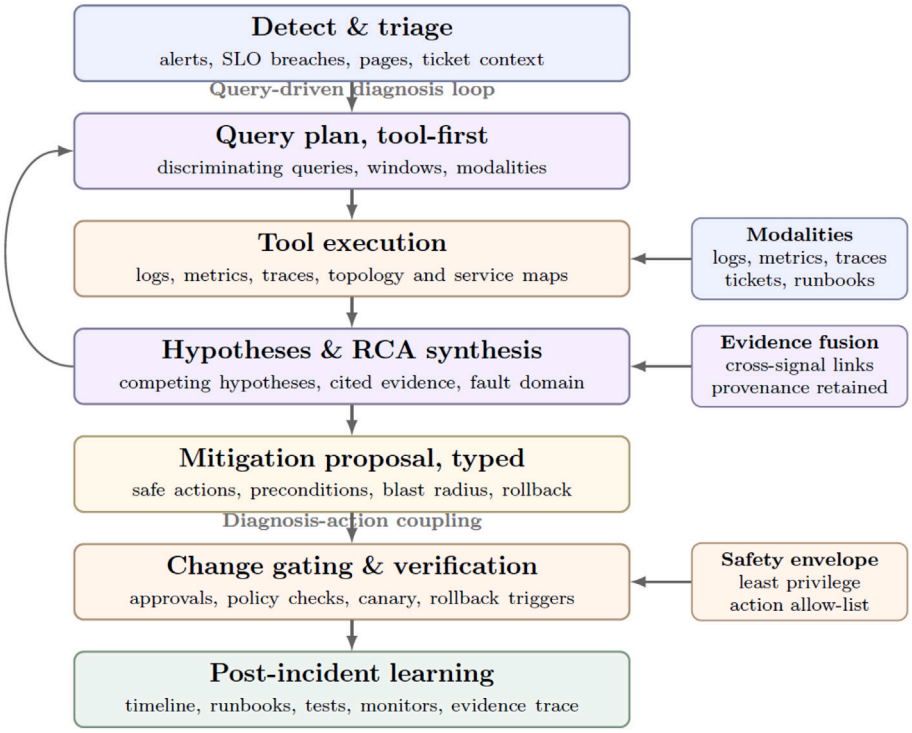


Fig. 10. Workflow view of agentic AIOps. The central loop is query-driven diagnosis: select the next discriminating query, gather evidence through tools, revise competing hypotheses, and prepare mitigation proposals that remain subject to policy checks, approval gates, and rollback-ready verification.

6.1. A causal framing for RCA under partial observability

Root-cause analysis in AIOps is rarely a single-step prediction problem. A more faithful model is iterative ranking and hypothesis testing over a dependency structure under incomplete sensing and changing context [71,77,144,145].

Let the service dependency graph be

$$G_s = (S, \mathcal{E}), \tag{30}$$

where nodes $u \in S$ are components (services, databases, queues, hosts), and edges encode dependencies (RPC, data, or resource coupling). During an incident, the agent obtains multi-modal evidence through tools: metrics, traces, logs, and recent change context. A compact way to express RCA is as a ranking problem:

$$\mathsf{score}(u) = f\big(\mathrm{metrics}, \mathrm{traces}, \mathrm{logs}, \mathrm{change}(u)\big), \tag{31}$$

where the agent returns a top-k list with evidence for each candidate. This framing matches production debugging as an iterative process: issue a discriminating query, refine the candidate set, and choose the next test [71,72,79,98]. Recent systems strengthen this approach by learning or constructing event-based causal graphs that yield interpretable RCA candidates instead of relying only on persuasive correlation [77,144–146]. For agentic AIOps, the architectural lesson is direct: a diagnosis cannot be justified by language alone. The agent must use tools to collect evidence and perform discriminating tests that rule out competing explanations [10,92,106].

6.2. Log understanding and anomaly detection with LLMs

Logs remain the most common operational signal and one of the least forgiving [80,81,147,148]. They are verbose, uneven across systems, and liable to drift as code, dependencies, and deployment habits change. Accordingly, AIOps pipelines rarely consume raw logs directly. They usually introduce at least three intermediate steps: parsing or templating to turn free text into structured events; aggregation to form service-, host-, or request-level sequences and distributions; and detection or scoring to identify anomalies that warrant escalation [80–82,149].

What LLMs change, and what they do not. A sensible way to place LLMs in log-based AIOps is to treat them as semantic adapters, rather than as replacements for classical detectors. Deep sequence models such as DeepLog have shown that event-sequence modelling can detect anomalies without hand-written rules [81]. In real deployments, however, this earlier promise depends on rather plain conditions: good log quality, stable parsing, manageable workload shift, and templates that do not change faster than the detector can learn them [82,147,149].

Practitioner studies sharpen the point. Operators need methods that remain useful under drift, produce evidence-linked outputs, and degrade safely when signals are missing [148]. LLMs can help here by mapping heterogeneous log language into more stable schemas, condensing long bursts into incident-focused summaries, and suggesting the next query or filter needed to test a hypothesis [8,10,85]. This is useful only when the system constrains the output format and preserves provenance [32,112]. If an LLM summary cannot be traced back to log lines and timestamps, it becomes a familiar operational nuisance: neat enough to circulate, and unsafe when someone has to act on it [92,100].

Parsing and benchmarking remain foundational. Because parsing errors cascade into every downstream step, log parsing and its evaluation remain first-class background. LogPai's work on tools and benchmarks for automated log parsing helped standardise evaluation practice and made clear that parser choice can dominate downstream quality [149]. LogHub consolidated commonly used datasets into a shared collection, enabling more reproducible comparisons across systems and domains [82]. For agentic AIOps, the implication is direct: if the agent is trained or evaluated on cleanly parsed logs, but deployed on noisy and drifting templates, autonomy will be miscalibrated [147,148]. This argues for explicitly reporting (i) parsing quality, (ii) drift sensitivity, and (iii) behaviour when parsers disagree [82,149].

6.3. Incident triage and RCA with tool-grounded agents

Incident response has an implicit structure that is easy to state and hard to operationalise: detect, triage, diagnose, mitigate, and learn [73,89]. Agentic systems add value in the gaps between these stages, where humans spend time translating between representations: ticket text to telemetry queries, telemetry results to hypotheses, hypotheses to mitigations, mitigations to change records [8,10,85,106].

From natural language to telemetry queries. A high-leverage capability is converting incident descriptions into concrete telemetry queries. Xpert studies the role of query writing in a large-scale incident workflow and proposes LLM-supported query recommendations to reduce time-to-signal [8]. This is a key agentic building block because it reframes "LLM reasoning" as evidence acquisition. If the system cannot reliably propose the next useful query, it will default to narrative [7,8,99].

Evidence fidelity beats narrative plausibility. Recent LLM-based RCA systems increasingly connect explanations to collected diagnostic evidence. RCACopilot illustrates this approach by combining predefined incident handlers with a language model that predicts root-cause categories and generates explanations [10]. The study evaluates the combined pipeline on real incidents, while the diagnostic collection component has a longer production history than the LLM prediction component; these should not be conflated. The transferable lesson is that an explanation becomes operationally useful when it records what was checked and why [10,92,112].

Agents and tool-augmented RCA. A parallel line of work frames RCA as an autonomous, tool-augmented agent problem. RCAgent uses tool-augmented action trajectories for data collection and analysis, and evaluates across multiple RCA targets (root cause, solution, evidence, ownership) [106]. For AIOps, the point is that RCA quality depends on tool sequencing and stopping rules, not only on language [61,99,106]. An agent that cannot stop when evidence is insufficient will consume budget on noisy queries and eventually "decide" based on fragile cues [32,106].

The updated evidence base separates three forms of RCA evidence. Large-scale task evaluations assess generated causes or mitigations against incident records, including more than 40,000 incidents in Ahmed et al., 100,000 production incidents in the in-context learning study, and cross-lifecycle grounding in X-Lifecycle [35,40,41]. Tool-grounded systems instead evaluate how an agent gathers and tests evidence. Roy et al. study ReAct-style agents with operational retrieval tools, while Flow-of-Action constrains multi-agent RCA with standard operating procedures and an explicit stopping agent [38,45]. OpenRCA provides a separate benchmark object with 335 failures from three enterprise systems and more than 68 GB of logs, metrics, and traces. Its best reported agent configuration resolves only a small fraction of complete cases, which is evidence against equating fluent investigation with reliable RCA [44].

6.4. Mitigation planning under a constrained action envelope

Once an agent proposes mitigations (restart, failover, throttling, config toggle), the system becomes a socio-technical control loop. The failure mode is no longer wrong answer but wrong change at the wrong time [4,90,105,125]. Safe autonomy therefore begins by formalising what actions are even on the table [32,92]. A useful example is queue-aware streaming intrusion detection [150], where online evidence accumulation triggers bounded mitigation rather than open-ended configuration changes. Such designs illustrate the kind of controlled action surface that agentic NetOps and AIOps systems should expose to LLM planners. The update adds two bounded execution studies. Cao et al. evaluate a GPT-4-based agent over 150 Linux administration tasks inside a Docker sandbox, and Sarda et al. generate and execute Ansible remediation playbooks for microservice faults [37,39]. These studies justify classifying some systems as write-limited planners or bounded executors. Their sandboxes, task envelopes, and scripted interfaces remain part of the result, so neither study supports broad closed-loop production autonomy. The distributed nature of modern services constrains AIOps remediation. Restarts, failovers, replica changes, traffic shifts, and rollbacks can involve multiple components, and their safety depends on consistency, replication, coordination, and recovery rules [151,152]. Stateless front-end failovers are generally low-risk. Replaying logs or changing leaders, however, can disrupt ordering, persistence, or client-visible consistency. AIOps systems must therefore account for state, arbitration, and leader-election rules and avoid remedial actions that could violate the required protocols. Let $\mathcal{M}$ be the permitted mitigation set, partitioned into low-risk and high-risk actions:

$$\mathcal{M} = \mathcal{M}^{\text{low}} \cup \mathcal{M}^{\text{high}}, \tag{32}$$

where $\mathcal{M}^{\text{low}}$ includes actions with well-tested rollback paths, while $\mathcal{M}^{\text{high}}$ includes changes that alter correctness assumptions or have long-tailed side effects (schema migrations, broad policy changes, or permanent configuration edits). This organisation-specific partition must be encoded in policy Π, not left to prompt text [32,119]. Mitigation selection can then be stated as a proposed constrained optimisation template:

$$a^\star = \arg\min_{a \in \mathcal{M}} \Big(\mathsf{Impact}(a) + \lambda\, \mathsf{Risk}(a) \Big) \quad \text{s.t} \quad g(a, E, \Pi) = 1, \tag{33}$$

where E is the collected evidence trace and $g(\cdot)$ is the non-bypassable gate that enforces policy, approvals, and required checks. The loss terms and λ are benchmark-specific and require empirical calibration; hard gate failure is not offset by a lower weighted objective. The separation is explicit: the agent proposes, while the surrounding system decides what is permitted and when [32,92,107,108].

6.5. Workflow artefacts as adversarial inputs

AIOps depends heavily on tickets, runbooks, on-call notes, postmortems, and chat transcripts [89]. These artefacts record valuable local practice but remain untrusted input channels: they may be stale, incomplete, or maliciously crafted. Prompt injection and integrity threats therefore belong to the primary design surface rather than to a peripheral "LLM application" checklist [100,101,115,153,154]. Operational artefacts must not directly authorise privileged actions [32,115]. A tool-first design may use them as leads, but must confirm or reject those leads against independent operational evidence before proposing a change [10,106,112]. Evaluations should test this distinction; benchmarks containing only clean, authoritative incident descriptions are likely to overstate safe autonomy [61,99].

6.6. Datasets and benchmarks for agentic AIOps

Progress is limited by proprietary incident corpora, heterogeneous telemetry infrastructures, and weak or ambiguous ground truth [18,155]. Public datasets support reproducibility, but agentic evaluation requires elements they often omit: labelled incidents or adjudicated causal structures, realistic tool interfaces, policy constraints, and operationally meaningful success criteria [32,61,99].

Logs: shared corpora and realism. LogHub makes evaluation repeatable, but that does not guarantee realism [82]. In real systems, log anomaly detection encounters missing events, changing templates, and unclear labels [147]. Practitioner studies suggest benchmarks should reward robustness along with actionable outcomes, not accuracy alone [148]. For agentic AIOps, benchmarks should cover drift, partial observability, noisy metadata, incorrect tickets, and outdated runbooks [99,147,148].

Traces and multi-modality. Trace-based RCA benchmarks expose causal structure more directly. TraceRCA localises microservice root causes using traces and related signals [146]. TADBench targets trace anomaly

Table 5
Task taxonomy for agentic NetOps and AIOps, stated at the tool and action levels. The final column links each task to the contract components most strongly stressed by the evaluation [59,161,162].

Task	Inputs	Tools (surface)	Success criteria (operational)	Contract components stressed
Incident triage & routing	alerts, tickets, short context	ticketing, CMDB/service map, on-call rota [73,85]	correct ownership/team, low time-to-first-meaningful-action, no privacy leaks [84,100]	mainly $\mathcal{R}_k, \mathcal{B}_k$: enough fresh evidence for routing, under time and privacy budgets
Evidence acquisition (querying)	incident description, symptoms	log search, metrics DSL, trace queries [8,163]	discriminating queries early, bounded tool budget, reproducible query trail [8, 161,162]	mainly $\mathcal{T}_k, \mathcal{R}_k, \mathcal{B}_k$: permitted queries, useful evidence, bounded tool cost
Root cause analysis (RCA)	logs/metrics/traces, change history	correlation, dependency graph, repo metadata [10,106,144,145]	correct root cause in top-k, evidence-linked explanation, calibrated uncertainty [10,106,124]	mainly $\mathcal{R}_k, \mathcal{B}_k$: sufficient evidence, calibrated uncertainty, and bounded diagnostic cost
Remediation planning	runbooks, policy, failure domain info	change API wrappers, feature flags, runbook retriever [10,164]	safe plan as diffs/typed steps, explicit preconditions, rollback trigger defined [10,164]	mainly $\mathcal{R}_k, \mathcal{G}_k, \mathcal{U}_k$: evidence-backed plan, policy gates, and rollback-ready steps
Net intent-to-config synthesis	intent, inventory, topology	config generators, compilers, verifiers [1,69,134]	correct diff, invariants hold under failures, ambiguity resolved before synthesis [1,69,134]	mainly $\mathcal{T}_k, \mathcal{G}_k, \mathcal{U}_k$: write-capable tooling, invariant gates, and controlled change artefacts
Change safety & rollout control	proposed diff, risk context	sandbox, canary controller, rollback hooks [11,59,91]	safe intermediate states, bounded blast radius, automatic rollback on guardrail breach [11,91]	mainly $\mathcal{G}_k, \mathcal{U}_k, \mathcal{B}_k$: gate compliance, staged rollout, rollback, blast-radius and latency budgets

detection. This matters when agents must decide whether an anomaly requires escalation or mitigation [156]. Causal and event-graph RCA provide richer ground truth. They show not only which service failed but also which event chain mattered and which tests ruled out alternatives [76,77,144,145].

Agentic benchmark requirements. A practical benchmark acts as an environment:

$$\mathcal{B} = (\mathcal{D}, \mathcal{T}, \Pi, \mathcal{A}, \mathcal{M}), \tag{34}$$

where $\mathcal{D}$ provides multi-modal observations, $\mathcal{T}$ exposes tool calls, Π encodes policy and approvals, $\mathcal{A}$ defines the permitted action envelope, and $\mathcal{M}$ defines metrics including accuracy, time-to-diagnosis, cost, and safety violations. Without $\mathcal{T}$ and Π, evaluation reduces to story quality [20,58,61,99].

Recent IT automation benchmarks use real tasks and interfaces rather than static text alone [61,99]. AIOps evaluation still lacks several elements: tool-call traces, intermediate hypotheses, action-outcome simulation, and penalties for unsafe actions even when the final diagnosis is correct [32,92,155]. As argued in Section 3, safe autonomy can be evaluated only when tools, gates, and action envelopes form part of the benchmark [32,61,99].

Three recent benchmarks illustrate this progression. OpenRCA evaluates long-context, multi-modal diagnosis over enterprise telemetry [44]. AIOpsLab deploys cloud environments, injects faults, exposes telemetry and shell interfaces, and evaluates detection, localisation, RCA, and mitigation [43]. TeleEval-OS covers 13 telecommunications operations-scheduling subtasks spanning ticket creation, resolution, closure, and operational assessment [48]. Their task units and interfaces differ and do not support a single leaderboard. They are better compared by task level, tool surface, available evidence, action envelope, and recovery obligation.

7. Evaluation and benchmarking

Many agentic operations papers state their main claims too loosely. The relevant test is correctness under operational constraints, not correctness alone [147,157–160]. Evaluation should therefore examine the workflow: what evidence was collected, whether the agent stopped appropriately, and whether its actions obeyed policy and rollback rules [59,106,161,162]. Reliable systems leave an auditable trace, behave conservatively under uncertainty, and recover when execution departs from plan [4,11,90,125].

Table 5 sets out the terminology by defining the tasks, their inputs, tool surfaces, operational success criteria, and the contract components most strongly stressed by each task. Table 6 then makes the metrics explicit, and Fig. 11 gives a minimal evaluation report card that every agentic operations paper should include.

7.1. A metrics glossary and a consistent scoring vector

Operational success is multi-objective and should be represented as a reporting vector rather than collapsed into a single aggregate score [161,163,165]. A compact proposed template that covers both diagnosis and action is:

$$\mathbf{m} = \big(\mathsf{RCA@}k,\ \mathsf{MRR},\ \mathsf{Violations},\ \mathsf{RollbackProfile},\ \mathsf{ToolCost},\ \mathsf{Latency}\big). \tag{35}$$

The key is that some dimensions are “hard” constraints (policy violations should be near zero), while others are trade-offs (tool cost versus time-to-signal). The entries are not commensurate: a paper should not compensate for a safety violation by averaging it with higher task accuracy. For hybrid causal–LLM systems, evaluation should also report where computation is spent: causal-graph construction or update, telemetry querying, retrieval, and LLM calls. This distinction matters because a causal model may amortise diagnostic cost across incidents, while an LLM-heavy design may repeatedly consume context and compute for each investigation. Explainability should likewise be scored separately from answer fluency, for example by checking whether the system returns a causal path, the evidence supporting each edge, and the tests that would falsify the proposed cause.

Metric reporting protocol. Every metric should identify (i) the autonomy rung and unit of analysis—incident, proposal, rollout, or trace; (ii) the ground truth or adjudication process; (iii) the unit, observation horizon, and direction of improvement; (iv) whether it is a hard constraint or trade-off; (v) aggregation by mean, median, percentile, or macro/micro averaging; (vi) uncertainty through bootstrap or repeated-run confidence intervals where dependence permits; (vii) important strata such as incident severity, change class, model version, and missing-telemetry condition; and (viii) known gaming paths and missing-data handling. Safety-event counts should be reported both as rates and as absolute numerators and denominators. Rare events require exposure-based denominators and conservative intervals rather than a claim of zero risk.

Metric-specific instantiation and applicability. The protocol above applies to every row in Table 6. MTTD and MTTR use seconds per incident or rollout, with benchmark-defined start and end events, censoring rules, tail percentiles, and uncertainty across independent incidents. RCA@k,

Table 6
Operational metric templates for agentic NetOps/AIOps. Each metric requires a declared rung, ground truth, unit, aggregation rule, direction, and uncertainty analysis; no single metric is sufficient [124,159,160].

Metric	Definition (what to compute)	Why it matters
MTTD (proxy)	Elapsed seconds from a benchmark-defined incident start to the first adjudicated correct triage signal or discriminating query; lower is better	Measures time-to-signal, but requires a fixed start event and adjudicated query utility [8,73]
MTTR (proxy)	Elapsed seconds to verified mitigation in replay or sandbox, or to an approved plan in a human-gated study; lower is better, with the endpoint named	Avoids conflating plan approval, change execution, and restored service [4,11,90]
RCA@k	Indicator that any adjudicated causal contributor appears in the top-k list; report a macro average and the multi-cause matching policy; higher is better	Reflects shortlist inspection while permitting interacting causes and partial labels [76,146]
MRR/nDCG	Ranking quality against an adjudicated relevance ordering for causal contributors or next-step recommendations, aggregated per incident, with higher values better	Requires fixed relevance grades and a candidate-set policy, preventing longer unbounded lists from appearing better [8,106,161]
PolicyViolations	Number and exposure-normalised rate of attempted or executed forbidden tools, scopes, or actions; zero is the hard target	Attempted and executed violations must be separated and reported explicitly [119,163,164]
RollbackProfile	Trigger recall on harmful canaries, false-rollback rate on healthy canaries, completion rate, and trigger-to-recovery latency	No scalar direction is valid: successful detection and poor proposals can both increase raw rollback frequency [11,125]
UnnecessaryActions	Actions beyond an adjudicated minimal sufficient plan, or actions whose paired replay or no-action counterfactual shows no benefit, with lower values better	Requires a declared counterfactual oracle and fixed action granularity, otherwise the count is not identifiable [73,105]
ToolCost	$\sum_i c(\tau_i)$ over tool calls τ_i (tokens, API cost, retries)	Budgets bind autonomy in practice [20,21,163]
Latency	End-to-end wall-clock time (tool time included), reported with percentiles	Determines operational usability [59,166]
Abstention quality	Coverage–risk curve and conditional stop rates under benchmark-labelled sufficient and insufficient evidence	Separates safe caution from paralysis; neither universal abstention nor universal action is rewarded [161,165]
ECE	$\mathrm{ECE} = \sum_b \frac{\lvert B_b \rvert}{n} \left\lvert \mathrm{acc}(B_b) - \mathrm{conf}(B_b) \right\rvert$, using a declared confidence source, binning rule, and outcome label; lower is better	Self-reported verbal confidence is not automatically calibrated; compare it with elicited probabilities or repeated-sample estimates [124]
Drift robustness	Score degradation under controlled drift: missing telemetry, template shifts, stale runbooks	Drift is the default case in operations [147,157,158,160]
CausalPathQuality	Fraction of adjudicated path edges supported by evidence, plus coverage of required disconfirming tests; higher is better.	Distinguishes a checkable causal explanation from plausible narrative [67,68,144].
ComputeBreakdown	Cost split across telemetry queries, causal-graph construction/update, retrieval, and LLM calls, reported in native units and normalised monetary cost where possible.	Prevents improvements from hiding a shift to more expensive repeated reasoning [68,127,163].

MRR, nDCG, and CausalPathQuality use an incident-level adjudicated causal set, relevance grades, or an evidence-edge oracle. They require macro or stratified aggregation and fixed candidate-set rules to prevent gaming through longer lists. PolicyViolations, RollbackProfile, and UnnecessaryActions use proposal or rollout units. Their oracles are the policy decision, paired harmful and healthy canaries, and an adjudicated minimal sufficient action set or paired no-action replay. Violations remain hard constraints rather than compensable utility terms. ToolCost, Latency, and ComputeBreakdown use trace or incident units and are interpreted as lower-is-better only conditional on matched task success. Abstention quality, ECE, and StopScore require independently labelled outcomes, a declared confidence source, and evidence-sufficiency labels, with coverage and missed-action rates reported separately. Drift robustness uses paired clean and perturbed cases and reports score degradation with incident-level uncertainty. Read-side outcome metrics apply from R1 upward, tool and trace metrics from R2 upward, and write, rollout, and rollback metrics only to R3–R4 studies that exercise the corresponding authority. Where the source does not expose the required oracle or unit, the metric is not estimable and should be reported as such.

A proposed stop-decision summary is:

$$\mathsf{StopScore} = \Pr(\text{stop} \mid \text{insufficient evidence}) - \Pr(\text{stop} \mid \text{sufficient evidence}). \tag{36}$$

Its range is $[-1, 1]$, and higher is better only when the benchmark defines evidence-sufficiency labels independently of the agent. It should be reported with conditional rates, confidence intervals, and sensitivity to the sufficiency adjudication; otherwise, the difference can conceal poor performance in one class. Under those conditions, it penalises premature conclusions while rewarding stopping or escalation when evidence is genuinely insufficient [106,161].

7.2. A standard evaluation report card (what every paper should report)

Many results are hard to compare because papers omit the operational interface. Fig. 11 is a minimal report card: datasets and splits, tool surfaces, budgets, gates, trace logging, drift tests, and cost and latency [159–161]. It is intentionally boring, because boring is what lets the field make scientific progress [157,158,160].

7.3. Worked example: scoring an agent trace (process quality is observable)

Let an agent produce a trace $E = (\tau_1, \ldots, \tau_n)$ where each τ_i is a tool call, a retrieved artefact, or a proposed action. A practical trace score decomposes into: (i) discriminating value, (ii) policy compliance, and (iii) budget use [161–163]. One minimal form is:

$$\mathsf{TraceScore}(E) = \sum_{i=1}^{n} \Big(\mathsf{Gain}(\tau_i) - \lambda c(\tau_i) - \mu \mathbb{I}\{\tau_i \text{ violates policy}\} \Big), \tag{37}$$

This is paired with outcome metrics such as RCA@k and the rollback profile. This is a proposed within-benchmark utility, not a validated cross-benchmark score. The benchmark must define and normalise $\mathsf{Gain}(\tau_i)$, for example as reduction in oracle-labelled hypothesis entropy or agreement with a gold evidence trace; fix λ and μ before testing; and publish sensitivity analyses. Hard policy violations should also be reported separately and should not become acceptable merely because accumulated gain is large. The useful property of the template is that policy violations and budget overruns are not hidden behind a fluent final paragraph [161,164].

7.4. Worked benchmark instantiations: one NetOps task and one AIOps task

NetOps change and rollback. Consider 40 repeated sandbox rollouts of an ACL diff against an emulated topology: 20 seeds contain an injected canary regression and 20 are healthy. The benchmark fixes the intended reachability and isolation predicates, the allowed toolset, a maximum of 12 read calls, the canary duration, and a 60-second recovery deadline. If rollback is triggered in 18 harmful cases and 2 healthy cases, the rollback profile reports trigger recall ($18/20 = 0.90$), false-rollback rate ($2/20 = 0.10$), completion among triggered rollbacks, and the distribution of trigger-to-restoration latency. A rollback rate of $20/40 = 0.50$ is not meaningful on its own. Proposal validity, verifier rejection, gate-bypass attempts, intermediate-state violations, and final recovery should

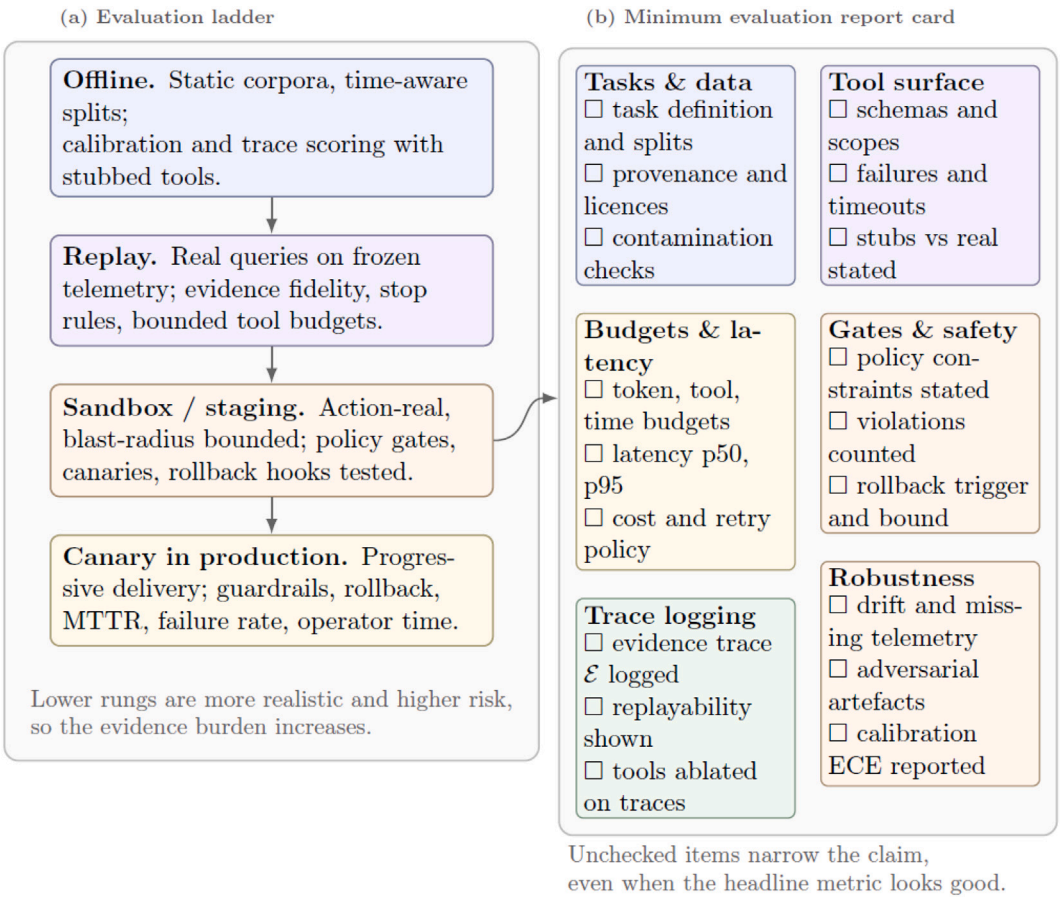


Fig. 11. Evaluation realism and matched reporting burden. The left side orders study environments from offline corpora to canary-in-production evaluation; the right side lists the minimum tasks/data, tool, budget, gate, trace, and robustness information needed for claims at each level. This figure concerns evidence strength, not agent architecture.

be reported separately. Rates should include binomial intervals, and latency should be given by median and tail percentiles across seeds.

AIOps diagnosis and stopping. Consider 100 incident replays from a microservice testbed, with 50 cases containing sufficient evidence for an adjudicated causal set and 50 with one required modality removed. If the agent stops or escalates in 42 insufficient-evidence cases and 6 sufficient-evidence cases, Eq. (36) gives $42/50 - 6/50 = 0.72$. The two conditional rates are also needed to distinguish appropriate caution from unnecessary abstention. Each replay should report RCA@k under a multi-cause matching rule, time to first discriminating query, causal-path evidence precision, tool cost, and abstention coverage. Mitigations are tested only in a sandbox. Correct diagnosis does not authorise live execution. Bootstrap intervals are computed by incident, with repeated stochastic runs nested within each incident. These examples make the templates measurable once the task, oracle, permissions, perturbations, horizon, and aggregation rule are fixed. They define illustrative protocols, not new experimental results.

7.5. Offline evaluation: datasets, metrics, and protocols

Offline evaluation is necessary, but it must be designed to avoid overclaiming [159–161].

(i) Leakage and contamination. Incident corpora contain near-duplicates (recurring alerts, repeated signatures, repeated playbooks). At minimum, splits should be time-aware and duplicate-aware, with overlap checks reported. This matters even more for LLM systems because pretraining contamination can silently inflate benchmark performance, and contamination checks are now part of mainstream LLM evaluation practice [62,159,167].

(ii) Metrics must match operational intent. For RCA, top-k accuracy is useful, but ranking metrics (MRR, nDCG) often match engineering practice better. For action planning, report policy violations, unnecessary actions, and rollback completeness. For confidence gating and abstention, calibration matters, because wrong-but-confident agents create the worst operational incidents [124,161]. LM-PACE provides domain-specific evidence for this requirement by estimating confidence for GPT-4 cloud-incident RCA predictions through retrieval-grounded prompting and calibration [47]. The result should be interpreted within its evaluated incident distribution and predictor family. Expected calibration error is informative only when binning, sample size, confidence definition, and shift conditions are reported, and calibration does not convert an unsupported diagnosis into authority to act.

(iii) Tool interfaces and traces are first-class. Agentic ops evaluation should not treat tools as an implementation detail. Benchmarks for tool use and long-horizon control provide useful templates for reporting tool schemas, trajectories, and budgets [19–21,163,166,168]. A pragmatic protocol is to run offline evaluation in a stubbed tool environment with deterministic responses, then score both outcomes and traces (Eq. 37; Fig. 12) [161,162].

Robustness under drift and partial observability. Operational signals drift continuously. Benchmarking should include stress variants: missing telemetry, delayed telemetry, contradictory signals, and evolving

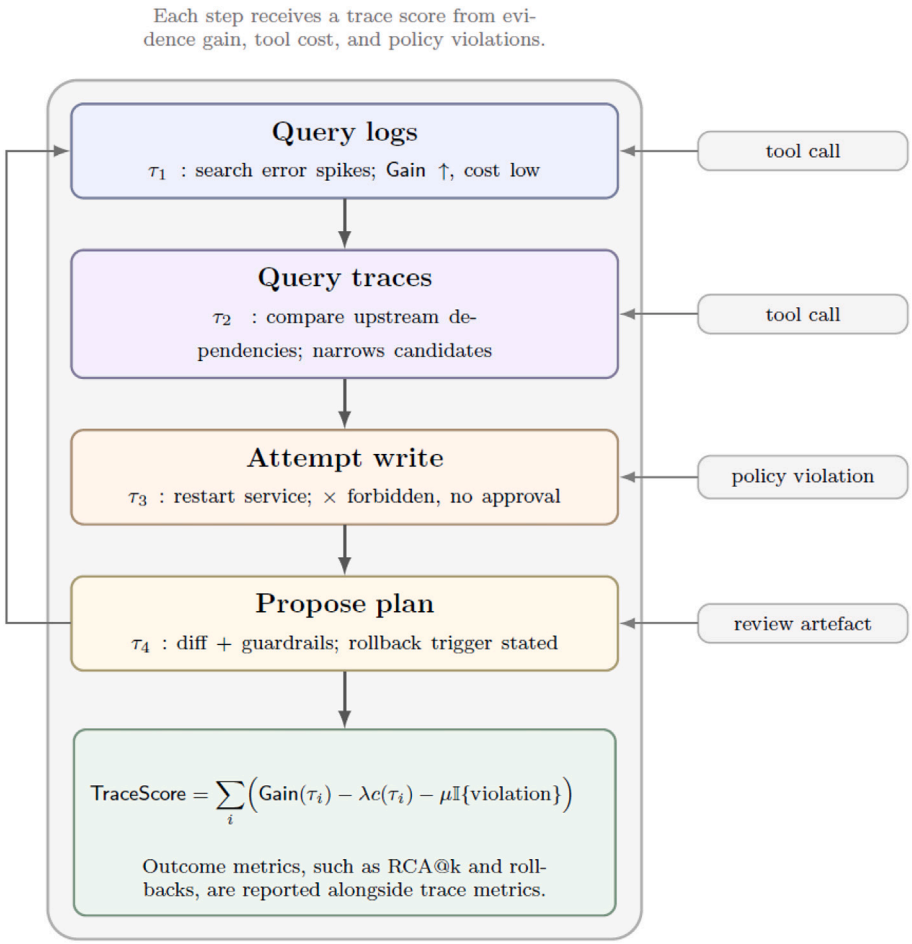


Fig. 12. Process quality is scorable. A trace is evaluated on discriminating value (did the query narrow hypotheses?), policy compliance (did it attempt forbidden actions?), and rollout discipline (were guardrails and rollback triggers specified?).

templates [147,159,160]. If an agent becomes more autonomous under these stressors, it is usually a sign that the evaluation is too forgiving.

LLM-as-a-judge is not enough. LLM graders can help with structure and readability, but they are not a substitute for tool-grounded correctness and safety scoring. Use LLM judges as an auxiliary signal, never as the only metric [160,167].

7.6. Benchmarks and evaluation realism

The benchmark used for an agentic operations system determines the strength of the claim that can be made about it. For NetOps and AIOps, a benchmark is not realistic merely because it contains operational vocabulary. It must expose the agent to the same sequence of constraints that appears in practice: incomplete evidence, bounded tool access, stale or conflicting artefacts, policy checks, rollout constraints, and recovery obligations. For this reason, benchmark realism should be treated as a ladder rather than as a single property.

At the first level, *static question answering* tests whether a model can answer questions about logs, tickets, incidents, configurations, or operating procedures. This is useful for measuring background knowledge and local reasoning, but it does not test whether the agent can gather evidence, choose tools, respect permissions, or avoid unsafe actions. A high score at this level therefore supports only a weak operational claim: the model can produce plausible operational text under a fixed prompt.

At the second level, *tool-use trace replay* evaluates the sequence of tool calls made by the agent. Here the output is not only the final answer, but also the trace: which tools were called, in what order, with which parameters, and whether each call reduced uncertainty. This level is better matched to agentic NetOps and AIOps because many operational tasks are solved by asking the right next query, not by producing the right sentence. Trace replay also makes it possible to score tool budgets, redundant calls, missing evidence, provenance, and stopping behaviour [99,161–163].

At the third level, *sandbox execution* places the agent inside a controlled operational environment. The agent may inspect logs, metrics, traces, tickets, configurations, or topology state, while proposed changes are applied only to a simulator, digital twin, replay environment, or validation pipeline. This level is necessary for testing planner-executor systems because a proposed mitigation may look safe in text but fail under policy, reachability, dependency, or rollout constraints. For NetOps, this level matches configuration analysis, invariant checking, and safe-update testing. For AIOps, it matches frozen incident replay over logs, traces, and change histories.

At the fourth level, *live testbed evaluation* runs the agent against a controlled but active system. This setting is stronger than sandbox replay because telemetry is produced by a running stack, tool responses may change over time, and the agent must operate under partial observability. Such testbeds are useful for evaluating diagnosis under drift, tool failures, noisy alerts, and ambiguous symptoms. They also support fault injection and chaos-style tests, where the benchmark can measure whether the agent detects, investigates, and contains the fault without widening the blast radius [59,164].

At the highest level, *canary and rollback-aware evaluation* tests whether the agent can support bounded execution. The benchmark must specify the approval path, the canary condition, the guardrail metric, the rollback trigger, and the maximum acceptable response time after a violation. This level is essential once $\mathcal{T}_k^{\text{execute}} \neq \emptyset$, because action quality cannot be separated from recovery quality. An agent that applies a correct final change, but cannot detect a bad canary or prepare a rollback, has not yet demonstrated safe autonomy.

ITBench is an important step in this direction because it evaluates agents on real-world-style IT automation scenarios rather than isolated question answering. It contains 94 scenarios across SRE, CISO, and FinOps, and its early results report low success rates for current agents, including 13.8% for SRE tasks and 25.2% for CISO tasks [61]. These results support the central claim of this survey: current agents may look capable in demonstrations, but operational competence requires workflow-level evaluation. The relevant benchmark question is therefore wider than answer correctness. It is whether the agent trace satisfies the contract C_k: the evidence is sufficient, the tool use is bounded, the gates are respected, the rollout is controlled, and the rollback and audit path are complete.

7.7. Online evaluation: sandboxes, canaries, and rollback tests

Online evaluation becomes necessary once an agent proposes actions, because action quality depends on the behaviour of the system being changed [59,125,166]. A useful first stage is incident replay, where telemetry queries are real but actions are simulated. This fits NetOps pre-change verification pipelines [1,69], and AIOps replay over frozen trace and log stores where the agent also leaves an auditable evidence trail [10,106,161].

A later stage is then a controlled rollout. A change may be correct in its final state yet unsafe while it is being applied, so the agent should stage the change, run a canary, validate the result, and only then widen the rollout [11,59,91].

Guardrail-triggered rollback as an evaluation primitive. An evaluation that cannot tell whether the agent would have rolled back is not yet measuring safe autonomy. A stronger protocol specifies the guardrails, the telemetry used to detect breaches, the rollback action, and the allowed response time [125,164]. Rollback-aware learning further indicates that reversibility should be incorporated into the action model [169].

Fault injection and chaos-style tests. Agents should be evaluated on controlled fault scenarios, because real incidents are rare, costly, and unevenly observed. Chaos engineering provides a disciplined way to inject failures and test whether detection and mitigation workflows behave as expected [125].

7.8. Benchmarks for RCA and operations

Against the realism ladder above, existing RCA and operations benchmarks can be read as partial tests of the contract: log and trace datasets mainly stress $\mathcal{R}_k$, tool-use benchmarks stress $\mathcal{T}_k$ and $\mathcal{B}_k$, while sandbox and rollout benchmarks begin to test $\mathcal{G}_k$ and $\mathcal{U}_k$. This shapes the scientific claim itself, as discussed earlier [59,161,162]. For AIOps, useful anchors include shared log corpora and trace-based RCA datasets [82,146]. In NetOps, configuration analysis and verification form the foundation. This ecosystem allows operators to check network invariants and test changes in a repeatable way, which helps ensure that updates do not introduce unexpected problems [1,69,139]. Agentic evaluation introduces two requirements often missing from existing benchmarks. These requirements reflect the practical demands created by an increasingly workflow-centred evaluation. First, the tool interface should be clearly defined. Second, it should be possible to acquire intermediate supervision information, such as by recovering evidence traces from task setup methods [161–163,165]. When these are absent, a plausible narrative can too easily stand in for a sound investigation.

7.9. Cost, latency, and operational usability

In practice, agentic systems often face limitations due to excessive tool invocations, retries, and the use of verbose context prompts. These factors are typically more critical than the performance of any single model invocation. Therefore, evaluations should encompass not only end-to-end latency but also the number of tool invocations, failure rate, token usage, cost estimates, and the time required for operator involvement [59,166,168]. Usability should be judged through the artefacts operators actually use: diffs, evidence trails, guardrail monitors, and rollback plans [4,90,161].

8. Security, privacy, safety, and governance

The introduction of agentic NetOps and AIOps systems has altered the risk profile of NetOps and AIOps. Large language models are no longer limited to generating text; they can now interact with the environment in a more direct way. Agents can now coordinate access to privileged tools, propose and even implement changes, and choose which evidence to rely on. This is a stark contrast to their previous, more passive role. This creates a classic confused-deputy problem: an attacker need not compromise the tool if the attacker can influence an authorised agent to misuse it [60,101,119,170]. A sound engineering response begins with architecture. This choice fixes how the system will behave in a live setting and sets the conditions for later hardening. In practice, operational artefacts should be treated as potentially adversarial from the outset. The system should enforce least privilege at every tool boundary, and all consequential operations should pass through independent approval gates that the model cannot bypass. As complexity increases, these controls reduce the risk that a plausible instruction becomes an unsafe action [102,171,172].

Fig. 13 presents the trust boundaries explicitly. Table 7 maps threats to controls and measurable tests. Section 8.7 gives a concrete evaluation protocol for security claims in agentic operations systems. Security is treated here as a set of enforceable controls and falsifiable tests, rather than as a catalogue of possible hazards [60,170].

8.1. Attacker model, assets, and trust boundaries

A usable threat model for agentic operations should name assets, attacker capabilities, and the concrete surfaces where those capabilities apply [60,101].

Assets at risk. Beyond confidentiality of secrets in logs and tickets, the primary assets are: (i) *integrity of operational truth* (what is happening in the system), (ii) *integrity of changes* (what was applied, where, and why), and (iii) *availability of the workflow* (can on-call staff still diagnose and act). In NetOps, a wrong config or unsafe rollout can create wide blast-radius outages. In AIOps, the more common failure is accelerating a bad mitigation, or delaying a correct mitigation through repeated refusal or mis-triage [103,114].

Attacker capabilities and goals. We model an attacker as the tuple

$$\Theta_{\text{adv}} = (C, \mathcal{G}, S), \tag{38}$$

where C is the set of capabilities, $\mathcal{G}$ is the set of goals, and S is the set of reachable surfaces. Capabilities commonly include: (1) writing or influencing text that enters prompts (tickets, chat, wiki pages), (2) inserting or modifying documents in retrieval stores (poisoning or jamming), (3) influencing telemetry strings (user-controlled payloads that appear in logs), and (4) inducing unsafe tool use by shaping the agent's context. Goals include unsafe action, denial of workflow (refusal storms), and data exfiltration. These are now well evidenced in prompt-injection and RAG-security work [101,113,114,170,173].

Trust boundaries in agentic ops. Operationally, it helps to split the system into four zones:

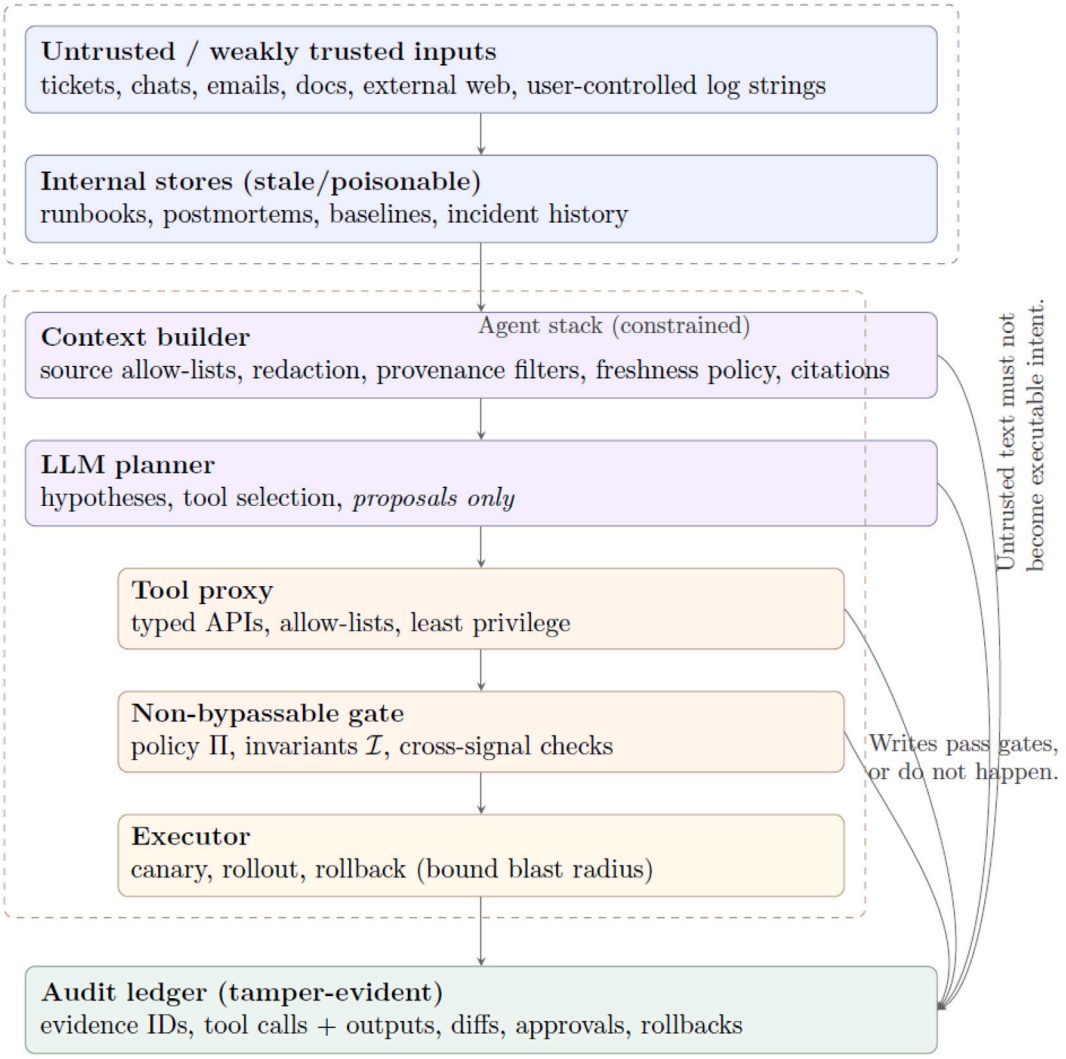


Fig. 13. Adversarial trust-boundary view. Untrusted artefacts can inject instructions or distort evidence; least-privilege tools, policy and invariant gates, cross-signal integrity checks, and audit trails constrain propagation into action. This figure models attack paths, unlike the operational workflow and architecture figures.

1. **Untrusted inputs:** tickets, external docs, user-influenced log lines, chat transcripts.
2. **Semi-trusted evidence stores:** internal runbooks, postmortems, baselines; high value, but stale or poisonable.
3. **Privileged tools:** change APIs, controller interfaces, restart and drain actions, ticketing actions.
4. **Verification and governance:** policy-as-code, validators, simulators, approvals, tamper-evident auditing.

The design requirement is simple: zone (4) must be non-bypassable by the model [60,119].

The scarcity of domain-specific security evaluations is itself material. AIOpsDoom directly tests telemetry manipulation in an operational-agent setting, while much of the remaining evidence is transferred from general tool-integrated agents. For example, Gasmi et al. compare function-calling and Model Context Protocol deployments across 3250 attack scenarios and show that architectural choice changes the attack surface [49,103]. This supports explicit testing of both model-facing and system-facing attack paths. It does not supply a NetOps or AIOps production-safety estimate.

8.2. Threats mapped to controls and measurable tests

Table 7 links the most common threat classes to concrete controls and to tests that can be run during evaluation.

8.3. Telemetry integrity via cross-signal consistency

In operations, failures of integrity often matter more than failures of confidentiality. An agent may still act wrongly, even when it is grounded, if the evidence on which it relies is false or incomplete [103]. Telemetry provides evidence, but it does not offer certainty. The state observed by an agent can be skewed due to a partial view [75,147,157,158]. In adversarial settings, compromised devices may inject misleading logs, counters, or protocol events [100,103]. Agentic systems should therefore attach provenance, coverage, and uncertainty to telemetry before using it for diagnosis or mitigation.

A basic safeguard is to check whether metrics, traces, and logs tell a consistent story. Let $\phi_m = \Phi_m(o^{\text{metrics}})$, $\phi_t = \Phi_t(o^{\text{traces}})$, and $\phi_\ell = \Phi_\ell(o^{\text{logs}})$ denote the corresponding feature views.

$$\text{Consistent} = \mathbf{1}\{d(\phi_m, \phi_t, \phi_\ell) \le \delta\}. \tag{39}$$

Here, $d(\cdot)$ may be a maximum normalised discrepancy or a learned disagreement score, and δ is fixed on clean validation data before adversarial testing. This proposed check does not establish truth: correlated sensors may agree and still be wrong. It is an integrity trigger for additional confirmation or reduced autonomy when modalities disagree [103].

Table 7
Threat-to-control matrix for agentic operations. Controls must be enforceable at the tool boundary, and they should come with measurable tests (attack success, unsafe tool calls, bypass attempts, and refusal storms).

Threat class	Typical surface and goal	Primary controls (enforceable)	Measurable tests
Prompt injection (direct/indirect)	Injected instructions in tickets, runbooks, or chats that steer tool use or bypass policy	Instruction–data separation; context-builder sanitisation; tool allow-lists; non-bypassable gates	Attack success rate; unsafe tool-call rate; bypass attempts blocked [101,102,170–173]
RAG poisoning (targeted)	Corrupt runbook snippets or "known fix" documents that induce a wrong change proposal	Provenance and allow-lists; signed or reviewed documents; retrieval filters by owner and freshness; evidence citation	Targeted answer-flip rate; source-trust violation rate; retrieval provenance coverage [113]
RAG jamming/denial	Blocker documents induce refusal loops, slow the workflow, or stop incident handling altogether	Refusal budget; fallback retrieval; document-level risk scoring; safe degraded mode	Refusal-storm rate; time-to-first-meaningful-action; false-positive refusal rate [114]
Telemetry manipulation (integrity)	Single-channel dishonest signals steer diagnosis and mitigation	Cross-signal consistency checks; independent measurements; "trust but verify" for each claim	Cross-signal inconsistency detection rate; wrong-mitigation rate under adversarial signals [103]
Tool misuse and excessive agency	Wrong tool, wrong arguments, unsafe sequencing, or broad blast radius	Least privilege by tool scope; typed tools; staged execution; approvals for writes	Forbidden-call rate; argument-safety violations; rollback completeness and timeliness [60]
Data exfiltration	Prompt or tool output coerces disclosure of secrets or memorised data	Pre-model redaction; role-based retrieval; output filtering; tight audit on data access	Secret-leak rate; policy violations on data access; retention and deletion compliance [174–176]

8.4. Non-bypassable policy gates and the propose–commit split

Every write action should pass through a gate that the model cannot bypass:

$$g(a, E, \Pi) = \mathbf{1}\{\Pi(a) = 1 \wedge \mathcal{I}(x \oplus a) = 1 \wedge \mathsf{Integrity}(E) = 1\}. \tag{40}$$

Here, a is the proposed action or diff, E is the evidence trace, Π is the access and change policy, $\mathcal{I}(x \oplus a)$ checks the candidate system state, and $\mathsf{Integrity}(E)$ checks evidence provenance and consistency. This distinction is important: state invariants are not functions of an evidence trace. The model may propose a change, but an independently enforced gate decides whether execution is permitted. The gate's assurance remains limited to the scope and soundness of its policy, invariant, and integrity checks.

8.5. Privacy, retention, and compliance as system properties

Operational data frequently includes secrets, customer identifiers, architectural information, location traces, traffic patterns, and service-usage metadata. Agentic systems heighten exposure risks, as such data may be incorporated into retrieval contexts, tool outputs, memory, and audit logs [175,176]. Even anonymized telemetry can remain linkable through rare-event patterns [177].

Accordingly, privacy controls must be enforced prior to model access. These include data minimisation, role- and incident-scoped retrieval, tenant isolation, secret scanning, redaction, retention limits, and auditable access justifications [31]. Post-generation filtering is inadequate because the model has already processed sensitive information [175]. As models may reproduce memorised content, operational agents should, by default, avoid persistent memory and treat each incident as a distinct retention boundary [174–176].

8.6. Governance as artefacts: approvals and audit integrity

Governance is most effective when expressed as enforceable artefacts within the operating process, rather than policy text alone. Policy checks, approvals, and audit records should identify the on-call operator, the authority for each affected domain, mandatory escalation points, and post-incident responsibility. Human approval may fail through fatigue, automation bias, misplaced trust, time pressure, or unclear ownership [23,24,105,178]. Higher-autonomy systems should therefore expose the approval path, the evidence and uncertainty under review, and avoid repeated low-information requests that promote routine approval.

Human performance across autonomy rungs. At the copilot and analyst rungs, evaluation should determine whether the agent reduces search effort without narrowing attention too early. Junior operators may benefit from procedural guidance but are less able to detect plausible errors. Experienced operators may detect subtler errors, although verbose explanations increase interruption cost. At the planner–executor rung, approval depends on whether $\mathcal{R}$ exposes change scope, failed checks, uncertainty, canary conditions, and rollback triggers under realistic time pressure. Closed-loop operation adds review fatigue, loss of situation awareness, and delayed intervention after long periods of correct operation.

Deskilling, handover, and responsibility. Longitudinal studies should examine whether routine delegation weakens diagnostic skill, knowledge of system dependencies, or the ability to operate during agent failure. Periodic manual exercises, replay-based training, and audits requiring reconstruction of causal and change rationales may limit this effect. Handover quality should be judged by whether another operator or team can recover the current hypothesis, unresolved uncertainty, actions taken, ownership boundary, and safe next step. Cross-team incidents also require an authority matrix because networking, security, application, and infrastructure teams may control different tools and policies. Acceptance, modification, rejection, override, and emergency bypass should be logged separately. Responsibility remains with the deployment owner, including the authority granted, gate design, monitoring, and recovery.

Status of the human-factors evidence. The update identifies human evaluation and operator confirmation, but no study whose primary contribution is longitudinal human-factors evaluation of an operational LLM agent. Incident-owner assessments in large cloud RCA studies examine perceived correctness or readability, while LUMI requires operator confirmation of interpreted intent [35,40,41,46]. These observations do not measure automation bias, review fatigue, deskilling, handover quality, or accountability over sustained use. The human-factors agenda therefore remains weakly supported by direct evidence.

Audit trails that support forensics. For agentic ops, the minimum audit record includes: policy version, retrieved evidence identifiers and provenance, tool calls with arguments and outputs, proposed and executed diffs, and verification results that permitted action. If audit logs are not integrity-protected, post-incident accountability collapses. Tamper-evident logging is a natural fit for this requirement [126].

8.7. Security evaluation protocols

Security claims should be evaluated with explicit attacker inputs and explicit tool surfaces, not only with benign accuracy tests. A practical protocol is: (i) define the permitted tool set and policy Π, (ii) define

attacker surfaces S (tickets, docs, telemetry strings), (iii) run injected and poisoned variants alongside clean variants, and (iv) report both task outcomes and security outcomes [60,170,175,176].

We recommend reporting at least: attack success rate, unsafe tool-call rate, bypass attempts blocked, refusal-storm rate (false positives that halt the workflow), and cost/latency impact under attack [60,114,176].

9. Open problems and research agenda

The field of agentic NetOps and AIOps sits at an awkward intersection. On one side, the tool surfaces (telemetry queries, configuration validators, deployment APIs, ticketing systems) are mature and operationally meaningful. On the other side, LLM agents are still difficult to bound, hard to evaluate under realistic drift, and easy to mislead when they treat untrusted text as instructions [179–181]. This section focuses on unresolved problems and transforms them into contract satisfaction questions.

Priority and test design. The agenda follows the dependencies in the available evidence. *Immediate priorities* are benchmark design and reporting. Existing systems can already support better evidence if studies declare the autonomy rung, expose tool interfaces and traces, report model configuration, use matched clean and adversarial cases, and distinguish hard safety constraints from performance trade-offs. Baselines should include a non-agentic method, an LLM without tools, and, where applicable, the same agent without the control under study. *Medium-term priorities* are architectural: compositional safety, calibrated stopping, causal–LLM interfaces, non-bypassable policy enforcement, and verification of write-side actions. *Long-term priorities* concern sustained operation, including cross-team accountability, retention of operator skill, interoperable audit records, calibration under drift, recovery from correlated failures, and governance of repeated autonomous action.

Each study should specify the NetOps or AIOps domain, operational environment or dataset, baselines, unit of analysis, autonomy rung, and measurable success and failure criteria. It should pose a testable question and, where meaningful, compare a non-agentic method, an LLM without tools, and the same agent without the control under study. The main empirical question is whether reported benefits and safeguards survive system change, partial observability, tool failure, and adversarial input.

9.1. Problem framing: autonomy levels and operational assurance contracts

A recurring mistake is to treat autonomy as binary. In practice, operations teams already use graded autonomy through access control, approval procedures, change windows, and progressive rollouts. A deployable research agenda should therefore treat autonomy as an explicit variable and evaluate systems at several rungs. General agent frameworks distinguish reasoning from action, but in operational settings this separation must be enforceable and auditable, rather than being stated only in prompts [13,14,179,181].

This survey expresses this requirement through the assurance contract C_k introduced in Section 4. Future work should specify the autonomy rung, permitted tool surface $\mathcal{T}_k$, required evidence $\mathcal{R}_k$, gates $\mathcal{G}_k$, rollout and audit protocol $\mathcal{U}_k$, and budget constraints $\mathcal{B}_k$. The central question is therefore whether an agent's execution trace satisfies this contract under drift, uncertainty, and adversarial inputs. Deployable autonomy can then be assessed directly through missing evidence, budget overruns, gate violations, unsafe actions, and incomplete audit trails [182,183].

9.2. Verification, guarantees, and compositional safety across layers and time

The central technical challenge for safe autonomy is that operational properties are layered and time-dependent. A NetOps change can satisfy final-state reachability and still create transient loops or black holes during rollout. An AIOps mitigation may reduce the error rate while still violating tail-latency SLOs and triggering cascading retries. The research problem is therefore not only verification, but *compositional safety* across layers and across the rollout timeline.

Network-level invariants remain the clearest foothold. NetOps has a strong tradition of proactive verification and safe-update reasoning. Batfish and Minesweeper show how real configurations can be analysed against protocol semantics and invariants before deployment [1,69]. Complementary systems emphasise fast online detection of policy violations, together with scalable reasoning about forwarding behaviour and network changes [2,3,120]. Work on consistent updates formalises the central risk: correct end states can still be reached through unsafe transient states, so deployment must be treated as a protocol [11].

Service-level safety needs testable operational semantics. For AIOps, safety targets are naturally expressed as SLO constraints, error budgets, and blast-radius limits. They often require monitored rollouts and realistic failure scenarios. Experience with large-scale systems shows that production failure modes are subtle, and validation must include representative workloads and fault patterns [184,185]. Benchmark suites such as DeathStarBench provide reproducible microservice workloads that can support end-to-end evaluation of diagnosis and mitigation policies under stress [186].

Compositional safety in a shared invariant interface. To avoid re-stating definitions already introduced in the assurance sections, the research agenda reuses the same state-invariant and rollout-safety semantics. The compositional target combines network-state constraints, service-level constraints, and rollout safety across intermediate states. A study should therefore report pre-change verification, rollout monitoring, and how often each layer catches a violation that the others miss [1,11].

9.3. Benchmark realism and reproducibility for contract-based evaluation

Agentic evaluation becomes weak when reduced to narrative scoring. Benchmarks should record tool use, intermediate hypotheses, budgets, and safety constraints. Without such records, disciplined investigation cannot be distinguished from plausible guessing. General agent benchmarks capture multi-step interaction and tool use, but operational settings require stricter safety and audit criteria [19,55,58,187]. SWE-bench makes a related point: evaluation changes when tasks involve real artefacts and operational constraints rather than isolated question answering [62,187].

A contract-based benchmark can score each instance based on whether $\tau \models C_k$, with partial credit for satisfied clauses. This is consistent with established evaluation practice, where robustness, calibration, and efficiency matter in high-stakes deployments [127,182,183].

9.4. Hybrid causal–LLM operations

An important open problem is how to combine causal diagnostic models with LLM agents without weakening either. Causal models provide structured explanations, reduce the hypothesis space, and lower repeated diagnostic costs once a dependency graph is available [66–68,188]. LLM agents can coordinate tools, interpret operator intent, summarise evidence, and draft mitigations for review [8,10,61,99,106]. A useful division of labour is therefore to let causal inference narrow and explain the fault space, while the LLM gathers evidence, interacts with operators, and prepares bounded mitigation proposals [68,106,188].

Several questions remain. Causal graphs must remain valid under topology, workload, and service drift [147,157,158]. Agents also need criteria for deciding when causal evidence is sufficient to support a mitigation proposal [67,68,188]. Evaluation should therefore extend beyond root-cause ranking to the causal path, supporting evidence, and cost of reaching the diagnosis [61,66–68,99,127,163].

Table 8
Prioritised research questions for agentic NetOps/AIOps. The surrounding text assigns immediate, medium-term, or long-term horizons; each row pairs a contract-aligned question with a dominant failure mode and a measurable evaluation handle.

Category	Research question (contract view)	Dominant failure mode	Evaluation handle
Safe autonomy & guarantees	How to choose $(\mathcal{R}_k, \mathcal{G}_k, \mathcal{B}_k)$ so that $\tau \vDash C_k$ is achievable yet non-trivial?	Refusal storms or over-permissive execution	$\Pr[\tau \vDash C_k]$, gate-violation rate, refusal rate under missing telemetry [94,134,181]
Compositional safety	How to bind $\mathcal{I}_{\text{net}}, \mathcal{I}_{\text{slo}}, \mathcal{I}_{\text{rollout}}$ into $\mathcal{U}_k$ and make it non-bypassable?	Unsafe transient states despite a "correct" final state	Pre-change vs. runtime catch rates; false-assurance rate [11]
Benchmark realism	What benchmark artefacts are minimally sufficient to score traces, not only outcomes?	Offline over-claim; brittle tool use under drift or budgets	Trace metrics (tool efficiency, stop quality) + outcome metrics under drift variants [19,55,58,182,187]
Security & integrity	How to reduce confused-deputy tool misuse under injection and poisoning attacks?	Unsafe tool call that appears tool-grounded	Attack success rate, unsafe tool-call rate, bypass attempts, over-blocking rate [102,179,180,189–191]
Human factors	What review bundles and uncertainty signals improve intervention quality?	Over-trust or under-trust; poor handover	Handover success, intervention appropriateness, time-to-key-evidence [104,105,182]
Interoperability	What minimal schema set enables replay, auditing, and comparison across stacks?	Bespoke systems that cannot be replayed consistently	Schema coverage, portability across back-ends, replay fidelity [26,27,29,30]
Continual ops & cost	How to maintain safety and calibration as tools, signals, and prompts drift?	Silent degradation; rising tool cost; confident wrong actions	Safety trends over time, drift-trigger accuracy, cost/latency per incident [31,127,183]

9.5. Security and integrity as contract clauses, not add-ons

The main security risk in agentic operations is not privacy alone, but confused-deputy behaviour. Untrusted tickets, runbooks, or telemetry may influence an agent with access to privileged tools. Prompt injection and tool hijacking have proved effective in tool-integrated systems, while prompt-only defences remain fragile [154,179,180,189,190]. Evidence poisoning poses a related threat to retrieval and telemetry pipelines. Provenance, allow-lists, and integrity checks must therefore form part of the control plane [191].

Structured query interfaces and strict separation of untrusted input are useful safeguards, although difficult to enforce across realistic tool chains [102,181,192]. In the contract model, these controls belong in $\mathcal{G}_k$: non-bypassable gates, provenance checks, audit integrity, and explicit bounds on unsafe tool calls, bypass attempts, and excessive blocking.

9.6. Human factors, accountability, and organisational fit

Agentic operations will fail in practice if they ignore how incident response actually works. On-call work is collaborative, interrupt-driven, and full of handoffs. This makes artefacts central: evidence bundles, proposed diffs, checks, ownership records, and rollback triggers. Empirical studies show that coordination and information foraging dominate real incident work [104], while trust in automation remains contextual and can be weakened by inconsistent or opaque behaviour [105]. In contract terms, the question is which $\mathcal{G}_k$ artefacts support fast, correct human intervention and post-incident learning, rather than merely producing persuasive prose [182].

9.7. Standardisation and minimal interoperable schemas

Interoperability is an enabling constraint. Without shared schemas for tool calls, evidence pointers, diffs, approvals, rollbacks, and audit logs, systems remain bespoke and results remain hard to compare.

There is a practical opportunity to align agent evidence collection with OpenTelemetry as a common substrate for logs, metrics, and traces [26]. For networking management, intent and telemetry standards and typed management APIs (such as gNMI) provide scaffolding for safe, auditable tool calls [27–29]. These do not solve agent safety, but they reduce accidental diversity so that traces can be replayed and compared [30].

Minimal schema set. A single universal standard is unlikely in the near term. A more practical route is a small set of interoperable schemas that preserve local system design while making traces checkable, replayable, and auditable.

- **Tool call:** tool name, version, typed arguments, timestamp, result pointer, and error code.
- **Evidence pointer:** source ID, provenance, trust tier, freshness timestamp, and citation anchors such as query IDs, offsets, or row IDs.
- **Diff:** structured change representation, including configuration diffs, feature-flag toggles, or deployment patches, with scope and target inventory.
- **Approval:** approver role, decision, and reviewed bundle, including the diff, checks, and rollout plan.
- **Rollback:** triggers, time bounds, rollback actions, and post-rollback verification queries.
- **Audit log:** hash-chained events linking prompts, policies, retrieved evidence, tool calls, diffs, approvals, and outcomes.

Together, these schemas provide the minimum structure needed to assess $\tau \vDash C_k$ and replay traces in reference sandboxes [30].

9.8. Continual operation: drift, learning, governance, and cost

Agentic NetOps and AIOps systems should be considered as a continuous process, not as a one-time deployment. Models and prompts can age, tool APIs may change, disappear, or acquire new behaviour, the signals underneath the system also drift, and all this sometimes happens quietly. Agent systems should therefore be managed as operational services, with routine evaluation, and where needed, retraining or recalibration. Governance frameworks such as the NIST AI Risk Management Framework provide a useful vocabulary for mapping risks, measuring performance, and monitoring systems over time. This vocabulary fits well with contract-based deployment and phased release practices, since both require the system boundary and its obligations to be made explicit [31].

General evaluation frameworks that track several dimensions and publish comparable artefacts are a useful starting point. For NetOps and AIOps, however, the record needs to be more operationally specific. An agent card specifies tool scopes $\mathcal{T}_k$, contract clauses, approval rules, rollback conditions, and monitors safety performance over time [127,182,183]. Table 8 links each research question to a corresponding failure mode and evaluation method. In all instances, empirical evidence is required; assertions of increased autonomy must clearly define permitted actions and demonstrate that the system fulfills these criteria.

10. Conclusion

In NetOps and AIOps, agents are no longer confined to producing operational documentation. They are increasingly involved in operational work itself, where reliability, security, and accountability remain active

constraints. In network and cloud operations, this work follows a controlled path from evidence to action. An agent may read artefacts, query tools, form a diagnosis, suggest a mitigation, and, in limited cases, carry it out. These steps need to remain separate. The evidence synthesised here suggests that operational reliability depends critically on the control system surrounding the agent, particularly as authority approaches the write boundary. Safer agent design therefore depends on clear authority boundaries, restricted tool access, and checks that remain visible during ordinary operation.

The evidence synthesis must nevertheless be read at the correct level. The six analytical pillars describe the full LLM-enabled NetOps and AIOps system, while the four evidential statuses constrain the claims that each record may support. Within the 28 directly audited LLM-facing records, only 13 are operational systems and none demonstrates broad closed-loop control. Evidence is strongest for read-oriented assistance and tool-grounded diagnosis and remains substantially thinner for write-capable operation, domain-specific security, and longitudinal human factors. Our qualitative synthesis therefore indicates, rather than statistically estimates, a capability–assurance gap.

Autonomy is best treated as a set of operational commitments rather than a single capability. Read-only support, evidence collection, proposal generation, limited execution, and self-healing expose different tools, risks, and duties of accountability. The sharpest distinction appears at the write boundary. An agent may recommend a configuration change, mitigation, or policy update, but it should not apply that change to a live system without checks that do not depend on the model. Small operational errors can spread. A misplaced rule or premature deployment may widen into service failure.

NetOps already has a tradition of pre-change checking, including reachability, isolation, loop freedom, and update safety. AIOps has a parallel diagnostic requirement: a proposed cause must be tied to dependencies and observed facts. Evaluation should therefore examine the workflow as well as the final answer. It should show what was investigated, which tools were used, what evidence was missing, and whether the causal path can be inspected. Policy violations, latency, rollback behaviour, robustness, and auditability matter because they describe how the system behaves under operational pressure. A sound evaluation path should move from offline, time-aware datasets to replay testing, sandbox testing, and controlled trials in live environments.

The mathematical constructs and several workflow scores in this survey are proposed conceptual or reporting templates, not theorem claims or universal metrics. Their quantities require benchmark-specific task definitions, independent oracles, calibrated thresholds, declared aggregation rules, and uncertainty reporting.

Security and governance follow the same reasoning. Operational artefacts contain local knowledge, but may be stale, sensitive, incomplete, or adversarially influenced. Trust must therefore be part of the workflow from the outset. Prompt injection, poisoned retrieval, tampered telemetry, excessive tool use, and refusal storms all indicate weak control over evidence, access, action, or recovery.

Accountability follows directly. An agent operating near a control surface must leave actions that can be explained after the event. Otherwise, it may remain useful, but it is difficult to govern. This also limits current evidence. Public benchmarks rarely represent the drift, partial observability, stale documentation, tool failure, and adversarial input found in real operations. Claims for closed-loop autonomy in NetOps and AIOps should therefore remain limited. Read-only assistance is already useful, and narrow self-healing may be feasible. Broad self-healing in large, variable systems remains substantially harder.

Human approval does not by itself resolve this problem. Automation bias, review fatigue, rubber-stamping, deskilling, and poor cross-team handover may weaken an otherwise sound approval process. Evaluation should therefore record who accepted, modified, rejected, or overrode a proposal, and whether another operator can safely continue the incident.

This survey therefore adopts a contract-based model of autonomy. At each level, the system should specify permitted tools, required evidence, mandatory checks, and the provisions for rollback and audit. Greater autonomy requires a dependable connection between evidence, permitted action, and safe recovery. Without it, autonomy demonstrated under controlled conditions may fail in operational use.

The conclusion is conditional. Operational assurance does not necessarily increase with operational authority. It must be designed, reported, and tested at each rung, and claims should not exceed the evidence available at that level of authority.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors thank Hamed Haddadi for comments on earlier drafts. AI-assisted tools were used for language editing and structural refinement. The authors remain responsible for the final text.

Appendix A. Supplementary data

Supplementary data for this article can be found online at doi: 10.1016/j.cosrev.2026.101075.

Data availability

Bilal, Muhammad; Crowcroft, Jon; Wang, Ruizhi; Xu, Xiaolong; Dustdar, Schahram (2026), “LLM NetOps/AIOps Six-Pillar Evidence Audit Dataset”, Mendeley Data, V3, doi: 10.17632/2p5ppxzy4s.3.

References

[1] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, T. Millstein, A general approach to network configuration analysis, in: Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI ’15), USENIX Association, 2015, pp. 469–483.

[2] P. Kazemian, G. Varghese, N. McKeown, Header space analysis: static checking for networks, in: Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2012.

[3] A. Khurshid, X. Zou, W. Zhou, M. Caesar, P.B. Godfrey, VeriFlow: verifying network-wide invariants in real time, in: 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), USENIX Association, 2013, pp. 15–27.

[4] N.R. Murphy, B. Beyer, C. Jones, J. Petoff, Site Reliability Engineering: How Google Runs Production Systems, O’Reilly Media, 2016.

[5] N. Forsgren, J. Humble, G. Kim, Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations, IT Revolution Press, 2018.

[6] J. Kephart, D. Chess, The vision of autonomic computing, Computer 36 (2003) 41–50, https://doi.org/10.1109/MC.2003.1160055

[7] Z. Yu, M. Ma, C. Zhang, S. Qin, Y. Kang, C. Bansal, S. Rajmohan, Y. Dang, C. Pei, D. Pei, Q. Lin, D. Zhang, MonitorAssistant: simplifying cloud service monitoring via large language models, in: Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, ACM, 2024, pp. 38–49, https://doi.org/10.1145/3663529.3663826

[8] Y. Jiang, C. Zhang, S. He, Z. Yang, M. Ma, S. Qin, Y. Kang, Y. Dang, S. Rajmohan, Q. Lin, D. Zhang, Xpert: empowering incident management with query recommendations via large language models, in: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, Association for Computing Machinery, 2024, https://doi.org/10.1145/3597503.3639081

[9] H. Wang, A. Abhashkumar, C. Lin, T. Zhang, X. Gu, N. Ma, C. Wu, S. Liu, W. Zhou, Y. Dong, W. Jiang, Y. Wang, NetAssistant: dialogue based network diagnosis in data center networks, in: Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI ’24), USENIX Association, 2024.

[10] 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, T. Xu, Automatic root cause analysis via large language models for cloud incidents, in: Proceedings of the 19th European Conference on Computer Systems (EuroSys ’24), ACM, 2024, pp. 674–688, https://doi.org/10.1145/3627703.3629553

[11] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, D. Walker, Abstractions for network update, in: Proceedings of the ACM SIGCOMM 2012 Conference, ACM, 2012, pp. 323–334, https://doi.org/10.1145/2342356.2342427

[12] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, D. Kiela, Retrieval-augmented generation for knowledge-intensive NLP tasks, in: Advances in Neural Information Processing Systems (NeurIPS 2020), 2020.

[13] T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, T. Scialom, Toolformer: language models can teach themselves to use tools, in: Proceedings of the 37th International Conference on Neural Information Processing Systems, Curran Associates Inc., Red Hook, NY, USA, 2023.
[14] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K.R. Narasimhan, Y. Cao, ReAct: synergizing reasoning and acting in language models, in: Proceedings of the International Conference on Learning Representations, 2023, https://openreview.net/forum?id=WE_vluYUL-X.
[15] A. Leivadeas, M. Falkner, A survey on intent-based networking, IEEE Commun. Surv. Tutor. 25 (2023) 625–655, https://doi.org/10.1109/COMST.2022.3215919
[16] M. Falkner, J. Apostolopoulos, Intent-based networking for the enterprise: a modern network architecture, Commun. ACM 65 (2022) 108–117.
[17] A. El-Hassany, P. Tsankov, L. Vanbever, M.T. Vechev, NetComplete: practical network-wide configuration synthesis with autocompletion, in: Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2018, pp. 579–594.
[18] P. Notaro, J. Cardoso, M. Gerndt, A survey of AIOps methods for failure management, ACM Trans. Intell. Syst. Technol. 12 (2021), https://doi.org/10.1145/3483424
[19] S. Zhou, F.F. Xu, H. Zhu, X. Zhou, R. Lo, A. Sridhar, X. Cheng, T. Ou, Y. Bisk, D. Fried, U. Alon, G. Neubig, WebArena: a realistic web environment for building autonomous agents, in: The Twelfth International Conference on Learning Representations, 2024.
[20] M. Li, Y. Zhao, B. Yu, F. Song, H. Li, H. Yu, Z. Li, F. Huang, Y. Li, API-bank: a comprehensive benchmark for tool-augmented LLMs, in: H. Bouamor, J. Pino, K. Bali (Eds.), Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Singapore, 2023, pp. 3102–3116, https://doi.org/10.18653/v1/2023.emnlp-main.187
[21] S.G. Patil, T. Zhang, X. Wang, J.E. Gonzalez, Gorilla: large language model connected with massive APIs, in: Advances in Neural Information Processing Systems (NeurIPS 2024), 2024, pp. 126544–126565.
[22] B. Tian, X. Zhang, E. Zhai, H.H. Liu, Q. Ye, C. Wang, X. Wu, Z. Ji, Y. Sang, M. Zhang, et al., Safely and automatically updating in-network ACL configurations with intent language, in: Proceedings of the ACM Special Interest Group on Data Communication, 2019, pp. 214–226.
[23] L. Bainbridge, Ironies of automation, Automatica 19 (1983) 775–779, https://doi.org/10.1016/0005-1098(83)90046-8
[24] R. Parasuraman, V. Riley, Humans and automation: use, misuse, disuse, abuse, Hum. Factors 39 (1997) 230–253, https://doi.org/10.1518/001872097778543886
[25] M. Bilal, J. Crowcroft, R. Wang, X. Xu, S. Dustdar, LLM NetOps/AIOps Six-Pillar Evidence Audit Dataset. Mendeley Data. 2026, https://doi.org/10.17632/2p5ppxzy4s.3. Version 1, dataset.
[26] OpenTelemetry Authors, OpenTelemetry specification, 2024, https://opentelemetry.io/docs/specs/ (Accessed: 2 February 2026).
[27] P. Borman, M. Hines, C. Lebsack, C. Morrow, A. Shaikh, R. Shakir, W.B. Li, D. Loher, gRPC network management interface (gNMI). Specification. 2023, https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-specification.md (Accessed 2 February 2026).
[28] H. Song, F. Qin, P. Martinez-Julia, L. Ciavaglia, A. Wang, Network Telemetry Framework. RFC 9232. 2022, https://doi.org/10.17487/RFC9232
[29] A. Clemm, L. Ciavaglia, L.Z. Granville, J. Tantsura, Intent-Based Networking - Concepts and Definitions. RFC 9315. 2022, https://doi.org/10.17487/RFC9315
[30] Model Context Protocol, Model context protocol specification. Official specification. 2025, https://modelcontextprotocol.io/specification/2025-06-18 (Accessed 21 August 2026). Protocol revision 2025-06-18.
[31] E. Tabassi, Artificial Intelligence Risk Management Framework (AI RMF 1.0). Technical Report. 2023, https://doi.org/10.6028/NIST.AI.100-1
[32] C. Autio, R. Schwartz, J. Dunietz, S. Jain, M. Stanley, E. Tabassi, P. Hall, K. Roberts, Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile. Technical Report. 2024, https://doi.org/10.6028/NIST.AI.600-1
[33] H.J. Yoon, M. Kiran, D. Ebling, J. Breen, OFCnetLLM: large language model for network monitoring and alertness, ACM SIGMETRICS Perform. Eval. Rev. 53 (2025) 63–68, https://doi.org/10.1145/3764944.3764957
[34] L. Xiao, A.G. Latha, V. Gudepu, A. Fumagalli, A YANG-grounded LLM agent supporting multi-vendor OpenROADM optical transport networks, in: Proceedings of the ACM SIGCOMM 2026 Workshop on Hot Topics in Optical Technologies and Networking, 2026, https://doi.org/10.1145/3789240.3822606
[35] T. Ahmed, S. Ghosh, C. Bansal, T. Zimmermann, X. Zhang, S. Rajmohan, Recommending root-cause and mitigation steps for cloud incidents using large language models, in: Proceedings of the 45th IEEE/ACM International Conference on Software Engineering, 2023, pp. 1737–1749, https://doi.org/10.1109/ICSE48619.2023.00149
[36] J. Wang, L. Zhang, Y. Yang, Z. Zhuang, Q. Qi, H. Sun, L. Lu, J. Feng, J. Liao, Network meets ChatGPT: intent autonomous management, control and operation, J. Commun. Inf. Netw. 8 (2023) 239–255, https://doi.org/10.23919/JCIN.2023.10272352
[37] C. Cao, F. Wang, L. Lindley, Z. Wang, Managing Linux servers with LLM-based AI agents: an empirical evaluation with GPT-4, Mach. Learn. Appl. 17 (2024) 100570, https://doi.org/10.1016/j.mlwa.2024.100570
[38] D. Roy, X. Zhang, R. Bhave, C. Bansal, P. Las-Casas, R. Fonseca, S. Rajmohan, Exploring LLM-based agents for root cause analysis, in: Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, 2024, pp. 208–219, https://doi.org/10.1145/3663529.3663841
[39] K. Sarda, Z. Namrud, M. Litoiu, L. Shwartz, I. Watts, Leveraging large language models for the auto-remediation of microservice applications: an experimental study, in: Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, 2024, pp. 358–369, https://doi.org/10.1145/3663529.3663855
[40] X. Zhang, S. Ghosh, C. Bansal, R. Wang, M. Ma, Y. Kang, S. Rajmohan, Automated root causing of cloud incidents using in-context learning with GPT-4, in: Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, 2024, pp. 266–277, https://doi.org/10.1145/3663529.3663846
[41] D. Goel, F. Husain, A. Singh, S. Ghosh, A. Parayil, C. Bansal, X. Zhang, S. Rajmohan, X-Lifecycle learning for cloud incident management using LLMs, in: Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, 2024, pp. 417–428, https://doi.org/10.1145/3663529.3663861
[42] H. Guo, J. Yang, J. Liu, L. Yang, L. Chai, J. Bai, J. Peng, X. Hu, C. Chen, D. Zhang, X. Shi, T. Zheng, L. Zheng, B. Zhang, K. Xu, Z. Li, OWL: a large language model for IT operations, in: International Conference on Learning Representations, 2024, https://proceedings.iclr.cc/paper_files/paper/2024/hash/6581ff85ba0ca663513d5f65b25f6631-Abstract-Conference.html.
[43] Y. Chen, M. Shetty, G. Somashekar, M. Ma, Y. Simmhan, J. Mace, C. Bansal, R. Wang, S. Rajmohan, AIOpsLab: a holistic framework to evaluate AI agents for enabling autonomous clouds, in: Proceedings of Machine Learning and Systems, 2025, https://proceedings.mlsys.org/paper_files/paper/2025/hash/d1f9e4a9f109b6e8b75ed362736f22ec-Abstract-Conference.html.
[44] J. Xu, Q. Zhang, Z. Zhong, S. He, C. Zhang, Q. Lin, D. Pei, P. He, D. Zhang, Q. Zhang, OpenRCA: can large language models locate the root cause of software failures? in: International Conference on Learning Representations, 2025, https://iclr.cc/virtual/2025/poster/32093.
[45] C. Pei, Z. Wang, F. Liu, Z. Li, Y. Liu, X. He, R. Kang, T. Zhang, J. Chen, J. Li, G. Xie, D. Pei, Flow-of-action: SOP-enhanced LLM-based multi-agent system for root cause analysis, in: Companion Proceedings of the ACM Web Conference 2025, 2025, pp. 422–431, https://doi.org/10.1145/3701716.3715225
[46] A.S. Jacobs, R.J. Pfitscher, R.H. Ribeiro, L.Z. Granville, R.A. Ferreira, W. Willinger, S.G. Rao, Establishing trust for using natural language for intent-based networking, IEEE Trans. Netw. Serv. Manag. 22 (2025) 4775–4787, https://doi.org/10.1109/TNSM.2025.3574626
[47] D. Zhang, X. Zhang, C. Bansal, P. Las-Casas, R. Fonseca, S. Rajmohan, LM-PACE: confidence estimation by large language models for effective root causing of cloud incidents, in: Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, Association for Computing Machinery, 2024, pp. 388–398, https://doi.org/10.1145/3663529.3663858
[48] Y. Wang, Y. Wang, J. Liang, Y. Xu, Y. Liu, Y. Xu, Z. Jiang, Z. Li, C. Lin, F. Li, L. Zhao, K. Xu, Q. Song, X. Li, A. Wahab, TeleEval-OS: performance evaluations of large language models for telecommunications operations scheduling, Intell. Data Anal. (2025), https://doi.org/10.1177/1088467X251389769, Online first.
[49] T. Gasmi, R. Guesmi, J. Bennaceur, I. Belhadj, Bridging AI and software security: a comparative vulnerability assessment of LLM agent deployment paradigms, Inf. Sci. 740 (2026) 123231, https://doi.org/10.1016/j.ins.2026.123231
[50] Q. Cheng, D. Sahoo, A. Saha, W. Yang, C. Liu, G. Woo, M. Singh, S. Saverese, S.C.H. Hoi, AI for IT operations (AIOps) on cloud platforms: reviews, opportunities and challenges, arXiv:2304.04661, 2023.
[51] L. Zhang, T. Jia, M. Jia, Y. Wu, A. Liu, Y. Yang, Z. Wu, X. Hu, P. Yu, Y. Li, A survey of AIOps in the era of large language models, ACM Comput. Surv. 58 (2025), https://doi.org/10.1145/3746635
[52] T. Xu, Y. Zhou, Systems approaches to tackling configuration errors: a survey, ACM Comput. Surv. 47 (2015), https://doi.org/10.1145/2791577
[53] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, W.X. Zhao, Z. Wei, J. Wen, A survey on large language model based autonomous agents, Front. Comput. Sci. 18 (2024), https://doi.org/10.1007/s11704-024-40231-1
[54] Z. Xi, W. Chen, X. Guo, W. He, Y. Ding, B. Hong, M. Zhang, J. Wang, S. Jin, E. Zhou, R. Zheng, X. Fan, X. Wang, L. Xiong, Y. Zhou, W. Wang, C. Jiang, Y. Zou, X. Liu, Z. Yin, S. Dou, R. Weng, W. Cheng, Q. Zhang, W. Qin, Y. Zheng, X. Qiu, X. Huang, T. Gui, The rise and potential of large language model based agents: a survey, Sci. China Inf. Sci. 68 (2025), https://doi.org/10.1007/s11432-024-4222-0
[55] Y. Qin, S. Hu, Y. Lin, W. Chen, N. Ding, G. Cui, Z. Zeng, X. Zhou, Y. Huang, C. Xiao, C. Han, Y.R. Fung, Y. Su, H. Wang, C. Qian, R. Tian, K. Zhu, S. Liang, X. Shen, B. Xu, Z. Zhang, Y. Ye, B. Li, Z. Tang, J. Yi, Y. Zhu, Z. Dai, L. Yan, X. Cong, Y. Lu, W. Zhao, Y. Huang, J. Yan, X. Han, X. Sun, D. Li, J. Phang, C. Yang, T. Wu, H. Ji, G. Li, Z. Liu, M. Sun, Tool learning with foundation models, ACM Comput. Surv. 57 (2024), https://doi.org/10.1145/3704435
[56] X. Li, A review of prominent paradigms for LLM-based agents: tool use, planning, and feedback learning, in: Proceedings of the 31st International Conference on Computational Linguistics, Association for Computational Linguistics, 2025, pp. 9760–9779.
[57] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, H. Wang, Retrieval-augmented generation for large language models: a survey, 2023, https://doi.org/10.48550/arXiv.2312.10997
[58] X. Liu, H. Yu, H. Zhang, Y. Xu, X. Lei, H. Lai, Y. Gu, H. Ding, K. Men, K. Yang, S. Zhang, X. Deng, A. Zeng, Z. Du, C. Zhang, S. Shen, T. Zhang, Y. Su, H. Sun, M. Huang, Y. Dong, J. Tang, AgentBench: evaluating LLMs as agents, in: ICLR 2024, 2024.
[59] T. Xie, D. Zhang, J. Chen, X. Li, S. Zhao, R. Cao, T.J. Hua, Z. Cheng, D. Shin, F. Lei, Y. Liu, Y. Xu, S. Zhou, S. Savarese, C. Xiong, V. Zhong, T. Yu, OSWorld: benchmarking

multimodal agents for open-ended tasks in real computer environments, 2024, https://doi.org/10.48550/arXiv.2404.07972
[60] H. Xia, H. Wang, Z. Liu, Q. Yu, Y. Guo, H. Wang, SafeToolBench: pioneering a prospective benchmark to evaluating tool utilization safety in LLMs, in: Findings of the Association for Computational Linguistics: EMNLP 2025, Association for Computational Linguistics, 2025, pp. 17643–17660, https://doi.org/10.18653/v1/2025.findings-emnlp.958
[61] S. Jha, R. Arora, Y. Watanabe, T. Yanagawa, Y. Chen, J. Clark, B. Bhavya, M. Verma, H. Kumar, H. Kitahara, N. Zheutlin, S. Takano, D. Pathak, F. George, X. Wu, B.O. Turkkan, G. Vanloo, M. Nidd, T. Dai, O. Chatterjee, P. Gupta, S. Samanta, P. Aggarwal, R. Lee, J.W. Ahn, D. Kar, A. Paradkar, Y. Deng, P. Moogi, P. Mohapatra, N. Abe, C. Narayanaswami, T. Xu, L.R. Varshney, R. Mahindru, A. Sailer, L. Shwartz, D. Sow, N.C.M. Fuller, R. Puri, ITBench: evaluating AI agents across diverse real-world IT automation tasks, in: Proceedings of the 42nd International Conference on Machine Learning, 2025.
[62] C.E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, K.R. Narasimhan, SWE-bench: can language models resolve real-world GitHub issues? in: The Twelfth International Conference on Learning Representations, 2024.
[63] R. Mahajan, D. Wetherall, T. Anderson, Understanding BGP misconfiguration, in: Proceedings of the ACM SIGCOMM 2002 Conference, 2002, pp. 3–16, https://doi.org/10.1145/633025.633027
[64] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: enabling innovation in campus networks, SIGCOMM Comput. Commun. Rev. 38 (2008) 69–74, https://doi.org/10.1145/1355734.1355746
[65] M. Reitblatt, N. Foster, J. Rexford, D. Walker, Consistent updates for software-defined networks: change you can believe in!, in: Proceedings of the 10th ACM Workshop on Hot Topics in Networks, 2011, pp. 1–6, https://doi.org/10.1145/2070562.2070569
[66] H. Yan, L. Breslau, Z. Ge, D. Massey, D. Pei, J. Yates, G-RCA: a generic root cause analysis platform for service quality management in large IP networks, IEEE/ACM Trans. Netw. 20 (2012) 1734–1747, https://doi.org/10.1109/TNET.2012.2188837
[67] S. Kobayashi, K. Otomo, K. Fukuda, H. Esaki, Mining causality of network events in log data, IEEE Trans. Netw. Serv. Manag. 15 (2018) 53–67, https://doi.org/10.1109/TNSM.2017.2778096
[68] S. Kobayashi, K. Otomo, K. Fukuda, Causal analysis of network logs with layered protocols and topology knowledge, in: 2019 15th International Conference on Network and Service Management (CNSM), 2019, pp. 1–9, https://doi.org/10.23919/CNSM46954.2019.9012718
[69] R. Beckett, A. Gupta, R. Mahajan, D. Walker, A general approach to network configuration verification, in: Proceedings of the ACM SIGCOMM 2017 Conference, 2017, pp. 155–168, https://doi.org/10.1145/3098822.3098834
[70] N.P. Lopes, N. Bjørner, P. Godefroid, K. Jayaraman, G. Varghese, Checking beliefs in dynamic networks, in: Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI '15), 2015, pp. 499–512, https://doi.org/10.5555/2789770.2789805
[71] M.Y. Chen, E. Kiciman, E. Fratkin, A. Fox, E.A. Brewer, Pinpoint: problem determination in large, dynamic internet services, in: Proceedings of the International Conference on Dependable Systems and Networks (DSN), IEEE, 2002, pp. 595–604, https://doi.org/10.1109/DSN.2002.1029005
[72] R. Bhagwan, R. Kumar, C.S. Maddila, A.A. Philip, Orca: differential bug localization in large-scale services, in: Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI '18), 2018, pp. 493–509.
[73] Z. Chen, Y. Kang, L. Li, X. Zhang, H. Zhang, H. Xu, Y. Zhou, L. Yang, J. Sun, Z. Xu, et al., Towards intelligent incident management: why we need it and how we make it, in: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020, pp. 1487–1497, https://doi.org/10.1145/3368089.3417055
[74] J. Jiang, W. Lu, J. Chen, Q. Lin, P. Zhao, Y. Kang, H. Zhang, Y. Xiong, F. Gao, Z. Xu, et al., How to mitigate the incident? An effective troubleshooting guide recommendation technique for online service systems, in: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20), 2020, pp. 1410–1420, https://doi.org/10.1145/3368089.3417054
[75] B. Li, X. Peng, Q. Xiang, H. Wang, T. Xie, J. Sun, X. Liu, Enjoy your observability: an industrial survey of microservice tracing and analysis, Empir. Softw. Eng. 27 (2022) 25, https://doi.org/10.1007/s10664-021-10063-9
[76] Y. Wang, Z. Zhu, Q. Fu, Y. Ma, P. He, MRCA: metric-level root cause analysis for microservices via multi-modal data, in: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), 2024, pp. 1057–1068, https://doi.org/10.1145/3691620.3695485
[77] Z. Zhu, C. Lee, X. Tang, P. He, HeMiRCA: fine-grained root cause analysis for microservices with heterogeneous data sources, ACM Trans. Softw. Eng. Methodol. 33 (2024), https://doi.org/10.1145/3674726
[78] R. Fonseca, G. Porter, R.H. Katz, S. Shenker, I. Stoica, X-Trace: a pervasive network tracing framework, in: Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation (NSDI '07), 2007, pp. 271–284.
[79] J. Kaldor, J. Mace, M. Bejda, E. Gao, W. Kuropatwa, J. O'neill, K.W. Ong, B. Schaller, P. Shan, B. Viscomi, V. Venkataraman, K. Veeraraghavan, Y.J. Song, Canopy: an end-to-end performance tracing and analysis system, in: Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP '17), 2017, pp. 34–50, https://doi.org/10.1145/3132747.3132749
[80] P. He, J. Zhu, Z. Zheng, M.R. Lyu, Drain: an online log parsing approach with fixed depth tree, in: 2017 IEEE International Conference on Web Services (ICWS), 2017, pp. 33–40, https://doi.org/10.1109/ICWS.2017.13
[81] M. Du, F. Li, G. Zheng, V. Srikumar, DeepLog: anomaly detection and diagnosis from system logs through deep learning, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2017, pp. 1285–1298, https://doi.org/10.1145/3133956.3134015
[82] J. Zhu, S. He, P. He, J. Liu, M.R. Lyu, Loghub: a large collection of system log datasets for AI-driven log analytics, in: 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE), 2023, pp. 355–366, https://doi.org/10.1109/ISSRE59848.2023.00071
[83] Y. Wu, B. Chai, Y. Li, B. Liu, J. Li, Y. Yang, W. Jiang, An empirical study on change-induced incidents of online service systems, in: Proceedings of the IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), 2023, pp. 234–245, https://doi.org/10.1109/ICSE-SEIP58684.2023.00027
[84] Y. Chen, X. Yang, H. Dong, X. He, H. Zhang, Q. Lin, J. Chen, P. Zhao, Y. Kang, F. Gao, Z. Xu, D. Zhang, Identifying linked incidents in large-scale online service systems, in: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), 2020, pp. 304–314, https://doi.org/10.1145/3368089.3409768
[85] P. Las-Casas, A.G. Kumbhare, R. Fonseca, S. Agarwal, LLexus: an AI agent system for incident management, SIGOPS Oper. Syst. Rev. 58 (2024), https://doi.org/10.1145/3689051.3689056
[86] R. Opdebeeck, M. Alfadel, A. Rahman, Y. Kashiwa, J.F. Ferreira, R.G. Kula, C.D. Roover, An empirical study of policy as code: adoption, purpose, and maintenance, in: Proceedings of the 23rd International Conference on Mining Software Repositories (MSR 2026), 2026, https://doi.org/10.1145/3793302.3793355
[87] D. Sokolowski, D. Spielmann, G. Salvaneschi, Automated infrastructure as code program testing, IEEE Trans. Softw. Eng. 50 (2024) 1585–1599, https://doi.org/10.1109/TSE.2024.3393070
[88] M. Al-Fares, V. Beauregard, K. Grant, A. Griffith, J. Hasan, C. Huang, Q. Leng, J. Li, A. Lin, Z. Liu, A. Mansy, B. Martinusen, N. Mehta, J.C. Mogul, A. Narver, A. Nigham, M. Obenberger, S. Smith, K. Steinkraus, S. Sun, E. Thiele, A. Vahdat, Change management in physical network lifecycle automation, in: 2023 USENIX Annual Technical Conference (USENIX ATC 23), 2023, pp. 635–653.
[89] J. Sillito, M. Pope, Learning from lessons learned: preliminary findings from a study of learning from failure, in: Proceedings of the 2024 IEEE/ACM 17th International Conference on Cooperative and Human Aspects of Software Engineering, 2024, pp. 97–102, https://doi.org/10.1145/3641822.3641867
[90] J. Humble, D. Farley, Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation, Addison-Wesley Professional, 2010.
[91] O. Alipourfard, J. Gao, J. Koenig, C. Harshaw, A. Vahdat, M. Yu, Risk based planning of network changes in evolving data centers, in: Proceedings of the 27th ACM Symposium on Operating Systems Principles, Association for Computing Machinery, New York, NY, USA, 2019, pp. 414–429, https://doi.org/10.1145/3341301.3359664
[92] C.C. Phiri, Creating characteristically auditable agentic AI systems, in: Proceedings of Intelligent Robotics FAIR 2025 (IntRob '25), 2025, pp. 1–14, https://doi.org/10.1145/3759355.3759356
[93] R. Mondal, A. Tang, R. Beckett, T. Millstein, G. Varghese, What do LLMs need to synthesize correct router configurations? in: Proceedings of the 22nd ACM Workshop on Hot Topics in Networks (HotNets '23), Association for Computing Machinery, 2023, pp. 189–195, https://doi.org/10.1145/3626111.3628194
[94] Y. Zhou, K. Hsieh, S.K. Mani, S. Kandula, Z. Liu, MeshAgent: enabling reliable network management with large language models, Proc. ACM Meas. Anal. Comput. Syst. 9 (2025), https://doi.org/10.1145/3771567
[95] D. Wu, X. Wang, Y. Qiao, Z. Wang, J. Jiang, S. Cui, F. Wang, NetLLM: adapting large language models for networking, in: Proceedings of the ACM SIGCOMM 2024 Conference, 2024, pp. 661–678, https://doi.org/10.1145/3651890.3672268
[96] Z. Wang, S. Lin, G. Yan, S. Ghorbani, M. Yu, J. Zhou, N. Hu, L. Baruah, S. Peters, S. Kamath, J. Yang, Y. Zhang, Intent-driven network management with multi-agent LLMs: the confucius framework, in: Proceedings of the ACM SIGCOMM 2025 Conference, Association for Computing Machinery, 2025, pp. 347–362, https://doi.org/10.1145/3718958.3750537
[97] B.H. Sigelman, L.A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, C. Shanbhag, Dapper, a Large-Scale Distributed Systems Tracing Infrastructure. Technical Report. Google, Inc., 2010 Technical report (widely circulated).
[98] J. Mace, R. Roelke, R. Fonseca, Pivot tracing: dynamic causal monitoring for distributed systems, in: Proceedings of the 25th Symposium on Operating Systems Principles (SOSP '15), Association for Computing Machinery, 2015, pp. 378–393, https://doi.org/10.1145/2815400.2815415
[99] S. Yao, N. Shinn, P. Razavi, K.R. Narasimhan, τ-bench: a benchmark for tool-agent-user interaction in real-world domains, in: International Conference on Learning Representations (ICLR 2025), 2025, https://openreview.net/forum?id=roNSXZpUDN.
[100] I. McCormack, Preserving integrity in the age of generative AI, 2025. https://www.ncsc.gov.uk/blog-post/preserving-integrity-in-age-generative-ai.
[101] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, M. Fritz, Not what you've signed up for: compromising real-world LLM-integrated applications with indirect prompt injection, in: Proceedings of the 2023 Workshop on Artificial Intelligence and Security (AISec '23), Association for Computing Machinery, 2023, pp. 79–90, https://doi.org/10.1145/3605764.3623985
[102] S. Chen, J. Piet, C. Sitawarin, D. Wagner, StruQ: defending against prompt injection with structured queries, in: 34th USENIX Security Symposium (USENIX Security 25), 2025, pp. 2383–2400.

[103] D. Pasquini, E.M. Kornaropoulos, G. Ateniese, O. Akgul, A. Theocharis, P. Efstathopoulos, When AIOps become "AI oops": subverting LLM-driven IT operations via telemetry manipulation, in: 35th USENIX Security Symposium (USENIX Security 26), USENIX Association, 2026, https://www.usenix.org/conference/usenixsecurity26/presentation/pasquini.
[104] J. Sillito, E. Kutomi, Failures and fixes: a study of software system incident response, in: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2020, pp. 185–195.
[105] D.G. Widder, L. Dabbish, J.D. Herbsleb, A. Holloway, S. Davidoff, Trust in collaborative automation in high stakes software engineering work: a case study at NASA, in: Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, Association for Computing Machinery, 2021, https://doi.org/10.1145/3411764.3445650
[106] Z. Wang, Z. Liu, Y. Zhang, A. Zhong, J. Wang, F. Yin, L. Fan, L. Wu, Q. Wen, RCAgent: cloud root cause analysis by autonomous agents with tool-augmented large language models, in: Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM), 2024, pp. 4966–4974, https://doi.org/10.1145/3627673.3680016
[107] J.T. Gu, X. Sun, W. Zhang, Y. Jiang, C. Wang, M. Vaziri, O. Legunsen, T. Xu, Acto: automatic end-to-end testing for operation correctness of cloud system management, in: Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP), ACM, 2023, pp. 96–112, https://doi.org/10.1145/3600006.3613161
[108] B. Grubic, Y. Wang, T. Petrochko, R. Yaniv, B. Jones, D. Callies, M. Clarke-Lauer, D. Kelley, S. Demetriou, K. Yu, C. Tang, Conveyor: one-tool-fits-all continuous software deployment at meta, in: 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23), USENIX Association, 2023, pp. 325–342.
[109] A. Nazari, Y. Zhang, M. Raghothaman, H. Chen, Localized explanations for automatically synthesized network configurations, in: Proceedings of the 23rd ACM Workshop on Hot Topics in Networks (HotNets), 2024, pp. 52–59, https://doi.org/10.1145/3696348.3696888
[110] X. Liu, P. Zhang, A. Abhashkumar, J. Chen, W. Jiang, Automatic configuration repair, in: Proceedings of the 23rd ACM Workshop on Hot Topics in Networks (HotNets), 2024, pp. 213–220, https://doi.org/10.1145/3696348.3696895
[111] S. Zhang, Y. Xue, Y. Zhang, X. Wu, A.T. Luu, C. Zhao, MRAG: a modular retrieval framework for time-sensitive question answering, in: Findings of the Association for Computational Linguistics: EMNLP 2025, Association for Computational Linguistics, Suzhou, China, 2025, pp. 3080–3118.
[112] J. Qi, G. Sarti, R. Fernández, A. Bisazza, Model internals-based answer attribution for trustworthy retrieval-augmented generation, in: Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP 2024), 2024, pp. 6037–6053.
[113] W. Zou, R. Geng, B. Wang, J. Jia, PoisonedRAG: knowledge corruption attacks to retrieval-augmented generation of large language models, in: 34th USENIX Security Symposium (USENIX Security 25), 2025, pp. 3827–3844.
[114] A. Shafran, R. Schuster, V. Shmatikov, Machine against the RAG: jamming retrieval-augmented generation with blocker documents, in: 34th USENIX Security Symposium (USENIX Security 25), USENIX Association, 2025, pp. 3787–3806.
[115] D. Chismon, Prompt injection is not SQL injection (it may be worse), 2025, https://www.ncsc.gov.uk/blog-post/prompt-injection-is-not-sql-injection (Accessed 12 January 2026).
[116] Y. Qin, S. Liang, Y. Ye, K. Zhu, L. Yan, Y. Lu, Y. Lin, X. Cong, X. Tang, B. Qian, S. Zhao, L. Hong, R. Tian, R. Xie, J. Zhou, M. Gerstein, D. Li, Z. Liu, M. Sun, ToolLLM: facilitating large language models to master 16000+ real-world APIs, in: The Twelfth International Conference on Learning Representations, 2024.
[117] T. Cai, X. Wang, T. Ma, X. Chen, D. Zhou, Large language models as tool makers, in: ICLR 2024, 2024.
[118] A. Gember-Jacobson, C. Raiciu, L. Vanbever, Integrating verification and repair into the control plane, in: Proceedings of the 16th ACM Workshop on Hot Topics in Networks (HotNets-XVI), ACM, 2017, pp. 129–135, https://doi.org/10.1145/3152434.3152439
[119] J.H. Saltzer, M.D. Schroeder, The protection of information in computer systems, Proc. IEEE 63 (1975) 1278–1308, https://doi.org/10.1109/PROC.1975.9939
[120] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, S. Whyte, Real time network policy checking using header space analysis, in: Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation, 2013, pp. 99–112.
[121] F. Ye, D. Yu, E. Zhai, H.H. Liu, B. Tian, Q. Ye, C. Wang, X. Wu, T. Guo, C. Jin, D. She, Q. Ma, B. Cheng, H. Xu, M. Zhang, Z. Wang, R. Fonseca, Accuracy, scalability, coverage: a practical configuration verifier on a global WAN, in: Proceedings of the ACM SIGCOMM 2020 Conference, 2020, pp. 599–614, https://doi.org/10.1145/3387514.3406217
[122] I. Weber, H. Wada, A.D. Fekete, A. Liu, L. Bass, Supporting undoability in systems operations, in: Proceedings of the 27th Large Installation System Administration Conference (LISA '13), USENIX Association, 2013, pp. 75–88.
[123] G. Yu, P. Chen, Z. He, Q. Yan, Y. Luo, F. Li, Z. Zheng, ChangeRCA: finding root causes from software changes in large online systems, Proc. ACM Softw. Eng. 1 (2024), https://doi.org/10.1145/3643728
[124] C. Guo, G. Pleiss, Y. Sun, K.Q. Weinberger, On calibration of modern neural networks, in: Proceedings of the 34th International Conference on Machine Learning (ICML), PMLR, 2017, pp. 1321–1330.
[125] A. Basiri, L. Hochstein, N. Jones, H. Tucker, Automating chaos experiments in production, in: Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, IEEE, 2019, pp. 31–40, https://doi.org/10.1109/ICSE-SEIP.2019.00012
[126] S.A. Crosby, D.S. Wallach, Efficient data structures for tamper-evident logging, in: 18th USENIX Security Symposium (USENIX Security 09), USENIX Association, 2009, pp. 317–334.
[127] P. Liang, R. Bommasani, T. Lee, D. Tsipras, D. Soylu, M. Yasunaga, Y. Zhang, D. Narayanan, Y. Wu, A. Kumar, B. Newman, B. Yuan, B. Yan, C. Zhang, C. Cosgrove, C.D. Manning, C. Re, D. Acosta-Navas, D.A. Hudson, E. Zelikman, E. Durmus, F. Ladhak, F. Rong, H. Ren, H. Yao, J. Wang, K. Santhanam, L. Orr, L. Zheng, M. Yuksekgonul, M. Suzgun, N. Kim, N. Guha, N.S. Chatterji, O. Khattab, P. Henderson, Q. Huang, R.A. Chi, S.M. Xie, S. Santurkar, S. Ganguli, T. Hashimoto, T. Icard, T. Zhang, V. Chaudhary, W. Wang, X. Li, Y. Mai, Y. Zhang, Y. Koreeda, Holistic evaluation of language models, Trans. Mach. Learn. Res. (2023), ISSN 2835-8856, https://openreview.net/forum?id=iO4LZibEqW.
[128] D. Turner, K. Levchenko, A.C. Snoeren, S. Savage, California fault lines: understanding the causes and impact of network failures, in: Proceedings of the ACM SIGCOMM 2010 Conference, 2010, pp. 315–326, https://doi.org/10.1145/1851182.1851220
[129] P. Gill, N. Jain, N. Nagappan, Understanding network failures in data centers: measurement, analysis, and implications, in: Proceedings of the ACM SIGCOMM 2011 Conference, 2011, pp. 350–361, https://doi.org/10.1145/2018436.2018477
[130] M. Brown, A. Fogel, D. Halperin, V. Heorhiadi, R. Mahajan, T. Millstein, Lessons from the evolution of the batfish configuration analysis tool, in: Proceedings of the ACM SIGCOMM 2023 Conference, 2023, pp. 122–135, https://doi.org/10.1145/3603269.3604866
[131] M. Bilal, Network-optimised spiking neural network for event-driven networking, arXiv preprint, 2025, https://doi.org/10.48550/arXiv.2509.23516
[132] M. Bilal, X. Xu, Network-optimised spiking neural network (NOS) scheduling for 6G O-RAN: spectral margin and delay-tail control, arXiv preprint, 2025, https://doi.org/10.48550/arXiv.2510.11291
[133] A. Horn, A. Kheradmand, M. Prasad, Delta-Net: real-time network verification using atoms, in: 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), USENIX Association, Boston, MA, 2017, pp. 735–749.
[134] R. Mondal, N. Bjørner, T.D. Millstein, A. Tang, G. Varghese, Tackling ambiguity in user intent for LLM-based network configuration synthesis, in: Proceedings of the 24th ACM Workshop on Hot Topics in Networks (HotNets '25), 2025, pp. 176–183, https://doi.org/10.1145/3772356.3772402. Workshop paper; PDF available from HotNets.
[135] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, D. Walker, Don't mind the gap: bridging network-wide objectives and device-level configurations, in: Proceedings of the ACM SIGCOMM 2016 Conference, 2016, pp. 328–341, https://doi.org/10.1145/2934872.2934909
[136] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, D. Walker, Network configuration synthesis with abstract topologies, in: Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), 2017, pp. 437–451, https://doi.org/10.1145/3062341.3062367
[137] A. El-Hassany, P. Tsankov, L. Vanbever, M. Vechev, Network-wide configuration synthesis, in: International Conference on Computer Aided Verification, 2017, pp. 261–281, https://doi.org/10.1007/978-3-319-63390-9_14
[138] L. Vanbever, S. Vissicchio, C. Pelsser, P. Francois, O. Bonaventure, Lossless migrations of link-state IGPs, IEEE/ACM Trans. Netw. 20 (2012) 1842–1855, https://doi.org/10.1109/TNET.2012.2190767
[139] X. Xu, W. Deng, R. Beckett, R. Mahajan, D. Walker, Test coverage for network configurations, in: 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23), USENIX Association, Boston, MA, 2023, pp. 1717–1732.
[140] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, N. McKeown, I know what your packet did last hop: using packet histories to troubleshoot networks, in: 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), USENIX Association, Seattle, WA, 2014, pp. 71–85.
[141] X. Hua, Q. Li, Z. Zhang, X. Fan, L. Zhu, R. Yu, Y. Zhou, Y. Zhang, J. Feng, C. Deng, X. Yuan, CSDNet: causal inference aided handover parameter adjusting effect estimation in cellular networks, in: ICC 2023 - IEEE International Conference on Communications, 2023, pp. 5304–5309, https://doi.org/10.1109/ICC45041.2023.10279707
[142] Y. Zhang, Q. Li, X. Hua, R. Yu, Z. Zhang, X. Fan, L. Zhu, T. Sha, Y. Zhang, DCDN: estimating handover parameter adjusting effect with causal inference, in: 2023 IEEE 98th Vehicular Technology Conference (VTC2023-Fall), 2023, pp. 1–5, https://doi.org/10.1109/VTC2023-Fall60731.2023.10333788
[143] M. Li, Y. Sun, X. Hua, R. Yu, X. Fan, L. Zhu, J. Feng, D. Pei, Predicting parameter change's effect on cellular network time series, arXiv:2408.15516, 2024.
[144] A. Ikram, S. Chakraborty, S. Mitra, S. Saini, S. Bagchi, M. Kocaoglu, Root cause analysis of failures in microservices through causal discovery, in: Advances in Neural Information Processing Systems (NeurIPS), 2022, pp. 31158–31170.
[145] Z. Yao, C. Pei, W. Chen, H. Wang, L. Su, H. Jiang, Z. Xie, X. Nie, D. Pei, Chain-of-event: interpretable root cause analysis for microservices through automatically learning weighted event causal graph, in: Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE-Companion), ACM, 2024, pp. 50–61, https://doi.org/10.1145/3663529.3663827
[146] Z. Li, J. Chen, R. Jiao, N. Zhao, Z. Wang, S. Zhang, Y. Wu, L. Jiang, L. Yan, Z. Wang, Z. Chen, W. Zhang, X. Nie, K. Sui, D. Pei, Practical root cause localization for microservice systems via trace analysis, in: Proceedings of the IEEE/ACM 29th International Symposium on Quality of Service (IWQoS '21), IEEE, 2021, pp. 1–10, https://doi.org/10.1109/IWQOS52092.2021.9521340
[147] N. Zhao, H. Wang, Z. Li, X. Peng, G. Wang, Z. Pan, Y. Wu, Z. Feng, X. Wen, W. Zhang, K. Sui, D. Pei, An empirical investigation of practical log anomaly detection for online service systems, in: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), Association for Computing Machinery, 2021, pp. 1404–1415, https://doi.org/10.1145/3468264.3473933

[148] X. Ma, Y. Li, J. Keung, X. Yu, H. Zou, Z. Yang, F. Sarro, E.T. Barr, Practitioners' expectations on log anomaly detection, IEEE Trans. Softw. Eng. 51 (2025) 2455–2471, https://doi.org/10.1109/TSE.2025.3586700
[149] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, M.R. Lyu, Tools and benchmarks for automated log parsing, in: Proceedings of the IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '19), IEEE, 2019, pp. 121–130, https://doi.org/10.1109/ICSE-SEIP.2019.00021
[150] M. Bilal, O. Tariq, H. Ahmed, NOS-gate: queue-aware streaming IDS for consumer gateways under timing-controlled evasion, IEEE Trans. Consum. Electron. (2026) 1, https://doi.org/10.1109/TCE.2026.3682516
[151] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Commun. ACM 21 (1978) 558–565, https://doi.org/10.1145/359545.359563
[152] F.B. Schneider, Implementing fault-tolerant services using the state machine approach: a tutorial, ACM Comput. Surv. 22 (1990) 299–319, https://doi.org/10.1145/98163.98167
[153] R. Pedro, M.E. Coimbra, D. Castro, P. Carreira, N. Santos, Prompt-to-SQL injections in LLM-integrated web applications: risks and defenses, in: Proceedings of the IEEE/ACM 47th International Conference on Software Engineering, IEEE Press, 2025, pp. 1768–1780, https://doi.org/10.1109/ICSE55347.2025.00007
[154] Y. Liu, G. Deng, Y. Li, K. Wang, Z. Wang, X. Wang, T. Zhang, Y. Liu, H. Wang, Y. Zheng, et al., Prompt injection attack against LLM-integrated applications, arXiv preprint arXiv:2306.05499, 2023.
[155] Z. Li, N. Zhao, S. Zhang, Y. Sun, P. Chen, X. Wen, M. Ma, D. Pei, Constructing large-scale real-world benchmark datasets for AIOps, arXiv preprint arXiv:2208.03938 arXiv:2208.03938, 2022.
[156] Y. Sun, M. Shao, X. Nie, K. Yang, X. Li, B. Hao, S. Zhang, C. Pei, D. He, Y. Li, D. Pei, A comprehensive benchmark and empirical study of trace anomaly detection, IEEE Trans. Serv. Comput. 18 (2025) 3364–3377, https://doi.org/10.1109/TSC.2025.3622122
[157] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.F. Crespo, D. Dennison, Hidden technical debt in machine learning systems, in: Advances in Neural Information Processing Systems 28, 2015, pp. 2503–2511.
[158] E. Breck, S. Cai, E. Nielsen, M. Salib, D. Sculley, The ML test score: a rubric for ML production readiness and technical debt reduction, in: 2017 IEEE International Conference on Big Data (IEEE BigData 2017), 2017, pp. 1123–1132, https://doi.org/10.1109/BIGDATA.2017.8258038
[159] C.G. Northcutt, A. Athalye, J. Mueller, Pervasive label errors in test sets destabilize machine learning benchmarks, in: Thirty-Fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1), 2021, https://openreview.net/forum?id=XccDXrDNLek.
[160] M.T. Ribeiro, T. Wu, C. Guestrin, S. Singh, Beyond accuracy: behavioral testing of NLP models with CheckList, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, 2020, pp. 4902–4912, https://doi.org/10.18653/v1/2020.acl-main.442
[161] C. Ma, J. Zhang, Z. Zhu, C. Yang, Y. Yang, Y. Jin, Z. Lan, L. Kong, J. He, AgentBoard: an analytical evaluation board of multi-turn LLM agents, in: Advances in Neural Information Processing Systems, Curran Associates, Inc., 2024, https://doi.org/10.52202/079017-2365, https://proceedings.neurips.cc/paper_files/paper/2024/file/877b40688e330a0e2a3fc24084208dfa-Paper-Datasets_and_Benchmarks_Track.pdf.
[162] L. Gioacchini, G. Siracusano, D. Sanvito, K. Gashteovski, D. Friede, R. Bifulco, C. Lawrence, AgentQuest: a modular benchmark framework to measure progress and improve LLM agents, in: Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: System Demonstrations), Association for Computational Linguistics, 2024, pp. 185–193, https://doi.org/10.18653/v1/2024.naacl-demo.19
[163] Z. Guo, Y. Huang, D. Xiong, CToolEval: a Chinese benchmark for LLM-powered agent evaluation in real-world API interactions, in: Findings of the Association for Computational Linguistics: ACL 2024, Association for Computational Linguistics, 2024, pp. 15711–15724, https://doi.org/10.18653/v1/2024.findings-acl.928
[164] L. Gioacchini, A. Delsanto, I. Drago, M. Mellia, G. Siracusano, R. Bifulco, AutoPenBench: a vulnerability testing benchmark for generative agents, in: Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track, Association for Computational Linguistics, 2025, pp. 1615–1624, https://doi.org/10.18653/v1/2025.emnlp-industry.114
[165] T. Xu, L. Chen, D.J. Wu, Y. Chen, Z. Zhang, X. Yao, Z. Xie, Y. Chen, S. Liu, B. Qian, A. Yang, Z. Jin, J. Deng, P. Torr, B. Ghanem, G. Li, CRAB: cross-environment agent benchmark for multimodal language model agents, in: Findings of the Association for Computational Linguistics: ACL 2025, Association for Computational Linguistics, 2025, pp. 21607–21647, https://doi.org/10.18653/v1/2025.findings-acl.1113
[166] R. Kapoor, Y.P. Butala, M.A. Russak, J.Y. Koh, K. Kamble, W. AlShikh, R. Salakhutdinov, OmniACT: a dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web, in: Proceedings of the European Conference on Computer Vision (ECCV), 2024, pp. 161–178, https://doi.org/10.1007/978-3-031-73113-6_10
[167] L. Zheng, W.L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E.P. Xing, H. Zhang, J.E. Gonzalez, I. Stoica, Judging LLM-as-a-Judge with MT-bench and chatbot arena, in: Proceedings of the 37th International Conference on Neural Information Processing Systems, 2023.
[168] J.Y. Koh, R. Lo, L. Jang, V. Duvvur, M. Lim, P.Y. Huang, G. Neubig, S. Zhou, R. Salakhutdinov, D. Fried, VisualWebArena: evaluating multimodal agents on realistic visual web tasks, in: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, 2024, pp. 881–905, https://doi.org/10.18653/v1/2024.acl-long.50
[169] A. Sorstkins, O. Tariq, M. Bilal, Learning to undo: rollback-augmented reinforcement learning with reversibility signals, 2025, https://doi.org/10.48550/arXiv.2510.14503
[170] Q. Zhan, R. Fang, H.S. Panchal, D. Kang, Adaptive attacks break defenses against indirect prompt injection attacks on LLM agents, in: L. Chiruzzo, A. Ritter, L. Wang (Eds.), Findings of the Association for Computational Linguistics: NAACL 2025, Association for Computational Linguistics, 2025, pp. 7116–7132, https://doi.org/10.18653/v1/2025.findings-naacl.395
[171] Y. Chen, H. Li, Z. Zheng, D. Wu, Y. Song, B. Hooi, Defense against prompt injection attack by leveraging attack techniques, in: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, 2025, pp. 18331–18347, https://doi.org/10.18653/v1/2025.acl-long.897
[172] R. Zhang, D. Sullivan, K. Jackson, P. Xie, M. Chen, Defense against prompt injection attacks via mixture of encodings, in: Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers), Association for Computational Linguistics, 2025, pp. 244–252, https://doi.org/10.18653/v1/2025.naacl-short.21
[173] Y. Chen, H. Li, Y. Sui, Y. Song, B. Hooi, Backdoor-powered prompt injection attacks nullify defense methods, in: Findings of the Association for Computational Linguistics: EMNLP 2025, Association for Computational Linguistics, 2025, pp. 4508–4527, https://doi.org/10.18653/v1/2025.findings-emnlp.242
[174] N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson, A. Oprea, C. Raffel, Extracting training data from large language models, in: 30th USENIX Security Symposium (USENIX Security 21), USENIX Association, 2021, pp. 2633–2650.
[175] Y. Song, R. Liu, S. Chen, Q. Ren, Y. Zhang, Y. Yu, SecureSQL: evaluating data leakage of large language models as natural language interfaces to databases, in: Findings of the Association for Computational Linguistics: EMNLP 2024, Association for Computational Linguistics, 2024, pp. 5975–5990, https://doi.org/10.18653/v1/2024.findings-emnlp.346
[176] J. Wang, T. Yang, R. Xie, B. Dhingra, Raccoon: prompt extraction benchmark of LLM-integrated applications, in: Findings of the Association for Computational Linguistics: ACL 2024, Association for Computational Linguistics, 2024, pp. 13349–13365, https://doi.org/10.18653/v1/2024.findings-acl.791
[177] A. Narayanan, V. Shmatikov, Robust de-anonymization of large sparse datasets, in: 2008 IEEE Symposium on Security and Privacy (SP 2008), 2008, pp. 111–125, https://doi.org/10.1109/SP.2008.33
[178] J.D. Lee, K.A. See, Trust in automation: designing for appropriate reliance, Hum. Factors 46 (2004) 50–80, https://doi.org/10.1518/hfes.46.1.50_30392
[179] Q. Zhan, Z. Liang, Z. Ying, D. Kang, InjecAgent: benchmarking indirect prompt injections in tool-integrated large language model agents, in: Findings of the Association for Computational Linguistics: ACL 2024, Association for Computational Linguistics, 2024, pp. 10471–10506, https://doi.org/10.18653/v1/2024.findings-acl.624
[180] Z. Wang, V. Siu, Z. Ye, T. Shi, Y. Nie, X. Zhao, C. Wang, W. Guo, D. Song, AGENTVIGIL: automatic black-box red-teaming for indirect prompt injection against LLM agents, in: Findings of the Association for Computational Linguistics: EMNLP 2025, Association for Computational Linguistics, 2025, pp. 23159–23172, https://doi.org/10.18653/v1/2025.findings-emnlp.1258
[181] F. Jia, T. Wu, X. Qin, A. Squicciarini, The task shield: enforcing task alignment to defend against indirect prompt injection in LLM agents, in: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, 2025, pp. 29680–29697, https://doi.org/10.18653/v1/2025.acl-long.1435
[182] Q. Zeng, C. Jin, X. Wang, Y. Zheng, Q. Li, AIRepr: an analyst-inspector framework for evaluating reproducibility of LLMs in data science, in: Findings of the Association for Computational Linguistics: EMNLP 2025, Association for Computational Linguistics, 2025, pp. 10170–10201, https://doi.org/10.18653/v1/2025.findings-emnlp.539
[183] D. Kim, G. Shim, Y. Chun, M. Kim, C. Park, H. Lim, Benchmark profiling: mechanistic diagnosis of LLM benchmarks, in: Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2025, pp. 15635–15650, https://doi.org/10.18653/v1/2025.emnlp-main.789
[184] K. Veeraraghavan, J. Meza, D. Chou, W. Kim, S. Margulis, S. Michelson, R. Nishtala, D. Obenshain, D. Perelman, Y.J. Song, Kraken: leveraging live traffic tests to identify and resolve resource utilization bottlenecks in large scale web services, in: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 635–651.
[185] X. Sun, W. Luo, J.T. Gu, A. Ganesan, R. Alagappan, M. Gasch, L. Suresh, T. Xu, Automatic reliability testing for cluster management controllers, in: 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22), 2022, pp. 143–159.
[186] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, Y. He, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvinsky, M. Espinosa, R. Lin, Z. Liu, J. Padilla, C. Delimitrou, An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems, in: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, 2019, pp. 3–18.

[187] L. Hu, G. Chen, X. Shang, S. Cheng, B. Wu, G. Li, X. Zhu, W. Zhang, N. Yu, CompileAgent: automated real-world repo-level compilation with tool-integrated LLM-based agent system, in: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, 2025, pp. 2078–2091, https://doi.org/10.18653/v1/2025.acl-long.103

[188] M. Li, Z. Li, K. Yin, X. Nie, W. Zhang, K. Sui, D. Pei, Causal inference-based root cause analysis for online service systems with intervention recognition, in: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery, New York, NY, USA, 2022, pp. 3230–3240, https://doi.org/10.1145/3534678.3539041

[189] Y. Liu, Y. Jia, R. Geng, J. Jia, N.Z. Gong, Formalizing and benchmarking prompt injection attacks and defenses, in: 33rd USENIX Security Symposium (USENIX Security 24), 2024, pp. 1831–1847.

[190] G. Deng, Y. Liu, Y. Li, K. Wang, Y. Zhang, Z. Li, H. Wang, T. Zhang, Y. Liu, MASTERKEY: automated jailbreaking of large language model chatbots, in: Proceedings of the Network and Distributed System Security Symposium, 2024, https://doi.org/10.14722/ndss.2024.24188

[191] N. Carlini, M. Jagielski, C.A. Choquette-Choo, D. Paleka, W. Pearce, H.S. Anderson, A. Terzis, K. Thomas, F. Tramèr, Poisoning web-scale training datasets is practical, in: 2024 IEEE Symposium on Security and Privacy (SP), 2024, pp. 407–425.

[192] T. Wen, C. Wang, X. Yang, H. Tang, Y. Xie, L. Lyu, Z. Dou, F. Wu, Defending against indirect prompt injection by instruction detection, in: Findings of the Association for Computational Linguistics: EMNLP 2025, Association for Computational Linguistics, 2025, pp. 19472–19487, https://doi.org/10.18653/v1/2025.findings-emnlp.1060