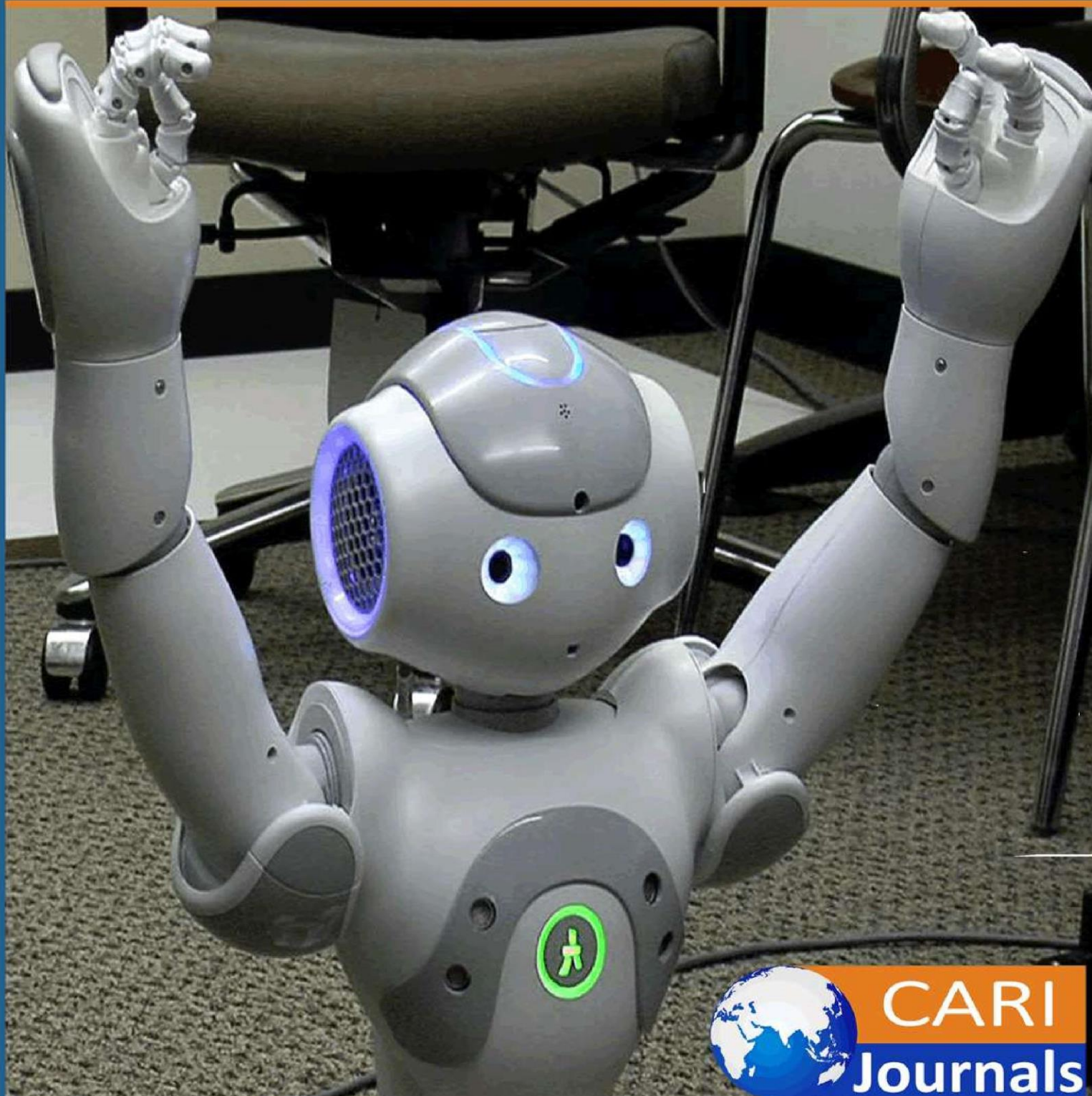
International Journal of
Computing and
Engineering
(IJCE)

TriageGPT for DataOps: Large Language Models as Intelligent Incident Classification and Resolution Agents in Data Pipeline Operations



CARI Journals

International Journal of Computing and Engineering
ISSN 2958-7425 (online)
Vol. 8, Issue No. 4, pp. 32 - 54, 2026
www.carijournals.org

TriageGPT for DataOps: Large Language Models as Intelligent Incident Classification and Resolution Agents in Data Pipeline Operations

Prabhu Ravi, Rajesh Thondamanati

Accepted: 19th Oct, 2026, Received in Revised Form: 9th Nov, 2026, Published: 19th Nov, 2026

Abstract

Purpose: This paper presents TriageGPT for DataOps, an architecture in which large language model agents classify data pipeline incidents, retrieve relevant context, and perform constrained autonomous remediation, in order to reduce the manual triage effort that drives incident duration and on-call fatigue.

Methodology: TriageGPT combines a retrieval-augmented classification agent that assigns alerts to a data pipeline specific incident taxonomy, a tool-use resolution agent that invokes actions from a vetted, blast-radius-tiered action library, and an audit-logged guardrail layer that gates and can automatically roll back actions, paired with confidence-based human-in-the-loop escalation. We describe the architecture, algorithms and calibration policy, map the design to safety and governance guidance, and report a prototype evaluation on a benchmark pipeline with a fourteen-category taxonomy, 2,800 labeled synthetic incidents and a twelve-week comparative deployment trial.

Findings: TriageGPT achieved 94.3 percent top-one classification accuracy and 95.1 percent factual attribution accuracy with retrieval, against 68.4 percent without it. It resolved 71.5 percent of test incidents without human intervention at an incorrect-action rate of 0.6 percent, every one of which the guardrail caught and reversed. In the deployment trial it reduced median time to resolution from 41 minutes under manual on-call practice to 9.8 minutes.

Unique Contribution to Theory, Policy and Practice: For theory, the paper treats autonomous remediation as something earned through calibrated confidence and bounded blast radius rather than assumed. For practice, it provides an architecture, algorithms and evaluation method for resolving routine pipeline incidents while deferring the rest to a human with an evidence-grounded case. For policy, it maps autonomous remediation to existing guidance, informing how organizations set guardrails, audit requirements and risk tolerance for AIOps agents.

Keywords: *Large Language Models, LLM agents, AIOps, DataOps, Incident Classification, Autonomous Remediation*

JEL Codes: *C45, C88, L86, M15, O33*

1. Introduction

A production data pipeline chains an orchestration scheduler, a distributed transformation engine, a streaming or batch ingestion layer that pulls from sources the platform team does not control, a warehouse or lakehouse, and automated data quality checks. Each layer emits its own alerts in its own vocabulary, and an on-call data engineer must read across all of them, decide what happened and decide what to do, often under time pressure and outside working hours. As pipelines, sources and consumers multiply, alert volume grows, yet the manual triage step of reading an alert, forming a hypothesis, checking it against logs, metrics and lineage, and selecting a remediation does not get materially faster. This triage burden is a leading driver of incident duration and on-call fatigue in data engineering organizations, and it is the problem this paper addresses.

Large language models can follow multi-step reasoning, synthesize evidence from heterogeneous text and select among tools, capabilities formalized in the reasoning-and-acting pattern and in function-calling frameworks [5], [7], [8], [10]. These map closely onto incident triage, and prior work has applied large language models to root cause analysis and incident summarization for cloud infrastructure [24], [25], [26], [27]. Data pipelines, however, present a distinct taxonomy dominated by schema drift, late-arriving data, data quality breaches and partition skew, which general cloud incident research does not target and which classical automated root cause analysis covers only where a fixed rule or learned anomaly signature applies [16], [17], [18], [21], [22].

TriageGPT classifies pipeline incidents against a purpose-built taxonomy, retrieves grounding context from a pipeline's own lineage metadata, runbooks and incident history, and, for a bounded and vetted set of actions, executes a resolution autonomously. The central tension is that an agent able to take a real remediation action can also take a wrong one, and restarting the wrong job or backfilling the wrong partition can make an incident worse. TriageGPT therefore does not treat autonomy as an unqualified goal: every action comes from a vetted library tagged with a blast-radius tier, actions below a confidence threshold are routed to a human, and every executed action is logged to support real-time rollback and audit, consistent with emerging governance guidance for autonomous AI systems [42], [43], [44].

The paper makes five contributions: (1) a data pipeline incident taxonomy and an architecture in which classification, resolution and guardrail components are distinct and independently evaluable; (2) classification, retrieval and constrained tool-use algorithms with an explicit calibration and escalation policy; (3) a mapping from the guardrail design to safety and governance guidance; (4) a prototype evaluation of accuracy, grounding, autonomous resolution safety and time to resolution against manual and rule-based baselines; and (5) illustrative use cases and an explicit discussion of residual risk. Section 2 reviews related work, Section 3 formulates the problem, Sections 4 to 7 present the architecture, algorithms, governance and taxonomy, Section 8 reports the evaluation, Section 9 presents use cases, Section 10 discusses limitations, and Section 11 concludes.

2. Background and Related Work

2.1 Large language model agents and tool use

Transformer-based, few-shot-capable language models underlie every TriageGPT component [1], [2], [3]. Instruction tuning from human feedback makes models more reliable at following structured task formats, and chain-of-thought prompting elicits intermediate reasoning that is both more accurate and more inspectable [4], [5]; a broader survey catalogs related prompting techniques such as exemplar selection and structured output [6]. The reasoning-and-acting pattern interleaves reasoning with actions against an environment, observing each result before the next step [7]. Toolformer showed that models can learn when to call external interfaces, and providers now expose structured function calling directly [8], [10]. Reflexion adds self-critique after failure, relevant when a remediation does not resolve an incident [9], and multi-agent and orchestration frameworks have popularized patterns for combining retrieval, tool use and memory in one agent loop [11], [12].

2.2 Classical and LLM-based AIOps

Earlier AIOps research used structured parsing and statistical or learned models over telemetry. Drain introduced fixed-depth-tree online log parsing [16]; DeepLog detected anomalous log sequences with recurrent networks, and LogRobust handled unstable logging statements across software versions [17], [18]. A systematic evaluation found reported accuracy in this literature sensitive to train and test splitting, a caution our evaluation follows by using a disjoint held-out set [19]. HotSpot localized anomalies in multi-dimensional KPIs, MicroRCA and Microscope localized root causes in microservices from dependency and causal graphs, and a survey catalogs this broader landscape [20], [21], [22], [23]. TriageGPT does not replace this machinery; its ingestion layer consumes such outputs as inputs to classification.

More recently, RCACopilot performed root cause analysis by reasoning over retrieved diagnostics and similar past incidents, follow-up work explored agents that iteratively query diagnostic tools, Xpert recommended diagnostic queries, and other work generated concise outage summaries [24], [25], [26], [27]. OpsEval and a survey of AIOps in the era of large language models catalog these tasks and note that data pipeline specific incident categories remain comparatively underexplored [28], [29], the gap TriageGPT addresses.

2.3 Retrieval, agent evaluation and pipeline observability

Retrieval-augmented generation grounds outputs in passages retrieved at inference time, reducing hallucination and allowing reasoning over private material such as runbooks [13]; it has since been extended into agentic retrieval and applied to enterprise knowledge management [14], [15]. Evaluating agents that act requires benchmarks beyond token accuracy, such as AgentBench, ToolLLM, StableToolBench and SWE-bench [30], [31], [32], [33]. Large language models validated against human preferences are widely used as automated judges [34], an approach we adapt to judge explanation attribution, and a recent survey organizes definitions and detection of

hallucination [35]. For pipelines specifically, Shankar et al. argue that production pipelines need observability built for their data-flow structure [38], motivating lineage-aware retrieval, and Spark's resilient distributed datasets define the execution model within which many resource and skew incidents arise [39]. Table 1 compares TriageGPT with representative approaches.

Table 1. Comparison of TriageGPT against representative approaches to data pipeline incident handling.

Approach	Incident classification	Diagnosis grounding	Remediation	Auditable safety boundary
Manual on-call triage	Implicit, in the engineer's judgment	Ad hoc runbook search	Human-executed	Informal
Static rule-based automation	Fixed keyword or threshold rules	None	Fixed pre-scripted action per rule	Deterministic, no confidence signal
Classical AIOps (log parsing, anomaly localization)	Anomaly detection, not taxonomy	Structured telemetry only	Not addressed	Not addressed
General-purpose LLM assistant (ungrounded)	Free text, inconsistent	Parametric, prone to hallucination	Text recommendation only	Not addressed
TriageGPT (this work)	Taxonomy classification with calibrated confidence	Lineage, runbooks and incident history	Constrained execution from a tiered action library	Confidence-gated escalation, rollback, audit log

3. Problem Formulation

3.1 Terminology

An *incident* is any alert or detected deviation from expected pipeline behavior that warrants triage; an *incident class* is a category in the taxonomy of Section 7 sharing a typical root cause and remediation set. *Triage* is classification plus retrieval of the evidence that justifies it, and *resolution* is selecting and, where appropriate, executing a remediation. Time to acknowledge (TTA) runs from first alert to a confident, evidence-backed classification, and time to resolution (TTR) to verified resolution; both are standard site reliability metrics [40]. An *autonomous action* is executed without prior human approval. *Blast radius* is a qualitative tier (Section 6) describing an action's scope and reversibility, from restarting one failed task to deleting a partition. The *guardrail* combines a confidence threshold, a blast-radius policy and a rollback mechanism. *Grounding* is

the degree to which an explanation is traceable to retrieved artifacts, and *hallucination* is an explanation citing an artifact that does not exist in the pipeline, which lineage metadata makes directly measurable [35].

3.2 Core problem and design requirements

An on-call engineer must answer what kind of problem an alert represents, why, and what to do. Answering manually is slow in proportion to the systems to cross-reference and the unfamiliarity of the failure, and rarely exploits incident history unless it is well indexed. TriageGPT answers the same questions computationally, executing the answer to the third automatically only where that is safe. Four requirements follow. First, classification must be structured and calibrated so that confidence can drive escalation. Second, every diagnostic claim must be grounded in retrievable, pipeline-specific evidence that a reviewer or auditor can verify. Third, autonomous remediation must be bounded by an explicit, auditable policy rather than the model's own judgment, because stated confidence is not a reliable safety signal. Fourth, every autonomous action must be reversible in practice, with incorrect actions detected and rolled back within a bounded window.

4. TriageGPT System Architecture

TriageGPT has six components (Figure 1): signal ingestion and normalization, a retrieval-augmented context builder, a classification agent, a resolution agent, a guardrail and escalation layer, and a feedback loop.

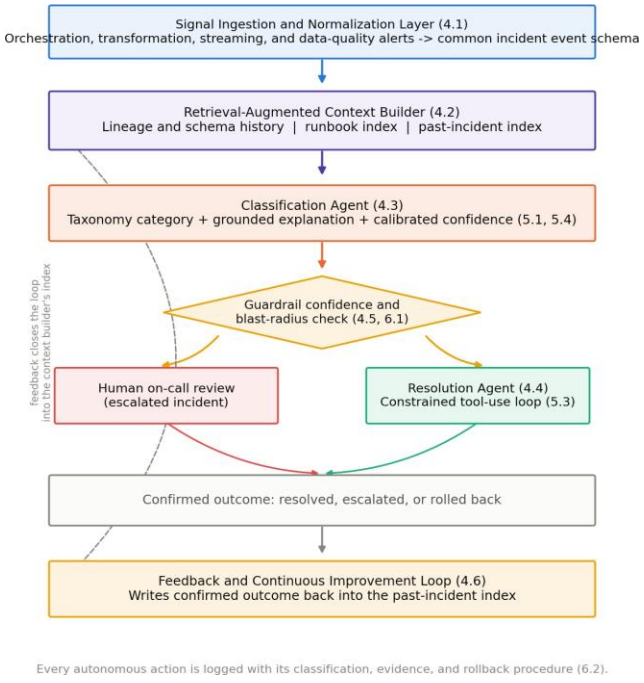


Figure 1. TriageGPT system architecture: signal ingestion, retrieval-augmented context builder, classification and resolution agents, guardrail and escalation layer, and the feedback loop into the indexed corpus.

4.1 Signal ingestion and normalization

Each pipeline system emits alerts in its own format: a scheduler emits task failures with stack traces, a data quality framework emits expectation failures naming the check and dataset, and a streaming platform emits lag or error-rate breaches. The ingestion layer normalizes every signal into a common incident event recording the source system, timestamp, structured fields such as task, dataset or check name, and the raw payload, so that a single classification agent can handle incidents from any layer.

4.2 Retrieval-augmented context builder

A normalized event rarely contains its own root cause: a task failure names the task but not the upstream dataset whose schema changed. The context builder maintains three indices: lineage and schema history built from lineage metadata, a runbook and documentation index, and a past-incident index of prior classifications, evidence and resolutions. It assembles the most relevant passages and lineage records from all three into the grounding context, implementing the grounding requirement of Section 3.2 [13].

4.3 Classification agent

The classification agent is a large language model prompted with the event, the retrieved context and few-shot exemplars. It assigns one of fourteen taxonomy categories, explains its choice by citing retrieved artifacts, and outputs a confidence score. It does not decide whether to escalate; the guardrail layer does, keeping classification and the safety-relevant escalation decision separately auditable.

4.4 Resolution agent

For incidents authorized for autonomous handling, the resolution agent follows the reasoning-and-acting pattern: it selects an action, invokes it through a structured function call, observes the result, and either concludes, tries a further action, or escalates after a bounded number of steps [7], [8]. Actions come only from a fixed library defined and reviewed by the platform team (Section 7.2), so the space of possible autonomous actions is auditable in advance.

4.5 Guardrail and human-in-the-loop escalation

The guardrail sits between the agents and production. Before any action runs, it checks classification confidence against the escalation policy (Section 5.4) and the action's blast-radius tier (Section 6) against what that confidence authorizes, and it logs enough to reconstruct and reverse the action. For actions with a rollback procedure, it monitors the incident's key indicators for a confirmation window and rolls back automatically if they do not improve.

4.6 Feedback loop

Every triaged incident is written back to the past-incident index with its confirmed outcome. Correct resolutions become future exemplars, and human corrections or rollbacks become labeled

cases for tightening the policy or action library. TriageGPT deliberately does not fine-tune model weights; improvement happens through the index and human-reviewable configuration, keeping behavior auditable and reversible.

5. Methodology: Classification, Retrieval, and Constrained Tool Use

5.1 Incident classification

Algorithm 1 summarizes classification. The agent requests grounding context, builds a prompt with the event, the context and one exemplar per taxonomy category so that every category is represented regardless of frequency, and produces a classification, cited explanation and confidence in a single structured output.

Algorithm 1: Incident classification with grounded explanation

Input: normalized incident event e, incident taxonomy T,
past-incident index I, lineage and runbook indices L, R
Output: classification c in T, grounded explanation x, confidence score p in [0, 1]

```
1  ctx <- RETRIEVE_CONTEXT(e, L, R)                 // Section 5.2
2  exemplars <- SELECT_EXEMPLARS(e, I, T)           // one per category in T
3  prompt <- BUILD_PROMPT(e, ctx, exemplars, T)
4  (c, x, p_raw) <- LLM_CLASSIFY(prompt)
5  cited <- EXTRACT_CITED_ARTIFACTS(x)
6  verified <- VERIFY_AGAINST_LINEAGE(cited, L)      // grounding check
7  if verified is not all true:
8      flag x for grounding failure
9      p_raw <- p_raw * GROUNDING_PENALTY
10 p <- CALIBRATE(p_raw)                            // Section 5.4
11 return (c, x, p)
```

Step 6 operationalizes the grounding requirement. Because lineage metadata is authoritative for which datasets, jobs and parameters exist, TriageGPT checks that every cited artifact is real and lies within two lineage hops of the incident's own job or dataset, and penalizes confidence otherwise, so an ungrounded explanation is unlikely to reach a confidence that authorizes action. The two-hop bound reflects the observation that most correctly diagnosed past incidents cited an

artifact within that distance, and it also catches the subtler failure of citing a real but unrelated artifact.

5.2 Retrieval-augmented context construction

Each index is embedded and queried by nearest-neighbor search with a query built from the event's fields and alert text. For lineage, the query also traverses the graph outward from any named dataset or job, retrieving immediate upstream producers and downstream consumers and any schema or configuration change within a look-back window, since schema drift and upstream failures are usually explained by changes to dependencies. Runbook and past-incident retrieval uses dense passage retrieval restricted to the top-k passages per corpus, bounding context size [13]. Figure 2 shows this flow for a schema drift incident.

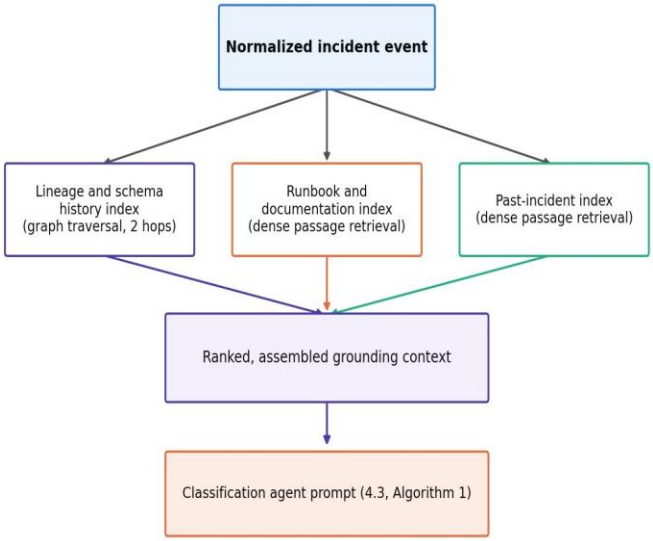


Figure 2. Retrieval-augmented context construction: lineage traversal, runbook retrieval and past-incident retrieval run in parallel and the ranked context is assembled into the classification prompt.

5.3 Constrained resolution tool use

When the guardrail authorizes autonomous handling, the resolution agent enters the bounded loop in Algorithm 2.

Algorithm 2: Constrained tool-use resolution loop

Input: classification c, grounded explanation x, incident event e,
vetted action library A(c) for category c, max steps N

Output: resolution outcome (RESOLVED, ESCALATED, or FAILED), ordered log of executed actions

```
log <- empty list
for step <- 1 to N:
    candidate <- LLM_SELECT_ACTION(c, x, e, A(c), log)
    if candidate is NONE:
        return (ESCALATED, log)          // no confident next action
    tier <- BLAST_RADIUS_TIER(candidate)    // Section 6
    if not GUARDRAIL_AUTHORIZES(tier, confidence(c)):
        return (ESCALATED, log)
    result <- EXECUTE(candidate)           // guardrail-mediated, Section 4.5
    log <- log + (candidate, result)
    if VERIFY_RESOLVED(e, result):
        return (RESOLVED, log)
    if result indicates candidate was incorrect:
        ROLLBACK(candidate)                // Section 6
        return (ESCALATED, log)
return (ESCALATED, log)                    // exhausted N steps without confirmed resolution
```

Steps 6 and 7 re-check authorization for every action, because a multi-step remediation can mix tiers, for example a diagnostic query followed by a configuration rollback, and confidence sufficient for the first action does not authorize the rest. The step budget N, set to four in our implementation, prevents a non-converging agent from acting indefinitely; exhausting it always escalates.

5.4 Confidence calibration and escalation policy

A model's self-reported confidence is not a reliable probability of correctness. TriageGPT calibrates the raw score from Algorithm 1 on a held-out validation set using isotonic regression over twenty equal-frequency bins, chosen over Platt scaling because miscalibration in our

benchmark was not consistently sigmoidal across categories. The guardrail then applies two thresholds: incidents above the high threshold may be resolved autonomously up to the tier their category permits; incidents between the thresholds go to a human together with the classification, explanation and evidence; and incidents below the minimum threshold are escalated with an explicit low-confidence flag.

6. Safety, Guardrails, and Governance for Autonomous Remediation

6.1 Blast-radius tiers

Every library action is assigned one of four tiers when added, and the authorization check in Algorithm 2 compares calibrated confidence with the tier's minimum (Table 2). Tier 4 actions are never autonomous, reflecting the position that calibrated confidence is not a sufficient basis for an action the system cannot undo, in line with the zero trust principle of granting capability only on verification [40].

Table 2. Blast-radius tiers for autonomous remediation actions.

Tier	Description	Example actions	Minimum calibrated confidence
Tier 1, diagnostic	Read-only; changes nothing	Fetch job logs, query lineage, dry-run a data quality check	None; always permitted during investigation
Tier 2, reversible-narrow	Quickly reversible, scoped to one task or partition	Restart a failed task, re-trigger one partition backfill, clear a stuck message	0.75
Tier 3, reversible-broad	Reversible, broader scope, longer recovery	Roll back a configuration change, pause a consumer, re-run a full DAG	0.90
Tier 4, irreversible or high blast radius	Not cleanly reversible or broadly impactful	Delete or overwrite a partition, change shared access control, notify external stakeholders	Never autonomous; always requires human approval

6.2 Rollback and audit logging

Every Tier 2 and Tier 3 library entry specifies a rollback procedure. Before execution, the guardrail logs the incident, classification and calibrated confidence, cited evidence, action and parameters, and rollback reference. After execution it monitors the incident's defining indicator for a confirmation window and, if the indicator has not recovered, rolls back and escalates. This log answers supervisory or audit questions about how an automated action was authorized, consistent

with AI management system expectations that organizations demonstrate how consequential decisions were made [43].

6.3 Mapping to external guidance

Table 3 maps the guardrail design to the OWASP guidance on LLM application risks, the ISO/IEC 42001 AI management system standard and the NIST generative AI risk management profile. The mapping shows how specific concerns are addressed; it is not a claim of certification, and deploying organizations must still assess compliance in their own context.

Table 3. Mapping from external safety and governance guidance to TriageGPT mechanisms.

Guidance source	Relevant concern	TriageGPT mechanism
OWASP Top 10 for LLM Applications, excessive agency [42]	More functionality or autonomy than a task requires	Blast-radius tiering; Tier 4 never autonomous (Section 6.1)
OWASP, prompt injection [42]	Untrusted retrieved content manipulating the agent	Retrieved context is evidence, not instructions; closed action library (Section 7.2)
OWASP, insufficient output validation [42]	Acting on unverified model output	Grounding verification against lineage before confidence is final (Algorithm 1)
ISO/IEC 42001 [43]	Accountability and continuous evaluation	Feedback loop (Section 4.6) and structured audit log (Section 6.2)
NIST AI RMF generative AI profile [44]	Human oversight and ability to reverse actions	Confidence-gated escalation (Section 5.4) and rollback (Section 6.2)

7. Data Pipeline Incident Taxonomy and Playbook Library

7.1 Incident taxonomy

Table 4 lists the fourteen incident classes, developed by reviewing common failures across orchestration, transformation, streaming and data quality tooling. Each is narrow enough that a grounded classification implies a specific remediation set, and each is shown with the highest tier its playbook reaches.

Table 4. Data pipeline incident taxonomy used in the reference implementation.

Incident class	Typical root cause	Representative signal source	Highest tier
Schema drift	Upstream field type, name or presence changes without notice	Schema history; data quality schema check	Tier 3
Late-arriving upstream data	Upstream source misses its delivery window	Orchestration SLA monitor	Tier 2
Upstream API failure or rate limiting	Third-party or internal API errors or throttles	Ingestion job error logs	Tier 2
Credential or secret expiry	Token, key or certificate expired or rotated incorrectly	Authentication errors	Tier 3
Resource exhaustion	Data volume or skew exceeds allocated compute	Executor logs and resource metrics	Tier 2
Orchestration scheduling deadlock	Circular or misconfigured dependencies stall a DAG	Scheduler state	Tier 2
Data quality threshold breach	Row count, null rate or distribution check fails	Data quality check result	Tier 2
Duplicate record ingestion	Upstream retry or ingestion bug duplicates records	Uniqueness check	Tier 3
Partition or data skew	Uneven keys let few tasks dominate runtime	Task duration metrics	Tier 1
Downstream consumer failure	Consumer cannot read the pipeline's output	Consumer error reports	Tier 2
Cost or budget anomaly	Cost deviates sharply from baseline	Billing and cost telemetry	Tier 1
Cloud storage connectivity timeout	Transient network or service issue on object storage	Job I/O errors	Tier 2
Orphaned or zombie task	Process does not terminate after completing or failing	Scheduler task-state monitor	Tier 2
Configuration drift	Deployed configuration differs from version control	Configuration diff against source control	Tier 3

7.2 Playbook and action library

Each class maps to a small, ordered set of candidate actions, each with a rollback procedure above Tier 1 (Table 5). Schema drift and duplicate ingestion reach Tier 3 because their most effective remedies change state broadly. No class has a Tier 4 playbook action; if an investigation reveals a

need for one, such as deleting a corrupted partition, Algorithm 2 finds no authorized candidate and escalates.

Table 5. Representative playbook actions from the vetted action library.

Incident class	Primary action	Tier	Rollback procedure
Late-arriving upstream data	Re-trigger the dependent task once upstream arrival is confirmed	Tier 2	Cancel the re-triggered task if data is still absent
Resource exhaustion	Restart with increased executor memory, within a pre-approved maximum	Tier 2	Revert memory allocation to its prior value
Orchestration scheduling deadlock	Clear the stalled task's lock and re-trigger the DAG	Tier 2	Restore prior scheduler state if the deadlock persists
Configuration drift	Redeploy the last known-good, version-controlled configuration	Tier 3	Keep the drifted configuration as a snapshot; redeploy it on regression
Downstream consumer failure	Notify the owning team of the identified change and pause automatic retries	Tier 2	Resume retries if notification alone resolves it
Cloud storage connectivity timeout	Retry with exponential backoff up to a pre-approved limit	Tier 1	Not applicable; retries are idempotent

8. Evaluation

The evaluation addresses classification accuracy, the effect of retrieval grounding, the frequency and safety of autonomous resolution, comparison with manual and rule-based practice, and latency and cost. Threats to validity are discussed in Section 10.

8.1 Benchmark environment

We built a reference pipeline instrumented at every layer: Apache Airflow for orchestration (task events, scheduler logs, SLA misses), Apache Kafka for streaming ingestion (consumer lag, error logs), Apache Spark for transformation (executor logs, resource and duration metrics), Great Expectations-style data quality checks [37], OpenLineage lineage events with schema history [36], and a cloud data warehouse (load status, consumer errors). We generated 2,800 labeled synthetic incidents, 200 per category, by injecting realistic failures and recording the telemetry with a ground-truth label and the lineage artifacts genuinely responsible. A disjoint set of 840 incidents (60 per category) was held out for testing; the remaining 1,960 populated the exemplar pool, the past-incident index and the calibration set.

8.2 Classification accuracy

Table 6 reports results on the held-out set. Top-one accuracy was 94.3 percent and top-three accuracy 98.6 percent. Per-category accuracy (Figure 3) ranged from 88.3 percent for cost or budget anomaly, whose signal often overlaps other categories' secondary symptoms, to 98.3 percent for credential or secret expiry, whose authentication errors are comparatively unambiguous. Calibration reduced expected calibration error from 0.087 to 0.021 (Figure 4), making the calibrated score substantially more trustworthy for driving escalation.

Table 6. Classification accuracy on the 840-incident held-out test set.

Metric	Value
Top-1 accuracy (all categories)	94.3%
Top-3 accuracy (all categories)	98.6%
Lowest per-category top-1 (cost or budget anomaly)	88.3%
Highest per-category top-1 (credential or secret expiry)	98.3%
Expected calibration error, raw confidence	0.087
Expected calibration error, after calibration	0.021

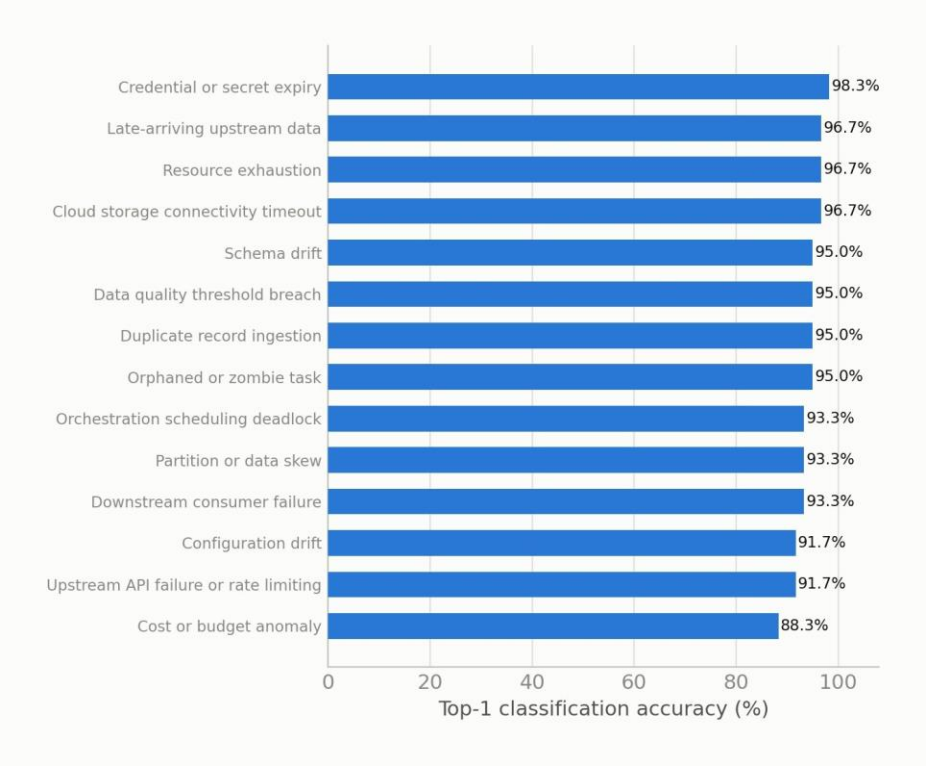


Figure 3. Top-1 classification accuracy by incident category on the 840-incident held-out test set.

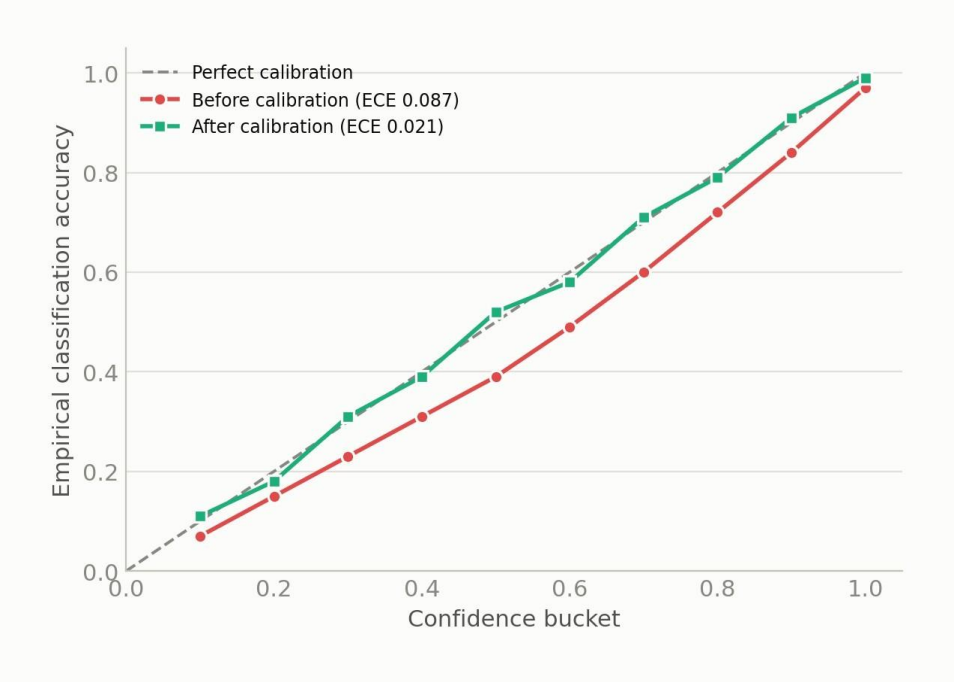


Figure 4. Reliability diagram: calibrated confidence versus empirical classification accuracy by confidence decile on the held-out test set.

8.3 Retrieval grounding and hallucination

In an ablation, the classifier was run on the test set with and without the context builder. Two large language model judges, prompted separately and required to agree, scored each explanation for factual attribution, meaning every cited artifact exists and is relevant, and for hallucination, meaning any cited artifact does not exist [34], [35]. Retrieval raised attribution accuracy by 26.7 percentage points and cut hallucination by nearly a factor of twelve (Table 7), consistent with the retrieval literature and validating the grounding requirement [13], [35].

Table 7. Retrieval grounding ablation.

Condition	Factual attribution accuracy	Hallucination rate
Without retrieval (incident event only)	68.4%	21.6%
With retrieval-augmented context	95.1%	1.8%

8.4 Autonomous resolution outcomes and guardrail safety

Of the 840 test incidents, 157 (18.7 percent) were escalated at classification for insufficient confidence, and 683 (81.3 percent) were authorized for resolution (Table 8). Of these, 601 (87.9 percent of attempts, 71.5 percent of the test set) were resolved autonomously and verified; 82

escalated because no confident action was found or a tier exceeded the authorized level. Of 650 executed actions, 4 (0.6 percent) were later found incorrect, in each case because the true category differed from the initial classification in a way the grounding check did not catch, and all 4 were detected and reversed by the confirmation-window rollback, at a median of 11 seconds. No Tier 4 action was proposed or executed, and no authorized action caused an unrecovered production impact.

Table 8. Autonomous resolution outcomes on the 840-incident held-out test set.

Outcome	Count	Share
Escalated at classification (confidence below threshold)	157	18.7%
Resolved autonomously (verified)	601	71.5%
Escalated during resolution (no confident action or tier denied)	82	9.8%
Incorrect autonomous action, detected and rolled back	4 of 650 actions	0.6% of actions

8.5 Comparative deployment trial

We compared TriageGPT with manual on-call triage and static alert-to-runbook automation over a twelve-week trial in which the same sequence of injected and organically occurring incidents was handled by each approach in separate, isolated windows (Table 9). Rule-based automation covered six of fourteen categories, those expressible as a fixed keyword or threshold match, leaving eight, including schema drift and cost anomalies, to manual triage. TriageGPT gave at least a grounded Tier 1 assessment for every category and full autonomous resolution for the twelve categories whose playbooks reach a tier authorized by the observed confidence distribution. Two independent reviewers scored audit evidence completeness against a postmortem checklist covering notification, supporting evidence, action rationale and verification, in the blameless postmortem tradition [41]; TriageGPT's structured log answered more of the checklist by construction.

Table 9. Manual on-call triage, rule-based automation and TriageGPT over a twelve-week comparative trial.

Metric	Manual on-call	Rule-based automation	TriageGPT
Median time to acknowledge	6.2 minutes	1.1 minutes (rule match only)	4.1 seconds
Median time to resolution	41 minutes	27 minutes	9.8 minutes
Incident classes with any automated handling	0 of 14	6 of 14 (43%)	14 of 14 (100%, Tier 1 minimum)
Audit evidence completeness (0 to 100 rubric)	51	64	92

8.6 Latency and cost

Median classification latency was 3.6 seconds (99th percentile 9.8 seconds). For autonomously resolved incidents, the median full resolution loop took 14.2 seconds (99th percentile 31.5 seconds): 0.08 seconds for normalization, 1.4 seconds for context construction, 2.1 seconds for classification and grounding verification, 3.9 seconds for action selection and execution, and 6.7 seconds for the guardrail confirmation window. The confirmation window is the largest contributor by design, since it waits for the defining indicator to recover rather than assuming success. Median inference cost was 0.084 US dollars per triaged incident and 0.11 US dollars per autonomously resolved incident.

9. Illustrative Use Cases

The following representative scenarios are constructed from the taxonomy and playbook library rather than reported from a named deployment.

Schema drift. An upstream team changes a customer identifier from integer to string without notice, and the next ingestion run fails a schema check rather than silently corrupting tables. The context builder retrieves the field-level change and timestamp from schema history, the schema drift runbook and two similar past incidents. The agent classifies the incident as schema drift with calibrated confidence 0.94, citing the field, source and timestamp, each verified against lineage. Because 0.94 exceeds the Tier 3 threshold, the resolution agent redeploys the downstream transformation's last known-good configuration extended to the new type; the guardrail confirms the next run and check pass, and no engineer is paged.

Upstream API rate limiting. A third-party provider returns rate-limit errors during a traffic spike, a pattern that resembles both credential expiry and an outage. Grounded in the specific error codes, the agent correctly identifies rate limiting, but because the evidence cannot show whether the limit will lift soon, calibrated confidence is 0.71, below the Tier 2 threshold of 0.75. The incident goes

to an engineer with the full case assembled, who confirms within about a minute that a reduced ingestion rate, not retry with backoff, is correct, showing the middle confidence band routing genuine ambiguity to a human.

Resource exhaustion during a backfill. A multi-month backfill runs with default executor memory sized for daily volume, and executors fail with out-of-memory errors. Grounded in executor logs and resource metrics, the agent classifies resource exhaustion at 0.89, above the Tier 2 threshold, and the resolution agent restarts the task with more memory, capped at the pre-approved maximum. The restarted task completes. Had it still failed, the agent would have escalated after its step budget rather than raising memory beyond what the playbook authorizes.

10. Discussion, Limitations, and Threats to Validity

Fixed taxonomy. The classifier is bounded by the fourteen categories: a novel failure is forced into the nearest one, and the grounding check verifies that cited artifacts are real, not that the category is the best fit. Because every benchmark incident belonged to a category by design, reported accuracy measures quality within a closed taxonomy. Deployments should review low-confidence and corrected incidents through the feedback loop to identify new categories and playbooks.

Confidence and autonomy trade-off. Lowering the thresholds in Table 2 would resolve more incidents autonomously at greater risk of incorrect action; raising them escalates more incidents that could have been resolved safely. Our thresholds were informed by the calibration curve, but the right values depend on each organization's relative cost of false escalation and false action.

Coverage. TriageGPT cannot act on systems that fail silently or emit no parseable alert, and grounding depends on complete, current lineage. Artifacts outside lineage coverage either lower confidence for otherwise correct classifications or are invisible to retrieval. We regard this as an honest consequence of a safety design that depends on verifiable ground truth.

Governance. Standing authority to execute Tier 2 and Tier 3 actions makes the system itself governance-sensitive and enlarges the attack surface relative to an advisory tool. The closed action library and the Tier 4 prohibition are the main structural defenses, but organizations should apply production-grade access control to who can add actions, change tiers or adjust thresholds, since a compromised configuration would undermine the reported safety properties.

Threats to validity. The evaluation uses one reference pipeline and a synthetic incident corpus rather than a production deployment, so stacks built differently, for example around batch file transfer, or incident mixes outside the taxonomy may yield different results. Attribution and hallucination figures rely on agreeing automated judges rather than full human review. The deployment trial replayed the same incident sequence in separate windows rather than a concurrent, randomized production comparison, so effects such as accumulated fatigue are not captured. The safety results reflect our thresholds, action library and benchmark of 2,800 labeled

incidents, and should be replicated at larger scale and over longer horizons before being treated as general guarantees.

11. Conclusion and Future Work

TriageGPT for DataOps classifies data pipeline incidents against a purpose-built taxonomy, grounds its diagnoses in pipeline-specific evidence, and executes a bounded, guardrail-gated set of remediations, treating autonomy as something earned through calibrated confidence and bounded blast radius. On a benchmark of 2,800 synthetic incidents it achieved 94.3 percent top-one accuracy, raised factual attribution from 68.4 to 95.1 percent while reducing hallucination from 21.6 to 1.8 percent, and resolved 71.5 percent of test incidents autonomously with a 0.6 percent incorrect-action rate, all reversed by the guardrail. Over a twelve-week trial it reduced median time to resolution from 41 minutes under manual practice, and 27 minutes under rule-based automation, to 9.8 minutes. These results rest on a closed taxonomy, organization-specific thresholds, instrumented systems and a single synthetic benchmark.

Future work includes semi-automated taxonomy discovery by clustering escalated and corrected incidents; correlating near-simultaneous alerts from one upstream cause into a single incident; fitting thresholds to each organization's cost of false escalation and false action; adding pre-execution simulation or an independent verification agent so that a carefully scoped subset of Tier 4 actions might eventually be authorized; and a multi-organization study on real production incidents. We view TriageGPT not as a replacement for the on-call engineer, but as a way to resolve routine incidents with verifiable confidence and to hand the rest to a human with a complete, evidence-grounded case.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2017, pp. 5998-6008.

[2] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, et al., "Language Models are Few-Shot Learners," in Proc. Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020, pp. 1877-1901.

[3] OpenAI, "GPT-4 Technical Report," arXiv:2303.08774, 2023.

[4] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al., "Training Language Models to Follow Instructions with Human Feedback," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2022.

[5] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2022.

[6] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha, "A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications," arXiv:2402.07927, 2024.

[7] Y. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," in Proc. Int. Conf. Learning Representations (ICLR), 2023.

[8] T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language Models Can Teach Themselves to Use Tools," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2023.

[9] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language Agents with Verbal Reinforcement Learning," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2023.

[10] OpenAI, "Function calling," OpenAI API Documentation. Available: https://developers.openai.com/api/docs/guides/function-calling

[11] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W. White, D. Burger, and C. Wang, "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework," arXiv:2308.08155, 2023.

[12] LangChain, "LangChain: Building Applications with LLMs Through Composability," Official Documentation. Available: https://python.langchain.com

[13] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Kuttler, M. Lewis, W. Yih, T. Rocktaschel, S. Riedel, and D. Kiela, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in Proc. Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020, pp. 9459-9474.

[14] A. Singh, A. Ehtesham, S. Kumar, et al., "Agentic Retrieval-Augmented Generation: A Survey on Agentic RAG," arXiv:2501.09136, 2025.

[15] "Retrieval-Augmented Generation (RAG) and Large Language Models (LLMs) for Enterprise Knowledge Management and Document Automation: A Systematic Literature Review," Applied Sciences, vol. 16, no. 1, article 368, 2026.

[16] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An Online Log Parsing Approach with Fixed Depth Tree," in Proc. IEEE Int. Conf. Web Services (ICWS), 2017, pp. 33-40.

[17] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning," in Proc. ACM SIGSAC Conf. Computer and Communications Security (CCS), 2017, pp. 1285-1298.

[18] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J.-G. Lou, M. Chintalapati, F. Shen, and D. Zhang, "Robust Log-Based Anomaly Detection on Unstable Log Data," in Proc. ACM Joint Meeting European Software Engineering Conf. and Symp. Foundations of Software Engineering (ESEC/FSE), 2019, pp. 807-817.

[19] V.-H. Le and H. Zhang, "Log-Based Anomaly Detection with Deep Learning: How Far Are We?," in Proc. IEEE/ACM 44th Int. Conf. Software Engineering (ICSE), 2022.

[20] N. Sun, T. Zhao, and D. Pei, "HotSpot: Anomaly Localization for Additive KPIs with Multi-Dimensional Attributes," IEEE Access, vol. 6, pp. 10909-10923, 2018.

[21] L. Wu, J. Tordsson, E. Elmroth, and O. Kao, "MicroRCA: Root Cause Localization of Performance Issues in Microservices," in Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS), 2020.

[22] J. Lin, P. Chen, and Z. Zheng, "Microscope: Pinpoint Performance Issues with Causal Graphs in Micro-Service Environments," in Proc. Int. Conf. Service-Oriented Computing (ICSOC), Lecture Notes in Computer Science, vol. 11236, Springer, 2018, pp. 3-20.

[23] J. Soldani and A. Brogi, "Failure Diagnosis in Microservice Systems: A Comprehensive Survey and Analysis," ACM Transactions on Software Engineering and Methodology, 2025.

[24] Y. Chen, H. Xie, M. Ma, Y. Kang, X. Gao, L. Shi, Y. Cao, X. Gao, H. Fan, M. Wen, J. Zeng, S. Ghosh, X. Zhang, C. Zhang, Q. Lin, S. Rajmohan, D. Zhang, and T. Xie, "Automatic Root Cause Analysis via Large Language Models for Cloud Incidents," in Proc. 19th European Conf. Computer Systems (EuroSys), 2024, pp. 674-688.

[25] D. Roy, X. Zhang, R. Bhave, C. Bansal, P. Las-Casas, R. Fonseca, and S. Rajmohan, "Exploring LLM-based Agents for Root Cause Analysis," arXiv:2403.04123, 2024.

[26] J. Jiang, W. Lu, J. Deng, X. Wang, N. Zhang, X. Yu, Y. Wu, Q. Zhu, W. Chen, et al., "Xpert: Empowering Incident Management with Query Recommendations via Large Language Models," in Proc. IEEE/ACM 46th Int. Conf. Software Engineering (ICSE), 2024.

[27] P. Jin, S. Zhang, M. Ma, H. Li, Y. Kang, L. Li, Y. Liu, B. Qiao, C. Zhang, P. Zhao, S. Rajmohan, Q. Lin, and D. Zhang, "Assess and Summarize: Improve Outage Understanding with Large Language Models," arXiv:2305.18084, 2023.

[28] Y. Liu, X. Wang, et al., "OpsEval: A Comprehensive Benchmark Suite for Evaluating Large Language Models' Capability in IT Operations Domain," arXiv:2310.07637, 2023.

[29] "A Survey of AIOps in the Era of Large Language Models," ACM Computing Surveys, 2025.

[30] 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, and J. Tang, "AgentBench: Evaluating LLMs as Agents," in Proc. Int. Conf. Learning Representations (ICLR), 2024.

[31] 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, and M. Sun, "ToolLLM: Facilitating Large Language Models to Master 16000+ Real-World APIs," in Proc. Int. Conf. Learning Representations (ICLR), 2024.

[32] Z. Guo, S. Cheng, Y. Wang, P. Li, and Y. Liu, "StableToolBench: Towards Stable Large-Scale Benchmarking on Tool Learning of Large Language Models," in Findings of the Association for Computational Linguistics (ACL), 2024.

[33] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan, "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?," in Proc. Int. Conf. Learning Representations (ICLR), 2024.

[34] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica, "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena," in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2023.

[35] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, and T. Liu, "A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions," ACM Transactions on Information Systems, 2025.

[36] OpenLineage, "OpenLineage: An Open Standard for Lineage Metadata Collection," Official Documentation. Available: https://openlineage.io/

[37] Great Expectations, "GX Core: Open Source Data Quality Platform," Official Documentation. Available: https://greatexpectations.io/

[38] S. Shankar, R. Garcia, J. M. Hellerstein, and A. G. Parameswaran, "Towards Observability for Production Machine Learning Pipelines [Vision Paper]," Proceedings of the VLDB Endowment, vol. 15, no. 13, pp. 4015-4023, 2022.

[39] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," in Proc. 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2012, pp. 15-28.

[40] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Eds., Site Reliability Engineering: How Google Runs Production Systems. Sebastopol, CA: O'Reilly Media, 2016.

[41] PagerDuty, "The Blameless Postmortem," Incident Response Documentation. Available: https://postmortems.pagerduty.com/culture/blameless/

[42] OWASP GenAI Security Project, "OWASP Top 10 for LLM Applications 2025," OWASP Foundation, 2025. Available: https://genai.owasp.org/llm-top-10/

[43] International Organization for Standardization, ISO/IEC 42001:2023, Information Technology, Artificial Intelligence, Management System, 2023.

[44] National Institute of Standards and Technology, "Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile," NIST AI 600-1, Jul. 2024.

2026 by the Authors. This Article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/)