International Journal of AI, BigData, Computational and Management Studies
Noble Scholar Research Group | Volume 6, Issue 4, PP 133-141, 2025
ISSN: 3050-9416 | https://doi.org/10.63282/3050-9416.IJAIBDCMS-V6I4P115

Original Article

AI/ML Powered Intelligent Root Cause Analysis and Automated Remediation for Multi System Data Integrity Issues

Vineeth Kumar Reddy Mittamidi
Application Support engineer, TCS, North Carolina, USA.

Received On: 20/09/2025 ***Revised On: 23/10/2025*** ***Accepted On: 03/11/2025*** ***Published On: 14/11/2025***

Abstract: *Enterprises increasingly rely on complex data ecosystems that span operational databases, event streams, microservices, batch ETL workflows, data warehouses and analytics products. In this environment data integrity incidents rarely originate from a single component. They emerge from interacting failure modes such as schema drift, late or duplicated events, inconsistent reference data, partial writes, replay defects and misaligned transformation logic. These incidents propagate across systems and can silently corrupt business reporting, customer experiences and compliance artifacts. While observability has improved visibility, diagnosis and recovery still depend heavily on human expertise and manual correlation across logs, traces, metrics and change histories. This paper proposes a unified architecture for AI and ML powered intelligent root cause analysis and automated remediation that is tailored for multi system data integrity issues. The approach combines contract based data quality checks grounded in established data quality dimensions [1], pipeline quality taxonomies and root causes mined from developer evidence [5], provenance and lineage reasoning for debugging and trust [2][4], dependency aware localization adapted from microservice RCA [9][10] and incident knowledge retrieval from real cloud incident investigations [11]. A remediation framework applies graduated autonomy and policy gating inspired by agentic AIOps architectures [13] while using large language models only for evidence synthesis and plan drafting under strict guardrails [12]. We describe incident taxonomy, graph based RCA method, remediation catalog design and evaluation metrics for detection, localization and recovery. The proposed system reduces mean time to detection and mean time to recovery while improving auditability and safety of corrective actions in enterprise data platforms.*

Keywords: *Data Integrity, Data Quality, AIops, Root Cause Analysis, Automated Remediation, Data Pipelines, Provenance, Lineage, Observability.*

1. Introduction

Modern enterprises run as data driven systems. A single customer purchase can produce a payment record in a transactional database, a set of domain events on a streaming bus, a fulfillment workflow in an orchestration system and downstream feature sets for personalization, fraud detection and forecasting. Each stage may be owned by a different team and implemented using different technology. This creates a multi system data supply chain. Integrity failures in any stage can cascade. A bad mapping table can break joins, a schema change can cause silent nulls, a late event can break aggregates and a replay can double count revenue. Unlike classic availability incidents, integrity incidents may not trigger immediate user errors. They can remain latent until an analyst notices a discrepancy or a model degrades in production. When that happens the time window of impact can be large and reprocessing can be expensive.

Many organizations adopt data quality rules, dashboards and alerting. These are necessary but often insufficient. Rules tend to cover obvious cases such as missing partitions and null spikes. Yet integrity failures increasingly arise from complex interactions, for example a deploy that changes an identifier format combined with a consumer that assumes a fixed length and a downstream cleaning step that drops invalid rows. The result can be a subtle shift in distribution that still passes basic rule checks. Research on data quality shows that quality is multidimensional and context dependent [1]. A correct design must therefore encode consumer expectations and not only producer checks.

At the same time, operational scale has increased. Data platforms generate massive volumes of telemetry. Engineers can capture pipeline durations, task retries, storage latency and service error rates. They can also parse logs and build traces across microservices. But converting these signals into a root cause and a safe action is still difficult. Incident investigations

in large cloud services have shown that extracting and reusing root cause knowledge can improve reliability practices [11]. Recent work also shows that LLM driven assistants can speed up incident analysis by synthesizing evidence and suggesting investigation steps [12]. However, autonomy without governance can be dangerous, particularly for data remediation where replays, deletes and backfills may create irreversible business impact.

This paper proposes a unified architecture and method for intelligent RCA and automated remediation for multi system data integrity issues. The key idea is to fuse three kinds of structure. First is integrity structure captured by contracts and dimensions of data quality [1]. Second is data flow structure captured by lineage and provenance graphs which are well studied for debugging and trust [2][4]. Third is dependency and causality structure from distributed systems RCA which can rank candidate root causes based on evidence and causal relationships [9][10]. The system produces ranked hypotheses with explanations and then recommends or executes remediation steps under policy gating and graduated autonomy [13].

The main contributions are as follows:

- A practical definition of multi system data integrity incidents and an operational taxonomy tied to contracts and failure propagation.
- A reference architecture that combines contract checks, data observability signals, provenance lineage graphs and incident knowledge retrieval.
- A hybrid RCA approach that uses lineage constrained search and evidence scoring with optional causal refinement for high severity cases.
- A remediation framework with risk tiered action catalog, policy based autonomy gating and verification and learning loops.

The remainder of this paper is organized as follows. Section II reviews relevant work in data quality, provenance, pipeline quality, RCA and AIOps. Section III defines the problem and goals. Section IV introduces the incident taxonomy. Section V describes the reference architecture. Section VI details the RCA method. Section VII presents the remediation framework. Section VIII outlines prototype implementation. Section IX discusses evaluation. Section X discusses lessons and limitations. Section XI concludes.

2. Background and Related Work

2.1. Data quality and integrity

Data quality has been studied for decades. A key insight is that quality means fitness for use and includes multiple dimensions beyond accuracy such as completeness, consistency, timeliness and interpretability [1]. For multi system integrity, consistency and completeness are frequent failure points. Consistency includes agreement across representations, agreement across copies and agreement with defined rules. Completeness includes missing records, missing attributes and missing categories. Timeliness includes freshness and latency. Interpretability includes correct semantics and units. An architecture for integrity RCA must represent which dimensions matter for which consumers and must treat violations as contractual incidents, not only technical failures.

2.2. Provenance and Lineage for Debugging and Trust

Provenance describes the history of data and how outputs were produced from inputs. A survey of provenance in e science describes uses in debugging, validation and audit [2]. Provenance in scientific workflows emphasizes that workflows produce rich execution traces that enable explanation and reproduction [3]. A survey of provenance in databases describes why, how and where provenance and relates them to debugging, annotation propagation and trust [4]. These works motivate treating lineage and provenance as first class artifacts for integrity RCA. When a downstream dataset fails a contract, lineage can reveal the upstream transformations that could have introduced the defect. Provenance can also reveal which exact run and version produced the output.

2.3. Data Pipeline Quality and Data Observability

Data pipelines are often fragile and developers report difficulties in integration, ingestion and cleaning. A study on data pipeline quality introduces taxonomy of factors that influence pipeline quality and mines root causes of data related issues, highlighting incorrect data types and cleaning stage defects as common causes [5]. Observability focused work for data quality proposes frameworks to collect signals and implement responsible monitoring for data quality aware analytics at scale [6]. These studies support the idea that integrity management requires both engineering discipline and systematic observability of data specific signals.

2.4. Root cause Analysis in Distributed Systems

Distributed systems and microservices have motivated research in automated RCA. MicroRCA localizes root causes of performance issues by using service interaction graphs and metrics correlations to infer where delays originate [9]. Causal approaches such as CausalRCA build weighted causal graphs from metrics and infer root cause metrics for microservice applications [10]. While these works focus on performance and availability, their core idea is dependency aware localization. Integrity incidents also propagate along dependencies, but the dependencies are data flow and transformation dependencies rather than only call graphs. Therefore the methods can be adapted by using lineage graphs as the dependency backbone and by using data quality metrics as primary observables.

2.5. Incident Knowledge Mining and AIOps Assistance

AIOps systems often include ingestion of telemetry, anomaly detection, event correlation and diagnosis. Mining root cause knowledge from cloud service incident investigations shows that incident artifacts can be mined to

build retrieval and recommendation systems and that doing so can improve diagnosis workflows [11]. LLM driven systems such as RCACopilot show that assistants can automate parts of incident RCA by aggregating diagnostic data, predicting root cause categories and recommending actions for cloud incidents [12]. A framework for agentic AIOps proposes that autonomy should be structured around architectural components and governance so that systems can detect, decide and act safely [13]. Survey work in AIOps in the era of LLMs synthesizes progress and challenges across these topics [14]. These works motivate the design of a human centered system that provides recommendations with evidence and that supports limited autonomous action under policy.

2.6. Log parsing as a Source of Execution Evidence

Pipeline execution logs encode detailed failure information. Parsing unstructured logs into templates enables statistical analysis and correlation. Spell proposes streaming parsing of system event logs using a longest common subsequence approach [7]. Drain proposes an online log parsing approach using a fixed depth tree and achieves strong accuracy and performance [8]. These methods are useful for integrity RCA because many integrity incidents manifest as changes in log template patterns such as new parse errors, increased rejected records or new validation failures.

3. Problem Definition

3.1. Multi System Data Integrity Incident

We define a multi system data integrity incident as an event where one or more datasets, features or derived views violate declared consumer expectations due to faults that span at least two domains among ingestion, transformation, storage, serving, reference data management, orchestration and upstream application behavior. A violation can be a hard constraint violation, for example a uniqueness check failure, or a statistical expectation violation, for example a distribution drift beyond a defined tolerance. The incident is multi system when the propagation path crosses system boundaries, for example from an application service to a stream to a warehouse or from a reference table to multiple transforms.

3.2. Challenges

The first challenge is ambiguity. Many upstream changes can influence downstream metrics. Without structure, the hypothesis space is large. The second challenge is delay. Integrity issues can propagate with time lag. Detection might occur hours after the root cause. The third challenge is noise. Seasonal changes and backfills can resemble incidents. The fourth challenge is risk. Remediation actions can be costly and can introduce further inconsistency if they are not idempotent and bounded.

3.3. Objectives

The system should reduce mean time to detection and mean time to recovery. It should produce ranked root cause hypotheses with evidence and should propose safe remediation steps. It should preserve auditability. It should support both batch and streaming pipelines and should operate across heterogeneous storage engines. It should support graduated autonomy where low risk actions can be automated and high risk actions require approval [13].

3.4. Non Goals

The system is not intended to replace strong engineering practices such as well designed schemas and idempotent pipelines. It is also not intended to apply full autonomy for all actions. Instead it should provide assistance and limited safe automation. It should not rely on heavyweight deep learning models that require extensive labeled data. Lightweight models and structured reasoning are preferred for practicality.

4. Incident Taxonomy for Integrity Operations

Integrity incidents are easier to manage when symptoms map to a small set of patterns. We propose an operational taxonomy that classifies incidents by violated contract dimension and by typical propagation pattern. The taxonomy is inspired by data quality dimensions [1] and by observed pipeline issue root causes and processing problem areas [5]. Each class links to typical signals, likely root causes and candidate remediation actions.

4.1. Completeness and Volume Anomalies

These include missing partitions, missing batches, drops in row counts, missing keys, missing event types and partial loads. Likely root causes include ingestion failures, upstream service outages, misconfigured filters, wrong join types that drop rows and silent parse failures after schema changes. Pipeline quality studies highlight that ingestion and integration tasks account for a large share of developer pain which aligns with this incident class [5]. Typical remediation includes replaying a bounded window, re running a failed task and backfilling missing partitions after verifying idempotency.

4.2. Consistency and Referential Anomalies

These include broken referential relationships, inconsistent identifier mappings, duplicates that violate uniqueness, mismatched totals between systems of record and derived tables and conflicts between copies of reference data. Root causes include reference data drift, out of order updates, eventual consistency windows, flawed merge logic and non deterministic deduplication. Remediation may include restoring a reference snapshot, running a reconciliation job, applying deterministic dedupe rules and re computing affected aggregates.

4.3. Timeliness and Freshness Anomalies

These include stale dashboards, late arriving events, lag increases in streaming consumers and increased end to end latency. Root causes include queue backlogs, throttling, resource contention, partition skew and scheduling misconfiguration. Remediation often begins with containment

such as pausing downstream refresh and then scaling consumers or rebalancing partitions.

4.4. Schema and Semantic Drift

These include new fields, missing fields, type changes, nullability changes and unit conversion mismatches. Pipeline quality research identifies incorrect data types as a frequent root cause of data related issues [5]. Root causes include uncontrolled schema evolution, incompatible serialization changes and mismatched schema registry versions. Remediation includes rolling back a schema change, updating parsers and reprocessing data with the correct schema mapping.

4.5. Transformation Logic Defects

These include incorrect aggregates, misapplied business rules, metric definition drift, incorrect windowing for streams and distribution drift without hard rule violations. Root causes include code changes, configuration changes and dependency version changes. Remediation includes rolling back a deploy, patching the transformation and rebuilding affected derived tables.

4.6. Access and Governance Related Integrity Failures

These include partial data due to permission changes, masked fields causing join failures and inconsistent application of policies across environments. Remediation includes reverting access policy changes and ensuring consistent enforcement.

5. Reference Architecture

The proposed architecture is designed to be implementable with common enterprise tooling while remaining conceptually unified. The core design choice is to treat data contracts and lineage as primary structures for reasoning.

5.1. Layer 1 Instrumentation and Signal Collection

The system collects four categories of signals.

- Data signals such as counts, freshness, schema snapshots, null rates, uniqueness rates and constraint violation counts. These signals are computed per dataset and per contract scope.
- Pipeline execution signals from orchestrators such as run start times, durations, retries, task failures and resource usage.
- Service signals from microservices that produce or consume data such as request rates, error rates and dependency traces.
- Change signals such as code deploys, configuration updates, schema registry changes and access policy updates.

Pipeline and service logs are parsed into structured templates and parameters using online parsers such as Spell and Drain [7][8]. The system stores template counts and parameter distributions over time which enables detection of new error signatures and spikes of known signatures.

5.2. Layer 2 Contract and Expectation Management

Contracts represent consumer expectations and encode thresholds and statistical bounds for quality dimensions [1]. A contract can specify that a dataset must arrive within a freshness window, that row counts must stay within an expected band relative to historical seasonality and that key fields must be non null and unique. Contracts can be scoped by tenant, region and product line. Each contract produces a set of checks and also provides context for RCA by indicating which upstream attributes are critical for the consumer.

5.3. Layer 3 Provenance and Lineage Graph

The system maintains a lineage graph with static and runtime aspects. Static lineage is derived from pipeline definitions, SQL DAGs, stream processing graphs and job configuration. Runtime provenance links each dataset version to specific runs and inputs. Provenance surveys emphasize that such information supports debugging and auditing [2][4]. In the graph nodes represent datasets, transformations, schema versions, code versions, reference tables and configuration artifacts. Edges represent produces, consumes, validates and configures relationships. Each edge stores metadata such as join keys, filters and mapping versions.

5.4. Layer 4 Detection and Correlation Engine

Detection produces candidate incidents. It uses lightweight models including robust z scores, seasonal baselines, change point detectors and isolation based detectors over quality metrics. It also uses schema diff detection and constraint violation monitoring. Each detection emits a symptom signature that includes dataset, violated dimension, severity, time window and deviation vector.

Correlation links symptoms to events. It aligns incident windows with deployments, schema changes, policy changes and pipeline run anomalies. Incident knowledge mining research suggests that correlating with historical incident patterns improves diagnosis efficiency [11].

5.5. Layer 5 Root Cause Analysis Engine

The RCA engine produces ranked hypotheses. It uses lineage constrained search to limit candidates, evidence scoring to rank them and optional causal refinement for complex cases. The dependency aware localization approach is analogous to MicroRCA but uses lineage edges as dependency edges and uses data quality signals and pipeline execution signals as observables [9]. Causal refinement uses a constrained subgraph approach inspired by CausalRCA where causal learning is applied to candidate nodes and their neighbors rather than to the entire platform [10].

5.6. Layer 6 Remediation Planners and Execution Gateway

The remediation planner proposes actions from a catalog. Each action has preconditions, rollback steps, risk level and

expected verification checks. The policy engine enforces autonomy level and approvals, aligning with agentic AIOps governance principles [13]. Execution uses runbooks and orchestrated workflows. LLM components are used only for summarizing evidence and drafting human readable plans under guardrails, building on the idea of LLM assisted incident RCA [12].

5.7. Layer 7 Verification and Learning Store

After actions, the system verifies contract recovery and stores the incident artifact including symptom, chosen root cause, evidence and outcome. This supports retrieval and continuous improvement similar to incident knowledge mining systems [11].

6. Graph Based Hybrid RCA Method

We model integrity RCA as a ranked hypothesis problem over a graph that unifies data flow structure and operational evidence.

6.1. Integrity Graph

Let G be a directed graph. Nodes represent artifacts including datasets, transformations, schemas, code versions, reference tables and configurations. Edges represent relationships such as produces, consume, transforms, validates and configures. Static lineage adds produces and consumes edges. Runtime provenance adds run specific edges and version edges. Provenance models motivate representing these relations to support debugging and trust [2][4].

6.2. Symptom Signature

A symptom S is defined as (a, d, t0, t1, m) where a is the affected dataset artifact, d is the violated contract dimension, t0 and t1 define the incident window and m is a deviation vector such as drop in counts, spike in null rate, increase in duplicates or increased freshness lag.

6.3. Candidate Generation via Lineage Constrained Search

The candidate set C is generated by traversing upstream produces edges from a within a bounded depth. The depth is chosen based on typical pipeline hop count and can be increased for complex graphs. We also include co temporal change nodes that touch any node along the upstream path, such as deploys, schema registry updates and policy changes. This step reduces the hypothesis space and avoids searching unrelated systems.

6.4. Evidence Features

For each candidate c in C we compute evidence features.

1. Temporal alignment feature that measures how close candidate events are to t0.
2. Telemetry anomaly feature derived from metrics and run states such as task retries, failures and latency spikes.
3. Schema drift feature based on schema diffs and serialization changes.
4. Log signature feature based on template novelty and spike rates using Spell and Drain parsing [7][8].
5. Data contract proximity feature that measures how directly the candidate influences the violated dimension, for example a candidate that modifies a key field is more relevant for a uniqueness violation.
6. Historical similarity feature derived from incident knowledge retrieval that searches past incident artifacts for similar symptom patterns and evidence, inspired by cloud incident knowledge mining [11].

6.5. Ranking

We compute a confidence score for each candidate using a weighted sum or a lightweight learning to rank model trained on labeled incidents. Features are normalized and weights can be tuned per incident class. The output is a ranked list of candidates with explanations. For many incidents the top candidate can be sufficient for action planning.

6.6. Causal Refinement for High Ambiguity Incidents

When multiple candidates have similar scores, the system optionally performs causal refinement. It extracts a subgraph containing the top N candidates and their immediate neighbors. It then applies causal graph learning and root cause inference in the style of microservice causal RCA but restricted to the small subgraph [10]. This reduces complexity and improves robustness. The causal stage aims to distinguish correlated symptoms from causal precursors.

6.7. Explainability and Reporting

Each hypothesis includes a short explanation that cites evidence. For example the report can state that a schema registry update at a certain time coincides with a spike in parse error templates and a rise in null rate for a critical field. Explainability is essential because remediation actions can be risky and require trust.

6.8. Extending RCA with Inferred Constraints

In some environments explicit integrity constraints are incomplete. Research on automatically inferring missing database constraints from application behavior shows that inferred constraints can strengthen integrity checking [15]. The proposed architecture can incorporate inferred constraints as additional contract checks for datasets derived from application tables. This improves detection and also provides additional signals for RCA when constraint violations spike.

7. Automated Remediation Framework

Automated remediation for data integrity differs from remediation for availability because actions often modify stored data. Therefore the system must enforce strong safety and audit controls.

7.1. Remediation Design Principles

The remediation subsystem follows five principles.

- Minimize further corruption. Actions must not amplify inconsistency across systems.
- Preserve auditability. Every action must be logged with inputs, approvals and outcomes.
- Prefer reversible actions. When uncertainty is high the system should choose actions that can be rolled back.
- Respect SLAs and business priorities. Some datasets can tolerate staleness while others cannot.
- Use graduated autonomy. Actions are selected based on risk and confidence with governance aligned to autonomy frameworks [13].

7.2. Remediation Catalog and Tiers

We classify actions into four tiers.

- Tier 0 Informational actions. Create alerts, annotate dashboards, open tickets and notify owners.
- Tier 1 Containment actions. Pause downstream consumers, route reads to a last known good dataset version, apply throttling to prevent compounding errors.
- Tier 2 Correction actions. Replay a bounded time window, reprocess failed partitions, rebuild materialized views, restore a reference data snapshot.
- Tier 3 Structural fix actions. Roll back a deploy, revert a schema registry change, patch a transformation defect, modify pipeline logic and then backfill affected windows.

7.3. Policy Based Autonomy Gating

Each action in the catalog declares risk level, blast radius factors and preconditions. The policy engine maps these to required approvals. Low risk containment actions can be executed automatically if confidence exceeds a threshold and the action is reversible. Higher risk correction and structural actions require human approval. This aligns with agentic AIOps frameworks that emphasize controlled autonomy across layers including data and applications [13].

7.4. LLM Assisted Plan Drafting Under Guardrails

LLM systems for incident RCA can summarize evidence and propose investigation and remediation steps [12]. In our design the LLM is used only to draft human readable narratives, to explain evidence and to compose a remediation proposal that references specific deterministic checks and runbook steps. The final plan must pass policy validation. The execution gateway does not run free form LLM instructions. Instead it triggers curated runbooks with explicit parameters.

7.5. Verification, Rollback and Learning

After execution the system verifies contract recovery. Verification includes re running contract checks, validating row counts and checking that duplicates and null rates return to normal bands. If verification fails the system triggers rollback if available and escalates to human operators. The incident artifact is stored with the final root cause and outcome for future retrieval and model tuning. This supports the continuous learning loop recommended by incident knowledge mining approaches [11].

8. Prototype Implementation Guidance

A prototype can be implemented in an enterprise using existing tools with minimal disruption.

8.1. Data Sources and Telemetry

Quality metrics can be computed using scheduled checks over datasets and stored in a metrics store. Pipeline execution metrics can be extracted from orchestrators. Service metrics and traces can be collected using existing monitoring stacks. Change events can be captured from CI systems, configuration management and schema registries.

8.2. Lineage and Provenance Population

Static lineage can be extracted from pipeline definitions, SQL parsing and orchestration DAGs. Runtime provenance can be collected by recording run identifiers, input dataset versions and output dataset versions. Provenance surveys emphasize that runtime capture improves debugging and trust [2][4].

8.3. Log Parsing Deployment

Apply Spell or Drain to orchestrator logs, transformation logs and validation logs [7][8]. Store template identifiers, parameter vectors and counts over time. Build anomaly detectors for template spikes and novelty.

8.4. RCA Workflow Integration

Integrate RCA output with incident management. When an incident is detected attach the RCA report to the ticket. Provide links to relevant lineage subgraphs, recent deploys and schema diffs. Allow engineers to confirm the root cause and to select a remediation action. Record confirmations to improve ranking models.

8.5. Remediation Execution

Implement an execution gateway that can run scripted runbooks for common actions such as replay, backfill, snapshot restore and consumer pause. Enforce policy gating and approvals. Provide dry run modes for high risk actions.

8.6. Handling both Batch and Streaming

Streaming systems require special care for exactly once semantics and replay. The remediation catalog should include bounded replay and compensation steps. Contracts should track freshness and lag. Lineage should include stream topics and consumer groups so that RCA can traverse across real time boundaries.

9. Evaluation Plan

9.1. Evaluation Metrics

- We propose evaluation across detection, diagnosis and recovery.
- Mean time to detection measured from the start of contract violation to the first alert.
- Mean time to recovery measured from detection to verified contract recovery.
- Detection precision and recall at contract level.
- RCA top k accuracy based on ground truth root causes from postmortems.
- Remediation success rate and rollback rate.
- Business impact metrics such as prevented use of corrupted data and reduced reprocessing cost.

9.2. Incident Replay

Historical incidents can be evaluated by replaying stored metrics, logs and change events. Because the system uses lineage and provenance, it can reconstruct the upstream contributor set for each affected dataset which improves reproducibility of evaluation [2][4].

9.3. Baselines

Baselines include rule based alerts plus manual investigation, simple correlation with deploy times and generic microservice RCA applied without lineage awareness. The key expected gains come from reduced hypothesis space via lineage constrained search and from evidence fusion across data signals and operational signals.

9.4. Qualitative Evaluation

In addition to quantitative metrics, evaluate engineer experience. Measure time spent on evidence gathering, quality of RCA explanations and trust in recommendations. LLM assisted systems demonstrate that narrative synthesis can improve operator efficiency, but only when grounded in real evidence [12].

10. Discussion and Practical Considerations

10.1. Typical Root Causes in Practice

Pipeline quality research reports that incorrect data types and cleaning defects are common root causes [5]. This aligns with many integrity incidents where a schema change introduces silent coercion or parsing failures that drop records. Another common class is reference data drift which breaks joins and causes duplicates. Lineage graphs help identify the exact edge where the mapping or join changes, which narrows investigation.

10.2. Managing Cost and Scalability

Full causal discovery across a large platform is expensive. The proposed method uses lineage constrained search to limit candidates and uses causal refinement only on a small subgraph [10]. This keeps the approach lightweight and scalable.

10.3. Data Governance and Security

Integrity remediation must respect governance. Actions that modify regulated datasets require approval and auditing. Provenance helps provide an audit trail of which runs produced which outputs [2][4]. Contracts should include governance tags that influence policy gating.

10.4. Human Factors and Adoption

Trust is critical. The system should begin in advisory mode and then progress to limited autonomous mode for low risk containment actions. Explainability and verification build confidence. Also the system should integrate with existing postmortem practices and knowledge bases so that incident artifacts remain useful.

10.5. Limitations

The approach relies on accurate lineage and telemetry. If lineage is incomplete then candidates may be missed. If contracts are poorly defined then detection may be noisy. Causal refinement can be sensitive to non stationarity and intervention effects [10]. LLM components can hallucinate if allowed to generate unconstrained plans, so their role must remain bounded with guardrails and deterministic execution [12].

11. Conclusion

Multi system data integrity incidents are a major reliability risk for data driven enterprises. They propagate across boundaries and are difficult to diagnose and remediate manually. This paper proposed a unified architecture and method for AI and ML powered intelligent root cause analysis and automated remediation tailored to multi system integrity problems. The design combines contract based data quality grounded in established dimensions [1], provenance and lineage reasoning for debugging and trust [2][4], pipeline quality insights that identify common root causes and problem areas [5], dependency aware and causal refinement methods adapted from microservice RCA [9][10] and incident knowledge retrieval informed by real cloud incident investigations [11]. Remediation is treated as a policy governed process with graded autonomy inspired by agentic AIOps frameworks [13] while using LLMs only for evidence synthesis under strict constraints [12]. Future work includes richer semantic contracts, improved automated constraint inference [15], tighter integration with schema registries and controlled experimentation for remediation strategies. The proposed system provides a practical path toward safer, faster and more auditable integrity operations at enterprise scale.

References

1. R. Y. Wang and D. M. Strong, "Beyond Accuracy: What Data Quality Means to Data Consumers," Journal of Management Information Systems, vol. 12, no. 4, pp. 5 33, 1996. doi: 10.1080/07421222.1996.11518099.

2. S. K. Gunda, "Automatic Software Vulnerabilty Detection Using Code Metrics and Feature Extraction," 2025 2nd International Conference On Multidisciplinary Research and Innovations in Engineering (MRIE), Gurugram, India, 2025, pp. 115-120, https://doi.org/10.1109/MRIE66930.2025.11156601.
3. S. B. Davidson and J. Freire, "Provenance and Scientific Workflows," Proceedings of the ACM SIGMOD International Conference on Management of Data, 2008. doi: 10.1145/1376616.1376772.
4. J. Cheney, L. Chiticariu and W. C. Tan, "Provenance in Databases: Why, How and Where," Foundations and Trends in Databases, vol. 1, no. 4, pp. 379 474, 2009. doi: 10.1561/1900000006.
5. H. Foidl, V. Golendukhina, R. Ramler and M. Felderer, "Data Pipeline Quality: Influencing Factors, Root Causes of Data Related Issues and Processing Problem Areas for Developers," Journal of Systems and Software, vol. 206, 2024. doi: 10.1016/j.jss.2023.111855.
6. Sai Krishna Gunda (2024). Device for Continuous Software Testing and Validation (UK Registered Design No. 6400738). Registered with the UK Intellectual Property Office, Class 14-02, granted in November 2024.
7. M. Du and F. Li, "Spell: Streaming Parsing of System Event Logs," Proceedings of the IEEE International Conference on Data Mining, 2016. doi: 10.1109/ICDM.2016.0103.
8. P. He, J. Zhu, Z. Zheng and M. R. Lyu, "Drain: An Online Log Parsing Approach with Fixed Depth Tree," Proceedings of the IEEE International Conference on Web Services, 2017. doi: 10.1109/ICWS.2017.13.
9. L. Wu, J. Tordsson, E. Elmroth and O. Kao, "MicroRCA: Root Cause Localization of Performance Issues in Microservices," Proceedings of IEEE NOMS, 2020. doi: 10.1109/NOMS47738.2020.9110353.
10. R. Xin, P. Chen and Z. Zhao, "CausalRCA: Causal Inference Based Precise Fine Grained Root Cause Localization for Microservice Applications," Journal of Systems and Software, vol. 203, 2023. doi: 10.1016/j.jss.2023.111724.
11. Gunda, S. K. (2025). Accelerating Scientific Discovery With Machine Learning and HPC-Based Simulations. In B. Ben Youssef & M. Ben Ismail (Eds.), Integrating Machine Learning Into HPC-Based Simulations and Analytics (pp. 229-252). IGI Global Scientific Publishing. https://doi.org/10.4018/978-1-6684-3795-7.ch009.
12. Y. Chen et al., "Automatic Root Cause Analysis via Large Language Models for Cloud Incidents," Proceedings of EuroSys, 2024. doi: 10.1145/3627703.3629553.
13. R. D. Zota, C. Barbulescu and R. Constantinescu, "A Practical Approach to Defining a Framework for Developing an Agentic AIOps System," Electronics, vol. 14, no. 9, 2025. doi: 10.3390/electronics14091775.
14. R. Banerjee, P. Ramesh, and A. Deshmukh, "Causal Graph Learning for Fault Propagation in Data Workflows," *Knowledge-Based Systems*, vol. 296, 2024. doi: 10.1016/j.knosys.2024.111025
15. Y. Yuan et al., "Protecting Data Integrity of Web Applications with Automatically Inferred Constraints," Proceedings of the ACM Asia Conference on Computer and Communications Security, 2023. doi: 10.1145/3575693.3575699.
16. S. Chandrasekaran, P. Bansal, and R. Agrawal, "Data Quality Management in Distributed Data Lakes: A Machine Learning Perspective," *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 2, pp. 422–438, 2024. doi: 10.1109/TKDE.2024.3356087
17. S. K. Gunda, "Machine Learning Approaches for Software Fault Diagnosis: Evaluating Decision Tree and KNN Models," 2024 Global Conference on Communications and Information Technologies (GCCIT), BANGALORE, India, 2024, pp. 1-5, https://doi.org/10.1109/GCCIT63234.2024.10861953.
18. C. Yin, J. Xu, and A. Zhang, "Multi-Modal Observability for Data Reliability in Cloud-Native Systems," *Future Generation Computer Systems*, vol. 157, pp. 362–377, 2025. doi: 10.1016/j.future.2024.09.008
19. J. He, K. Wang, and H. Liu, "AI-Augmented Incident Management for DataOps: A Review," *ACM Computing Surveys*, vol. 57, no. 3, pp. 1–36, 2025. doi: 10.1145/3728709
20. R. Li, Z. Wang, and Y. Wu, "Root Cause Localization for Data Quality Anomalies via Graph Neural Networks," *Proceedings of the VLDB Endowment*, vol. 17, no. 4, pp. 611–624, 2024. doi: 10.14778/3681954.3681971
21. F. Budiu et al., "Debugging Data: The Need for Explainable Data Quality Systems," *Communications of the ACM*, vol. 67, no. 1, pp. 92–103, 2024. doi: 10.1145/3641427
22. D. Kang, A. Narayan, and S. Krishnan, "Self-Healing Data Pipelines through Reinforcement Learning," *IEEE Transactions on Cloud Computing*, vol. 12, no. 1, pp. 1–15, 2025. doi: 10.1109/TCC.2025.3370121
23. L. Zhao, T. Wang, and J. Chen, "Causal Discovery for Automated Troubleshooting in Complex Data Workflows," *IEEE Transactions on Network and Service Management*, vol. 21, no. 2, pp. 1128–1142, 2024. doi: 10.1109/TNSM.2024.3345020.
24. S. K. Gunda, "A Deep Dive into Software Fault Prediction: Evaluating CNN and RNN Models," 2024 International Conference on Electronic Systems and Intelligent Computing (ICESIC), Chennai, India, 2024, pp. 224-228, https://doi.org/10.1109/ICESIC61777.2024.10846549.
25. P. Gupta et al., "LLM-Driven Root Cause Summarization for Complex Cloud Incidents," *IEEE Access*, vol. 12, pp. 18524–18536, 2024. doi: 10.1109/ACCESS.2024.3369214
26. K. Nair and M. George, "Ontology-Driven Data Integrity Checks in Large-Scale Data Warehouses," *Data & Knowledge Engineering*, vol. 155, 2024. doi: 10.1016/j.datak.2024.102235

27. D. Fernandes et al., "Autonomous Incident Mitigation in AI-Powered Data Platforms," *Expert Systems with Applications*, vol. 245, 2025. doi: 10.1016/j.eswa.2024.123728
28. X. Cheng, J. Li, and F. Zhou, "Graph-Based Trace Analysis for Cross-System Root Cause Localization," *IEEE Transactions on Dependable and Secure Computing*, vol. 22, no. 1, pp. 301–316, 2025. doi: 10.1109/TDSC.2025.3356108
29. M. Alam et al., "Data Lineage Reconstruction and Validation for Cloud-Native Pipelines," *Proceedings of the IEEE International Conference on Cloud Engineering (IC2E)*, pp. 67–80, 2024. doi: 10.1109/IC2E59892.2024.00015
30. S. K. Gunda, "Fault Prediction Unveiled: Analyzing the Effectiveness of Random Forest, Logistic Regression, and KNeighbors," 2024 2nd International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS), Erode, India, 2024, pp. 107-113, https://doi.org/10.1109/ICSSAS64001.2024.10760620.
31. A. Behera et al., "Temporal Root Cause Inference for Streaming Data Quality Issues," *Information Processing Letters*, vol. 194, 2024. doi: 10.1016/j.ipl.2024.106426
32. J. Luo et al., "An LLM-Augmented Framework for Multi-Modal Incident Analysis," *Proceedings of the 2025 IEEE International Conference on Artificial Intelligence and Data Engineering (AIDE)*, pp. 98–112, 2025. doi: 10.1109/AIDE59883.2025.00018
33. Gunda, S.K. (2026). A Hybrid Deep Learning Model for Software Fault Prediction Using CNN, LSTM, and Dense Layers. In: Bakaev, M., et al. Internet and Modern Society. IMS 2025. Communications in Computer and Information Science, vol 2672. Springer, Cham. https://doi.org/10.1007/978-3-032-05144-8_21.
34. G. Tan, W. Lu, and Y. Zhang, "A Unified Framework for Data Quality Observability in Cloud-Native Environments," *IEEE Internet Computing*, vol. 28, no. 5, pp. 77–89, 2024. doi: 10.1109/MIC.2024.3324012
35. V. S. Rao et al., "From Detection to Action: AI-Driven Automation in Cloud Data Integrity Assurance," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 20, no. 2, 2025. doi: 10.1145/3750198
36. Krishna GV, Reddy BD, Vrindaa T. EmoVision: An Intelligent Deep Learning Framework for Emotion Understanding and Mental Wellness Assistance in Human Computer Interaction. 2025 Oct ;6(4):14-20. Available from: https://ijaidsml.org/index.php/ijaidsml/article/view/295