International Journal of Emerging Trends in Computer Science and Information Technology
ISSN: 3050-9246 | https://doi.org/10.63282/3050-9246.IJETCSIT-V6I3P116
Eureka Vision Publication | Volume 6, Issue 3, 104-111, 2025

Original Article

Leveraging AI and ML for Predictive Monitoring and Error Mitigation in Change Data Capture Pipelines

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

Received On: 24/05/2025 ***Revised On: 16/07/2025*** ***Accepted On: 28/07/2025*** ***Published On: 21/08/2025***

Abstract - Change Data Capture pipelines are widely used to propagate database changes into event streams, analytical stores and operational read models with low latency. As enterprises expand the number of source databases, connectors, downstream consumers and serving systems, operational reliability becomes harder to sustain. Failures that begin as a minor lag increase or a subtle schema evolution can cascade into missed records, duplicated events, inconsistent materialized views and downstream business defects. Traditional monitoring based on static thresholds and manual triage struggles because CDC behavior is non stationary, highly correlated across components and sensitive to workload shifts and change management practices. This paper proposes an architecture that embeds adaptive intelligence into CDC operations through predictive monitoring, anomaly detection, diagnosis and guarded automation. The approach fuses three families of signals. The first family is pipeline telemetry such as connector lag, throughput, offsets, retries and backpressure. The second family is data integrity signals such as row count deltas, key uniqueness checks and reconciliation between source and sink. The third family is change signals such as deploys, connector configuration edits and schema registry events. Lightweight models learn baselines and predict near term risk for lag growth, event loss and replication divergence. A graph based diagnosis method constrains root cause search using CDC topology and lineage and then ranks hypotheses using multi modal evidence including structured log templates. Finally, an action layer executes risk tiered mitigation steps such as auto scaling consumers, pausing downstream writes, triggering bounded replays and initiating snapshot repair with human approval gates for high impact actions. The paper outlines a prototype design and an evaluation plan using historical incident replay. It argues that the combination of predictive signals, topology aware diagnosis and policy based automation can reduce mean time to detection and mean time to recovery while improving trust in CDC driven data products.

Keywords - Change Data Capture, Data Pipelines, Predictive Monitoring, Anomaly Detection, Data Integrity, Observability, Automated Remediation, Root Cause Analysis, Concept Drift.

1. Introduction

Enterprises increasingly treat operational data as a product that must be delivered continuously to multiple consumers. Online applications need current views for search and personalization, fraud systems need fast features and analytics teams need timely facts for reporting. Change Data Capture, often implemented through log based connectors and event streaming, is a common backbone for these needs. In a typical CDC deployment, a connector reads transaction logs from a source database, converts changes into a standardized event format, publishes events to a streaming platform and downstream processors apply transformations and updates to sinks such as data warehouses, key value stores or search indexes. CDC enables low latency propagation without intrusive triggers and reduces the cost of full reload ETL patterns. Yet the very characteristics that make CDC attractive also make it operationally fragile. The pipeline spans multiple domains, evolves over time and must preserve correctness under failures and reconfiguration.

Operational issues in CDC pipelines are not limited to uptime. They include silent integrity degradation where the pipeline continues to run but produces incorrect data. Examples include missing updates due to connector offset mismanagement, duplicated events after retries, out of order application of changes, inconsistent snapshots, schema drift that causes parse failures and propagation of tombstones that unexpectedly delete records in a sink. These defects can persist for hours before detection because symptoms appear in downstream aggregates rather than in connector error logs. Data pipeline quality research highlights that data related issues often arise from incorrect types, integration defects and processing problem areas that are hard for developers to manage at scale [5].

Standard monitoring practices focus on dashboards, rule based alerts and manual runbooks. Such practices are necessary but insufficient for modern CDC where normal behavior shifts with workload, schema evolution and scaling. Data quality and monitoring literature emphasizes that quality is multi dimensional, involving completeness, consistency and timeliness as key concepts [1]. CDC reliability demands that these dimensions be monitored end to end, not only at a single component. Observability frameworks for data quality emphasize the need to combine quality signals, freshness signals and lineage signals in order to prevent issues before they accumulate [6].

At the same time, the broader operations community has advanced automated anomaly detection, root cause analysis and incident knowledge mining. Structured log anomaly detection pipelines can be built as modular systems that accelerate creation and management of monitoring items [8]. Root cause knowledge can be mined from incident investigations and used to recommend diagnosis steps and remediation actions [7]. Microservice RCA research shows that dependency aware reasoning over service graphs can localize root causes more precisely than isolated metric correlation [11][12]. However, CDC pipelines combine database logs, stream processing and data integrity constraints which require specialized modeling.

This paper contributes an architecture for predictive monitoring and error mitigation tailored to CDC. The key design principle is to treat CDC as a socio technical system where telemetry, data integrity and change management signals must be fused. Rather than relying on a single large model, the architecture uses lightweight predictors for specific operational risks and a graph based diagnosis method that narrows the hypothesis space using CDC topology and lineage. Automated actions are executed under policy based gates to preserve safety and auditability.

The rest of the paper is organized as follows. Section II reviews related work in CDC, data quality, observability and operational intelligence. Section III describes CDC pipelines and an incident taxonomy. Section IV presents the proposed architecture. Section V details predictive models and anomaly detection. Section VI describes topology constrained diagnosis and RCA. Section VII presents mitigation and automation with governance. Section VIII outlines a prototype implementation. Section IX proposes an evaluation plan. Section X discusses limitations and future directions. Section XI concludes.

2. Background and Related Work

2.1. Change Data Capture systems

CDC is the practice of capturing changes from a source system and propagating them downstream. A common pattern uses transaction logs to avoid write amplification and to preserve ordering. Event based data integration work emphasizes that logs and event streams provide a durable backbone for distributing changes across distributed systems [2]. In data warehousing contexts, workload aware CDC strategies have been proposed to balance overhead and freshness by selecting among trigger based, timestamp based and log based approaches depending on workload characteristics [3]. Practical CDC deployments also require an initial snapshot in order to establish full state and then incremental changes from logs. A watermark based framework such as DBLog interleaves log events with chunked snapshot reads to capture full state while maintaining progress of log consumption, addressing production needs for repair and resynchronization [4].

2.2. Data quality and data pipeline reliability

Data quality is best understood as fitness for use with multiple dimensions such as completeness, consistency and timeliness [1]. In CDC pipelines these dimensions map to tangible properties: completeness relates to missing changes or missing partitions in derived stores, consistency relates to duplicates and referential mismatches between source and sink and timeliness relates to replication lag and freshness windows. A recent study on data pipeline quality provides a taxonomy of influencing factors and identifies common root causes of data related issues such as incorrect types and cleaning problems [5]. These findings align with CDC incidents where schema evolution or conversion logic introduces silent coercion and downstream corruption.

2.3. Data observability and responsible monitoring

Observability for data systems aims to provide sufficient signals to explain pipeline behavior and to detect integrity issues early. Frameworks such as TENSAI propose practical observability for data quality aware analytics by combining measurement, responsible monitoring and operational integration [6]. This work supports the idea that data quality monitoring should be systematic, not ad hoc, and that observability must include provenance and context.

2.4. Anomaly detection, log parsing and operational intelligence

In large systems, anomaly detection is used to identify deviations in metrics and logs. ADOps demonstrates a modular pipeline for anomaly detection in structured logs and reports a PVLDB implementation with configuration driven task creation [8]. Many log based approaches rely on parsing unstructured logs into templates. Spell proposes streaming parsing of system event logs [9] and Drain proposes an online log parsing approach using a fixed depth tree [10]. These methods enable template novelty detection and burst detection which are valuable in CDC because many failures manifest as new error signatures or spikes of known signatures.

2.5. Root cause analysis and incident knowledge mining

Topology aware RCA has been widely studied for microservices. MicroRCA localizes performance issues by using service interaction graphs and metric correlations [11]. CausalRCA applies causal inference to localize root causes at fine granularity in microservice applications [12]. While CDC pipelines differ from microservices, they share a dependency graph structure where downstream symptoms can be traced to upstream components. Separately, incident knowledge mining approaches extract patterns from postmortems and incident investigations to accelerate RCA workflows [7]. These ideas motivate storing CDC incident artifacts and using retrieval to recommend likely causes and mitigation steps.

2.6. Concept drift and non stationary monitoring

Monitoring models degrade when normal behavior changes. A survey on unsupervised drift detection emphasizes that drift can occur without labels and that monitoring systems need clear definitions and robust methods to detect shifts [13]. In streaming settings, drift and anomalies can interact. Graph stream research has studied drift and anomaly detection when relationships evolve and

shows that graph based representations can capture evolving structure [14]. CDC pipelines exhibit both metric drift due to workload seasonality and structural drift due to topology and schema changes, so drift awareness is crucial.

3. CDC Pipeline Model and Failure Taxonomy

3.1. Reference CDC pipeline

We model a CDC pipeline as a set of stages connected by durable queues. A source database emits a transaction log. A connector reads the log and converts each change into an event record. The event is published to a streaming platform. Downstream processors consume events and apply transformations such as enrichment, filtering and aggregation. Finally, sink connectors or application services apply changes to target systems. In many deployments, a schema registry governs event schemas and provides compatibility rules. A control plane manages connector configuration, scaling and restart behavior.

The pipeline must maintain three invariants. First, it must preserve at least once or exactly once delivery semantics depending on the design. Second, it must preserve ordering constraints required by sinks, often per key. Third, it must preserve data integrity by ensuring that downstream state converges to source state over time. Achieving these invariants under failures requires careful offset management, idempotent sinks and bounded reprocessing strategies.

3.2. Failure categories

We propose a taxonomy of CDC failures aligned to data quality dimensions [1] and to pipeline quality root causes [5]. The taxonomy helps map symptoms to likely causes and guides model selection and mitigation action.

- Timeliness failures these are characterized by increasing replication lag, consumer backlog and missed freshness objectives. Causes include insufficient resources, downstream sink throttling, network partitions and skewed partitions. Timeliness failures are often precursors to correctness failures because large lag increases the window where logs can expire and forces snapshot resynchronization.
- Completeness failures these include missing events, dropped records and gaps in offsets. Causes include connector crashes with offset loss, log retention limits, misconfigured filters and parsing failures caused by schema changes. Completeness failures can be hard to detect because row counts can remain stable while specific segments are missing.
- Consistency failures these include duplicates, out of order applies and mismatch between source and sink. Causes include non idempotent sink writes, retries that reapply changes, inconsistent deduplication logic and race conditions across parallel consumers. Consistency failures often manifest as reconciliation mismatches or uniqueness violations in derived stores.
- Schema and semantic drift failures These include incompatible schema evolution, field type changes, new enumerations and changes in meaning such as units or encoding. A pipeline quality study identifies incorrect types as a frequent root cause of data issues [5]. In CDC, drift can cause silent coercion, null spikes or ingestion halts if compatibility rules reject events.
- Control plane and change management failuresThese include misconfiguration, wrong connector parameters, invalid secrets and unsafe deploys. Control plane events often correlate with incidents and are important features for diagnosis and prediction. Event driven systems research emphasizes that distributed change propagation must be treated systematically to avoid dual writes and inconsistency [2].

3.3. Observability signals for CDC integrity

We categorize observability signals into telemetry, integrity and change signals. Telemetry signals include source log position, connector offsets, publish latency, consumer lag, throughput and retry counts. Integrity signals include source to sink reconciliation, row count deltas by partition, key uniqueness checks and checksum comparisons for sampled keys. Change signals include connector configuration edits, schema registry updates, deploy events and scaling events. Data observability frameworks suggest combining quality, freshness and lineage signals to improve trust [6]. Our architecture uses this combination as the basis for learning and action.

4. Proposed Architecture for Predictive Monitoring and Mitigation

The architecture consists of seven layers. Each layer is designed to be modular and to integrate with existing enterprise tooling. The central idea is to build a closed loop system where detection and prediction lead to diagnosis, mitigation and learning.

4.1. Layer 1 Data and control plane instrumentation

This layer collects metrics, logs and traces from connectors, streaming platforms, processors and sinks. It also collects change events from CI systems, configuration stores and schema registries. Logs are parsed into templates using online log parsers [9][10]. Traces are used where available to link connector operations to downstream processing and sink writes. This layer also extracts CDC specific integrity signals such as per table lag, per partition offsets and snapshot progress indicators.

4.2. Layer 2 Contract and integrity specification

Integrity expectations are defined as contracts with thresholds and statistical bounds. Contracts encode objectives for freshness, expected volume ranges, schema compatibility rules and reconciliation requirements. Data quality dimensions provide the conceptual basis for contract design [1]. Contracts are scoped by dataset, table, topic and sink and can include criticality tags that drive policy and severity.

4.3. Layer 3 Baseline learning and predictive risk scoring

This layer trains lightweight models that learn normal behavior and forecast near term risk. Predictors estimate

future lag growth, probability of log retention breach and likelihood of reconciliation divergence. Drift detection monitors whether baseline assumptions are changing [13]. Models are trained per entity where possible and also globally with grouping by similar patterns. Outputs are risk scores that feed alerting and mitigation planning.

4.4. Layer 4 Anomaly detection and incident formation

This layer detects anomalies in telemetry, integrity metrics and log template streams. It uses robust statistical detectors for simple cases and incremental methods for streaming metrics. For structured logs it can use pipeline style approaches such as ADOps which separate data preparation, model selection and deployment [8]. Each detection generates an incident candidate with a symptom signature including affected entity, dimension, time window and deviation vector.

4.5. Layer 5 Topology aware diagnosis and RCA

Diagnosis operates over a CDC topology graph that includes sources, connectors, topics, processors, sinks and control plane artifacts. The engine performs constrained upstream search from the symptom node to generate candidates and then ranks hypotheses using evidence features. The method draws inspiration from dependency aware microservice RCA [11] and can apply causal refinement for high ambiguity cases following causal inference ideas [12]. Incident knowledge retrieval from prior postmortems augments ranking and provides recommended checks and actions [7].

4.6. Layer 6 Mitigation planner and guarded automation

The planner selects actions from a catalog. Actions are categorized by risk. Low risk actions include scaling consumers, restarting a stuck connector and pausing downstream writes. Higher risk actions include bounded replay, snapshot repair and schema registry rollback. A policy engine enforces approval requirements based on risk and confidence. This governance is essential because corrective actions can modify data state.

4.7. Layer 7 Verification and learning store

After action execution the system verifies recovery by rerunning contracts and reconciliation. It records the incident artifact, root cause label, actions and outcome. These artifacts support continuous improvement of ranking and prediction and align with incident knowledge mining approaches [7].

5. Predictive Monitoring and Anomaly Detection Methods

5.1. Predicting replication lag and retention risk

Replication lag is a primary leading indicator for CDC health. If lag grows beyond log retention windows the pipeline can lose the ability to read changes and must rebuild via snapshot. We propose a forecasting model for lag per connector and per table where supported. Features include current lag, throughput, source commit rate, sink apply rate, retry rates, partition skew and recent change events. A simple approach uses exponential smoothing with change point detection for abrupt shifts. For more complex patterns a lightweight autoregressive model or gradient boosted tree can be used. The key is to produce a near term risk estimate, such as the probability that lag will exceed a threshold within a time horizon.

Change point and anomaly detection methods for correlated time series provide motivation for combining drift and anomaly signals in high dimensional monitoring [15]. Drift detection is also applied to the residuals of the forecasting model to identify when the model should be retrained.

5.2. Predicting integrity divergence

Lag alone does not capture correctness. We propose an integrity divergence predictor that estimates risk of source to sink mismatch. The predictor uses features from reconciliation checks, such as delta counts per key range, checksum differences for sampled keys and uniqueness violations. It also uses features from schema evolution events. Because labels are rare, the model can be trained using weak supervision where known incidents provide positive labels and quiet windows provide negative labels.

5.3. Anomaly detection across metric families

The anomaly detection layer operates across several metric families Connector health metrics include task failures, restart loops, connection errors and offset commit failures.Streaming metrics include topic produce error rates, consumer lag and broker throttling. Sink metrics include write latency, error rates and rate limit responses. Integrity metrics include reconciliation deltas and unexpected null spikes. Log template metrics include new templates and burst rate of known templates.

For metrics with stable seasonal patterns, use robust z scores and seasonal baselines. For non seasonal streaming metrics use incremental detectors and isolation based methods. For structured log streams use template novelty detection and frequency anomaly detection based on parsed logs [9][10]. ADOps provides an example of modular deployment that can manage many anomaly tasks and support both simple rule based checks and model based detection [8].

5.4. Drift detection and model management

CDC pipelines evolve due to new tables, new connectors and schema evolution. Drift detection should monitor not only metrics but also the joint distribution of key features. The unsupervised drift survey emphasizes the need to distinguish drift from anomalies and to define change types clearly [13]. In our architecture drift detection triggers model retraining and also increases uncertainty in predictive risk scores, which in turn tightens automation gating.

Graph based drift is relevant when the topology itself changes, for example when a new consumer group is added or a sink is migrated. Graph stream research on drift and anomaly detection motivates representing topology changes explicitly rather than treating them as noise [14].

5.5. Practical alert design

Predictive monitoring supports earlier alerts but can also increase noise if not tuned. We recommend a two stage alert design. Stage one is a predictive early warning when risk exceeds a threshold but contracts are still within bounds. Stage two is a contract violation alert when integrity checks fail. This separation helps operators prioritize and provides time to apply low risk mitigations before correctness degrades.

6. Topology Constrained Diagnosis and Root Cause Analysis

6.1. CDC topology graph

We represent the CDC system as a directed graph where nodes include source databases, log readers, connector tasks, topics, processor jobs, sink connectors, schema versions and control plane changes. Edges represent produces, consumes, transforms and configures relations. This graph is derived from configuration and runtime metadata and can be augmented with lineage information. The topology graph is the primary structure that constrains diagnosis.

6.2. Symptom signatures

An incident is represented by a symptom signature S that includes the affected node, the violated dimension, the time window and deviation measures. For example a signature can represent lag growth on a connector combined with a rising reconciliation delta for a sink table.

6.3. Candidate generation via constrained traversal

Given a symptom node, generate a candidate set by traversing upstream edges within a bounded depth and including co temporal change nodes that touched any node along the path. This reduces the hypothesis space and aligns with dependency aware approaches in microservice RCA [11].

6.4. Evidence scoring and ranking

For each candidate compute evidence features. Temporal alignment between candidate events and symptom onset Anomaly evidence from metrics and log templates Schema drift evidence from registry diffs Backpressure evidence from queue metrics and sink latency Reconciliation proximity evidence based on which entities directly influence the violated dataset Historical similarity evidence from retrieval of past incident artifacts as suggested by incident knowledge mining work [7] Candidates are ranked by a weighted scoring function or a lightweight learning to rank model trained on labeled incidents. The output includes an explanation that links evidence back to observables and contracts. Explainability is critical for trust in automation.

6.5. Causal refinement for ambiguous incidents

When the top candidates have similar scores, apply causal refinement on a small subgraph containing the candidates and their neighbors. CausalRCA shows that causal inference can improve precision for microservice localization [12]. In CDC, causal refinement is applied to distinguish whether lag is caused by source slowdown, connector bottleneck or sink throttling. Restricting causal learning to a small subgraph helps manage complexity and reduces sensitivity.

6.6. Integrating postmortem knowledge

Incident investigations often reveal recurring patterns such as schema evolution causing parse errors or connector configuration changes causing offset resets. Mining root cause knowledge from incident investigations can support retrieval and recommendation systems [7]. The learning store in our architecture captures CDC incidents and enables similarity search by symptom signature and evidence patterns. When a new incident appears, the system retrieves similar cases and suggests checks and mitigation steps that worked previously, improving operator efficiency.

7. Error Mitigation and Automated Remediation

7.1. Design goals for mitigation

Mitigation actions must be safe because they can affect data correctness. We define four goals. Contain impact so that corrupted data does not propagate to consumers Restore correctness by repairing missing or duplicated changes Restore timeliness by reducing lag and clearing backlogs Preserve auditability and enable rollback where possible

7.2. Action catalog with risk tiers

Tier 0 Informational actions create a ticket, annotate dashboards, notify owners and attach diagnosis evidence. Tier 1 Containment actions Pause sink writes, route reads to a last known good snapshot, quarantine affected partitions, disable a problematic table capture temporarily. Tier 2 Recovery actions Scale connector tasks and consumers, restart a stuck task with controlled offset handling, trigger bounded replay of a time window, rebuild a materialized view, run a targeted reconciliation epair for a key range. Tier 3 Structural actions Rollback a connector or processor deployment, revert a schema registry change, trigger a snapshot resynchronization using a watermark based method, perform a controlled cutover to a repaired sink.

A watermark based CDC framework supports safe snapshot repair because it allows interleaving snapshot reads with log events and can be triggered for specific tables or keys [4]. Workload aware CDC research also suggests that strategy choice influences overhead and correctness, which can inform mitigation design [3].

7.3. Guarded automation via policy

Automation is gated by policy based rules that consider action risk, confidence in diagnosis and business criticality tags from contracts. For example, auto scaling a consumer group can be executed automatically when confidence is high and rollback is straightforward. By contrast, forcing a snapshot resynchronization or rewinding offsets can cause duplicates and requires human approval. Policy gates also enforce rate limits on actions to avoid oscillations.

7.4. Verification and reconciliation after action

After any action the system verifies recovery. Verification includes checking lag, rerunning reconciliation

and validating that duplicates and null rates return to expected ranges. If verification fails, the system escalates. This verification loop is also used to label outcomes for learning.

7.5. Human in the loop workflows

Operators remain essential for complex correctness incidents. The system is designed to assist, not replace, engineers by reducing evidence collection time, ranking hypotheses and proposing safe actions. Incident knowledge retrieval supports this by surfacing prior cases and recommended steps [7].

8. Prototype Implementation Guidance

8.1. Data collection and storage

Implement a unified telemetry store that ingests metrics, log templates and change events. Store per connector metrics at fine granularity and integrity checks at a cadence aligned to business requirements. Use a time series database for metrics and a document store for incident artifacts.

8.2. Log parsing and signature features

Deploy online log parsers such as Spell or Drain to convert logs into templates [9][10]. Maintain template frequency time series and detect new templates and bursts. For CDC this is useful for recognizing new failure modes after deploys or schema changes.

8.3. Topology graph construction

Construct the CDC topology graph from connector configurations, streaming metadata, processor DAGs and schema registry links. Include control plane changes as nodes with edges to affected components. This allows diagnosis traversal and also supports impact analysis before change management actions..

8.4. Model training and deployment

Start with interpretable models. Use simple forecasting for lag and robust detectors for integrity metrics. Add learning to rank for diagnosis after collecting labeled incidents. Use drift detection to schedule retraining. Prefer lightweight models because CDC incidents are sparse and high precision is more valuable than marginal recall gains.

8.5. Integration with incident management

Integrate with existing ticketing and on call processes. When an incident is created attach the symptom signature, diagnosis report and recommended actions. Allow operators to confirm root cause and action. Store confirmations for learning.

8.6. Safety and audit

Record every automated action with parameters and approvals. Ensure that replays and offset rewinds are bounded and idempotent where possible. Provide a dry run mode that estimates impact and displays which partitions or key ranges will be affected.

9. Evaluation Plan

9.1. Metrics

We propose evaluating the system across detection, diagnosis and recovery. Mean time to detection from the start of an integrity violation to alert Mean time to recovery from detection to verified contract recovery Precision and recall for contract violation detection Top k accuracy for diagnosis compared to postmortem ground truth Mitigation success rate and rollback rate Business impact measures such as prevented consumption of corrupted data and reduced reprocessing cost.

9.2. Historical incident replay

Replay historical telemetry and integrity checks to evaluate prediction and detection. The incident knowledge mining literature demonstrates that incident artifacts can be used for downstream tasks including retrieval and recommendation [7]. Use stored postmortems to label root causes and compare diagnosis ranking quality.

9.3. Baselines

Compare against threshold based monitoring, simple correlation with deploy times and generic microservice RCA methods without CDC topology constraints. We expect topology constrained search to reduce false leads and predictive monitoring to reduce detection delays.

9.4. Stress testing with synthetic faults

Inject controlled faults in a staging environment. Examples include artificial sink throttling, schema changes that introduce type mismatches, connector restarts with offset perturbations and log retention reductions. Evaluate whether the system predicts risk, detects anomalies, diagnoses correctly and proposes safe actions.

9.5. Operator evaluation

Conduct qualitative evaluation with on call engineers. Measure time saved in evidence gathering and perceived trust in recommendations. The goal is to create a system that improves reliability without creating alert fatigue.

10. Discussion, Limitations and Future Work

10.1. Handling exactly once semantics

Some CDC deployments require exactly once semantics. Achieving this end to end depends on connector guarantees, streaming platform configuration and idempotent sink behavior. The proposed architecture does not change the underlying semantics but helps detect when guarantees are violated and suggests mitigation that preserves idempotence.

10.2. Dependence on integrity signals

The architecture is strongest when integrity checks exist. If reconciliation is absent, the system relies more on indirect signals and uncertainty grows. Data quality research emphasizes that dimensions should be operationalized into measurable checks [1]. Implementing a minimal set of checks for key datasets is therefore a prerequisite.

10.3. Drift and evolving topology

Drift can reflect benign change, such as a new product launch, or harmful change, such as a misconfigured connector. Drift detection methods and clear definitions are needed to avoid spurious retraining and unnecessary alerts [13]. Topology evolution should be modeled explicitly and can itself be a predictor of risk when many changes occur in a short window.

10.4. Safety of automation

Automation for data correctness incidents carries higher risk than automation for availability incidents. Policy gating and verification loops are essential. Future work can explore more formal safety constraints for actions and simulation based impact estimation.

10.5. Future directions

Future work can incorporate richer causal inference and counterfactual reasoning to estimate which action will reduce risk with minimal side effects. It can also integrate organizational signals such as change management approvals and deployment pipelines. Finally, it can extend incident knowledge retrieval with structured templates and standardized taxonomies to improve cross team learning.

11. Conclusion

CDC pipelines are foundational for delivering timely data across enterprise systems, yet they are increasingly complex and prone to subtle integrity failures. This paper presented an architecture that leverages AI and ML for predictive monitoring and error mitigation in CDC. The approach combines contract based integrity monitoring grounded in established data quality dimensions [1], data observability principles [6], predictive models for lag and divergence risk, topology constrained diagnosis inspired by dependency aware RCA [11] and incident knowledge mining for operational learning [7]. A guarded automation layer executes risk tiered mitigation actions and verifies recovery to maintain trust.

By embedding adaptive intelligence into CDC operations, organizations can move from reactive troubleshooting to proactive risk management. The expected outcome is faster detection, faster recovery and lower business impact from integrity incidents. As enterprises continue to expand event driven architectures and rely on CDC for critical products, such predictive and policy governed operational intelligence will be an essential reliability capability.

References

[1] S. K. Gunda, "Analyzing Machine Learning Techniques for Software Defect Prediction: A Comprehensive Performance Comparison," 2024 Asian Conference on Intelligent Technologies (ACOIT), KOLAR, India, 2024, pp. 1-5, https://doi.org/10.1109/ACOIT62457.2024.10939610.

[2] M. Kleppmann, "Thinking in events: from databases to distributed collaboration software," Proceedings of the ACM Symposium on Principles of Distributed Computing, 2021. doi: 10.1145/3465480.3467835.

[3] W. Qu, J. Huang, J. Zhang and H. Chen, "A Workload Aware Change Data Capture Framework for On Demand Data Warehousing," in Advances in Databases and Information Systems, 2021. doi: 10.1007/978-3-030-86534-4_21.

[4] A. Andreakis and I. Papapanagiotou, "DBLog: A Watermark Based Change Data Capture Framework," arXiv, 2020. doi: 10.48550/arXiv.2010.12597.

[5] S. K. Gunda, "Software Defect Prediction Using Advanced Ensemble Techniques: A Focus on Boosting and Voting Method," 2024 International Conference on Electronic Systems and Intelligent Computing (ICESIC), Chennai, India, 2024, pp. 157-161, https://doi.org/10.1109/ICESIC61777.2024.10846550.

[6] H. L. Truong et al., "TENSAI: Practical and Responsible Observability for Data Quality Aware Large Scale Analytics," Journal of Data and Information Quality, 2024. doi: 10.1145/3708014.

[7] A. Saha et al., "Mining Root Cause Knowledge from Cloud Service Incident Investigations for AIOps," Proceedings of the IEEE ACM International Conference on Automated Software Engineering, 2022. doi: 10.1145/3510457.3513030.

[8] X. Song, Y. Zhu, J. Wu, B. Liu and H. Wei, "ADOps: An Anomaly Detection Pipeline in Structured Logs," Proceedings of the VLDB Endowment, vol. 16, no. 12, pp. 4050 to 4053, 2023. doi: 10.14778/3611540.3611618.

[9] S. K. Gunda, "Enhancing Software Fault Prediction with Machine Learning: A Comparative Study on the PC1 Dataset," 2024 Global Conference on Communications and Information Technologies (GCCIT), BANGALORE, India, 2024, pp. 1-4, https://doi.org/10.1109/GCCIT63234.2024.10862351.

[10] 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.

[11] 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.

[12] 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.

[13] S. K. Gunda, "Comparative Analysis of Machine Learning Models for Software Defect Prediction," 2024 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS), Chennai, India, 2024, pp. 1-6, https://doi.org/10.1109/ICPECTS62210.2024.10780167.

[14] D. Zambon, L. Alippi and L. Livi, "Concept Drift and Anomaly Detection in Graph Streams," IEEE Transactions on Neural Networks and Learning Systems, vol. 29, no. 11, pp. 5592 to 5605, 2018. doi: 10.1109/TNNLS.2018.2804443.

[15] M. Tveten, P. Fryzlewicz and S. K. Wied, "Scalable change point and anomaly detection in cross correlated data," Annals of Applied Statistics, vol. 16, no. 2, 2022. doi: 10.1214/21-AOAS1508.

[16] D. Seenivasan and M. Vaithianathan, "Real Time Adaptation: Change Data Capture in Modern Computer Architecture," International Journal of Advanced Computer Technology, 2023. doi: 10.56472/25838628/IJACT-V1I2P106.

[17] Xie, Y., Zhang, H., & Babar, M. A. (2022). LogGD: Detecting anomalies from system logs by graph neural networks. arXiv. https://doi.org/10.48550/arXiv.2209.07869

[18] S. Ghosh, S. Biswas and S. B. Roy, "Online anomaly detection with concept drift adaptation," Proceedings of the ACM International Conference on Information and Knowledge Management, 2017. doi: 10.1145/3152494.3152501.

[19] Sai Krishna Gunda (2024). Smart Device for Object-Oriented Software Prototype (UK Registered Design No. 6400739). Registered with the UK Intellectual Property Office, Class 14-02, granted in November 2024.

[20] 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.

[21] F. Hinder, M. Schmidt, C. Wirth, N. L. Dürr and U. Brefeld, "One or two things we know about concept drift: a survey on unsupervised drift detection," Frontiers in Artificial Intelligence, 2024. doi: 10.3389/frai.2024.1330257.

[22] Abu Alhija, H., Azzeh, M., & Almasalha, F. (2022). Software defect prediction using support vector machine. arXiv. https://doi.org/10.48550/arXiv.2209.14299