International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Self-Healing Data Pipelines: AI-Driven Automation in Financial Data Operations

Karthikeyan Sundar Raj

Abstract: Net asset value calculation, treasury reporting, and risk aggregation all run on data pipelines that are expected to work every day, on time, without exception. In practice they don't always. Upstream feeds arrive late. Source schemas change without warning. Null rates spike. Replication channels back up under load. Static-threshold monitoring catches the obvious cases and misses the rest, and it treats a market-wide price swing the same way it treats a corrupted feedvas an alert to be triaged manually. This paper proposes a self-healing pipeline architecture built around four separated, auditable stages: telemetry collection, machine learning-driven anomaly detection, dependency-aware diagnosis, and governed remediation. The detection logic draws directly from production price-validation practicevchecking a security's movement against correlated peers rather than judging it alone because that distinction is what separates a real anomaly from ordinary market behavior. Evaluation combines synthetic fault injection with replay of historical incidents, and reports verified production evidence: dependency-aware filtering of NAV pricing exceptions reduced false-positive review volume by roughly 95%, and eliminating redundant source-side updates cut change-data-capture replication lag from hours to seconds. Detection and recovery speed are characterized qualitatively rather than with an invented figure, since no verified time-to-detect or time-to-recover statistic exists for the reference production system. The paper closes with a look at generative-AI-assisted remediation reporting and regulatory-aware automation as next steps.

Keywords—*self-healing pipelines, anomaly detection, financial data architecture, root cause analysis, ETL reliability, AIOps*

I. Introduction

A fund accounting platform lives or dies on its feeds. Pricing data, corporate action notices, position updates, all of it has to arrive correctly and on schedule to support daily net asset value calculation. Treasury investment systems face the same pressure from a different angle, consolidating trade activity across available-for-sale and held-to-maturity portfolios for mark-to-market reporting. Neither system has room for a quiet failure. A late price update or a malformed corporate action record doesn't just delay a report; left unchecked, it becomes an inaccurate valuation that reaches a client or a regulator.

Four failure patterns show up again and again in these environments, summarized in Table 1. Feeds arrive late or incomplete, often around holidays or when a vendor changes its own systems without much notice. Source schemas drift; a field gets renamed, a format changes, and downstream transformation logic that assumed a stable contract quietly breaks. Data quality regresses in ways that pass basic validation but corrupt an aggregate calculation: an unexplained spike in nulls, a duplicate record, or a price that's technically valid but wrong. And infrastructure bottlenecks, replication contention, and resource exhaustion at peak load introduce latency that gets worse exactly when it can least be tolerated. None of this is speculative; each pattern has shown up directly in production fund accounting and treasury systems, not just in the generic DevOps literature that self-healing research usually draws on [1].

Independent Researcher, USA

Table 1: Failure Taxonomy in Financial Data Pipelines

Failure Mode	Typical Trigger	Detection Signal	Operational Consequence
Upstream feed delay	Vendor system change; market holiday scheduling	Missing or late file arrival; stale timestamp	Delayed NAV cutoff; late client reporting
Schema drift	Uncoordinated upstream field or format change	Parsing failure; unexpected null pattern	Silent transformation error; downstream corruption
Data-quality regression	Source system defect; corrupted extract	Null spike; duplicate records; outliers passing basic validation	Inaccurate aggregate valuation
Infrastructure bottleneck	Replication contention; peak-load resource exhaustion	Rising queue depth; replication lag; session contention	Compounding latency under stress; missed SLA

Threshold alerts catch the symptom, not the cause. A fixed limit on job duration flags a slow batch without telling you whether it's a network blip or a genuine schema break. And during real market stress, a rate shock, a credit event, or legitimate price movement spikes across the board, which is exactly when static thresholds fail hardest: they can't tell a systemic market move from a single feed going bad. This is the problem that motivated dynamic, correlation-aware price validation in production NAV processing in the first place. Instead of judging one security's move in isolation, checking it against a basket of correlated instruments tells you whether the whole market moved or just your feed did [2], [3].

The position this paper takes is that recovery-time improvement in regulated financial pipelines comes primarily from dependency-aware diagnosis, not from the sophistication of the anomaly-scoring model on its own. A system that can tell "the sector moved" apart from "the feed broke" closes the real gap between merely flagging something and acting on it safely. This isn't an uncontroversial claim. A systems engineer could reasonably argue that model architecture matters more than dependency awareness, regardless of how the diagnosis layer is built, and the evaluation later in this paper is designed to engage that disagreement head-on rather than assume it away.

This article contributes the following:

- A layered self-healing architecture that keeps telemetry, ML-based detection, dependency-aware diagnosis, and governed remediation as separate, auditable stages a structure suited to regulated environments in a way generic self-healing pipelines are not.
- A dynamic-threshold detection approach adapted from correlated-instrument validation logic used in production financial pricing systems, rather than borrowed wholesale from CI/CD anomaly detection.
- A control loop designed for hybrid environments, where legacy change-data-capture replication feeds cloud-native processing.
- An evaluation protocol combining synthetic fault injection and historical incident replay, grounded in verified production evidence rather than estimated figures.

Section II surveys prior work on threshold monitoring, ML-based anomaly detection, and self-healing automation and identifies where that literature falls short for regulated financial data. Section III lays out the proposed framework. Section IV covers evaluation methodology and results. Section V concludes and points to future work.

II. Related Work

A. Traditional ETL and Threshold-Based Monitoring

Batch ETL still dominates fund accounting and regulatory reporting, mainly because its deterministic execution is easy to audit, even as streaming architectures pick up ground for intraday risk use cases. Monitoring for both patterns has leaned on static thresholds, job duration caps, error-

rate ceilings, and missing-file alerts. These catch known failure modes reliably enough, but they generate excessive false positives under variable load, and they miss anomalies that build gradually rather than announcing themselves as a hard failure [4].

B. Machine Learning-Based Anomaly Detection in Data and DevOps Pipelines

Unsupervised methods like autoencoders, isolation-based scoring, and statistical drift detectors have found traction in CI/CD telemetry and pipeline log analysis precisely because labeled failure examples are scarce and production failures rarely repeat themselves in exactly the same form [5], [6]. Isolation-based scoring in particular has become a common baseline for high-dimensional, mixed-type operational telemetry [5]. The same anomaly-detection pattern extends well beyond software pipelines: comparable unsupervised techniques have been applied to physical industrial pipelines to catch sensor and flow anomalies [7] and separately to secure ETL pipeline design against data-integrity and encryption threats [8], [9], a related but distinct concern from the operational-reliability focus of this paper. Concept-drift detection tackles a further related but distinct problem: separating a genuine shift in the underlying data-generating process from a passing anomaly, which is directly relevant to schema evolution in long-running financial feeds [3].

C. Self-Healing and Automated Remediation in Cloud and DevOps Contexts

The original autonomic-computing vision framed self-managing systems as a necessary response to escalating operational complexity; self-configuration, self-healing, self-optimization, and self-protection were treated as design goals rather than afterthoughts [10]. CI/CD pipelines have since put versions of this vision into practice, closing the loop between detection and automated rollback without waiting on human confirmation and reporting substantial MTTR reductions as a result [1], [11]. Diagnosing which component is actually responsible for an anomaly, rather than merely flagging that one exists, has been addressed through large-scale performance-issue triaging in cloud services [12] and through dependency-graph and causal-discovery methods that identify which upstream component most plausibly explains an anomaly instead of scoring each signal in isolation [13], [14], [15], [16].

Table 2: Comparison of Detection and Remediation Approaches

Dimension	Static-Threshold Monitoring	Generic ML Anomaly Detection (CI/CD-style)	Dependency-Aware Self-Healing (this paper)
Distinguishes market event from feed fault	No	Partial, model-dependent	Yes, a correlated peer check
Remediation	Manual only	Automated rollback (software context)	Governed: auto-correct / auto-quarantine / escalate
Audit trail	Ad hoc	Rarely designed in	Built-in, immutable log
Domain fit	Generic	Generic software/DevOps	Regulated financial data operations

D.

Gaps in Regulated Financial Data Environments

Almost all of this literature was demonstrated in generic cloud-native settings: microservice fleets, CI/CD pipelines, and IoT telemetry, not in regulated financial data operations. Three constraints get lost in translation. Remediation has to leave an audit trail a supervisor can actually review after the fact [17], [18]. False remediation carries a downstream reconciliation cost that a rolled-back web deployment simply doesn't. And legitimate data volatility driven by real market events has no clean equivalent in CI/CD anomaly detection; there's no such thing as a "market-wide event" corrupting a software deployment's metrics. This paper builds its framework around those three constraints directly,

using correlated-instrument validation logic from production financial systems as the connecting thread.

III. Research Methodology

A. Research Gap

Existing self-healing pipeline work shows that closing the loop between detection and remediation cuts recovery time substantially in software delivery [1], [11]. What's missing is guidance on how that loop should be built when the data itself, not just system health metrics, is what's being validated; when large legitimate swings in that data are routine rather than exceptional; and when every remediation action has to be explainable to an auditor after the fact. The framework below is built specifically around those three constraints.

B. Proposed Framework

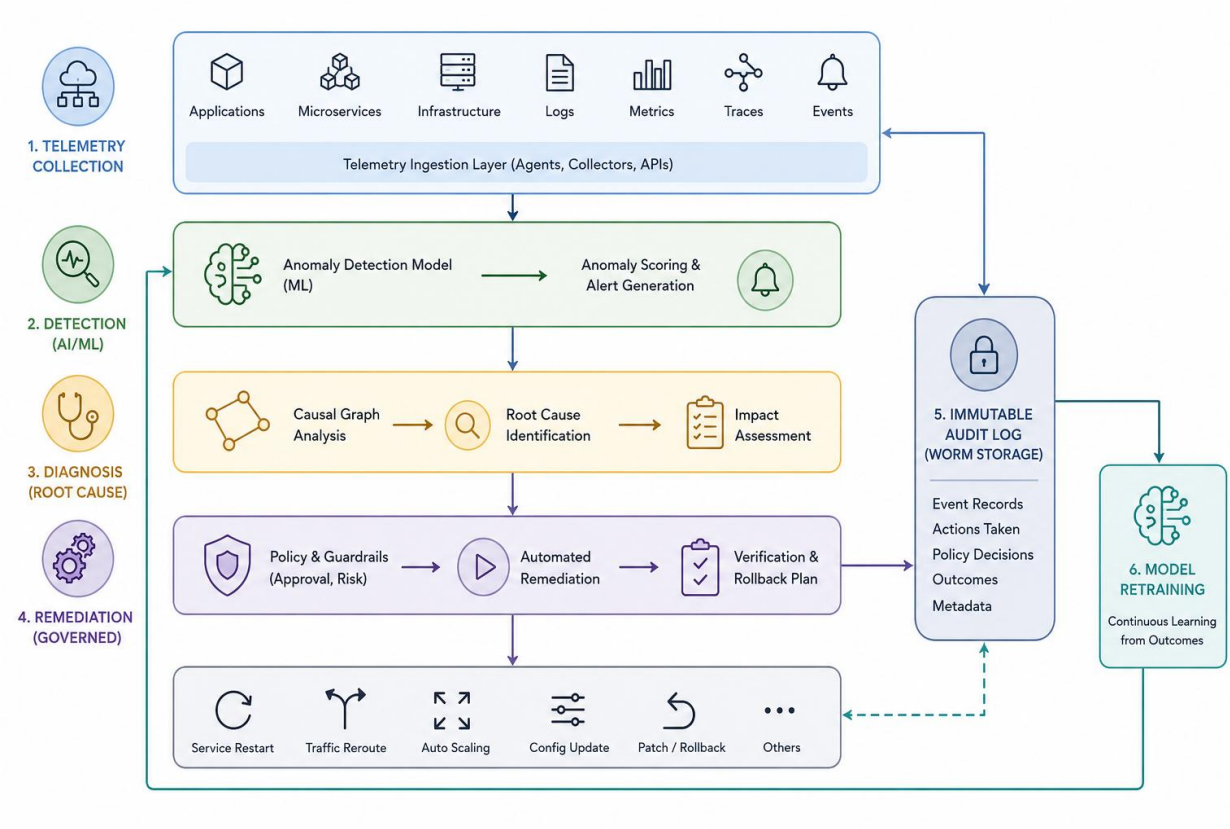

Fig. 1. Layered self-healing architecture

Four concerns get conflated in most monitoring tooling: watching the pipeline, deciding whether something is anomalous, working out why, and deciding what to do about it. Keeping them separate isn't an aesthetic preference; it's what makes each remediation decision independently reviewable later, which matters a great deal when a regulator asks why a specific price value was excluded from a NAV calculation.

C. Real-Time Telemetry and Monitoring Layer

Telemetry spans three levels: job metrics (execution duration, record counts, replication lag), data metrics (null rates, per-field distributional statistics, referential integrity checks), and infrastructure metrics (queue depth, database session contention). In hybrid environments, where change-data-capture replication feeds a cloud-native processing layer, telemetry has to cover both halves consistently a lag introduced at the replication tier is invisible to any monitoring instrumented only downstream of it.

D. ML-Driven Dynamic-Threshold Anomaly Detection

Instead of a single static threshold per metric, the detection layer computes a dynamic acceptable range per indicator using an interquartile-range approach grounded in each instrument's own historical behavior adapted directly from production price-validation practice in NAV processing [2]. Isolation-based scoring [5] runs as a secondary, unsupervised check on the full multivariate feature

vector per pipeline run, catching combinations of individually unremarkable values that are jointly anomalous. In the production application of this dynamic-threshold approach to NAV pricing validation, threshold-based exception review had generated approximately 20,000 pricing exceptions under a static-threshold baseline, of which approximately 19,000 (roughly 95%) were false positives, legitimate price movements incorrectly flagged for manual review, leaving only around 1,000 exceptions that genuinely required analyst attention.

E. Dependency-Aware Root Cause Analysis

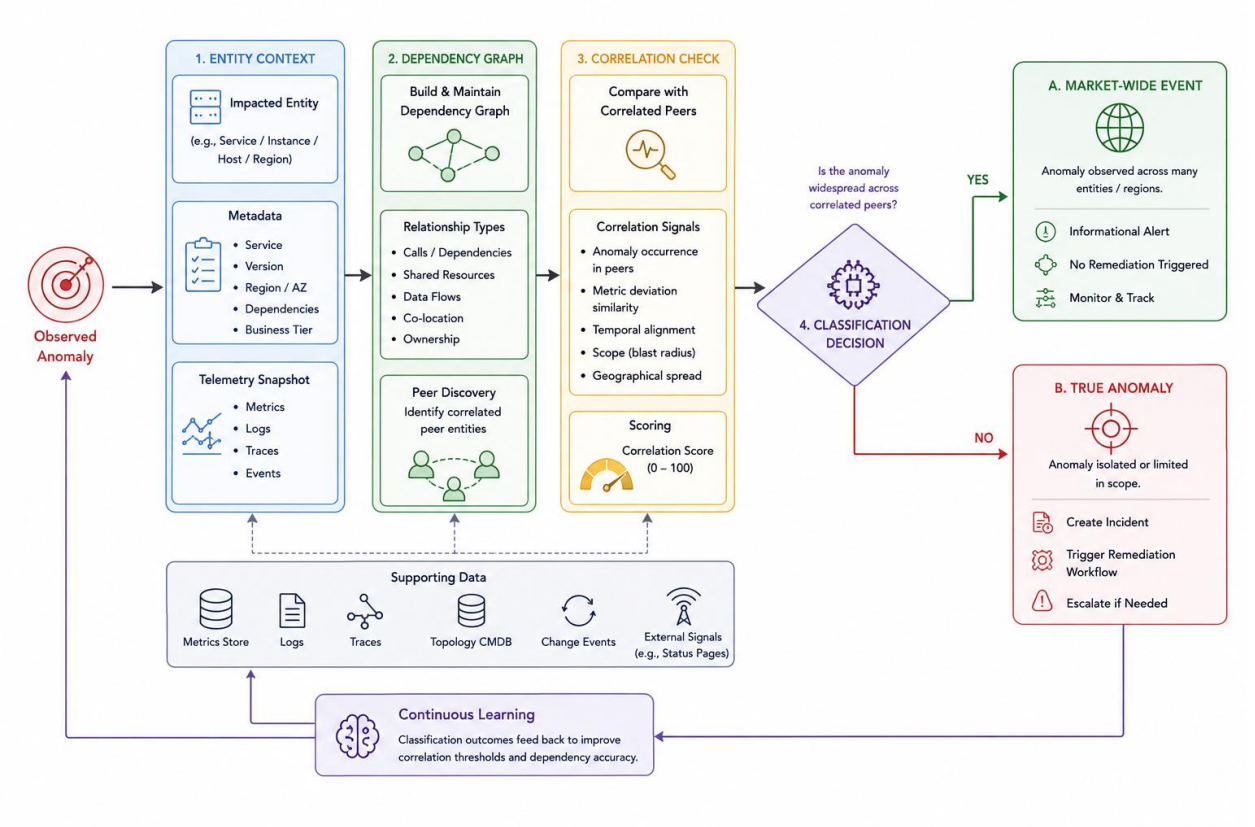

Fig. 2. Dependency-aware diagnosis

Here is where this architecture departs most clearly from generic AIOps practice. Instead of scoring a metric's deviation on its own, the diagnosis layer cross-references correlated data entities for a security and its historically correlated peers; for a feed, it cross-references related feeds sharing an upstream source before calling something a true anomaly. This mirrors validated production logic directly: when a security's price jumps, checking it against a basket of correlated instruments is what tells you whether the sector moved or the feed broke [14], [15]. It's also where this paper's central claim gets operationalized that dependency awareness, more than model sophistication, is what drives recovery-time reduction.

F. Governed Remediation Control Loop

Once diagnosis assigns a cause with enough confidence, the remediation controller chooses among three policies, summarized in Table 3: automatic correction, automatic quarantine, or escalation to a human operator.

Table 3: Remediation Policy Decision Table

Diagnostic Confidence	Feed Criticality	Selected Policy	Rationale
High	Non-regulatory-reporting feed	Automatic correction	Low downstream risk; fast recovery

High	Regulatory-reporting feed	Automatic quarantine + notification	Correction risk outweighs speed benefits.
Low or ambiguous	Any	Escalate to human operator	Insufficient confidence for automated action

Every action, along with the diagnostic evidence behind it, gets written to an immutable audit log not as blockchain-style theater but because supervisory review genuinely needs a reconstructible decision trail consistent with established incident-handling documentation practice [17], [18], [19].

G. Hybrid Legacy and Cloud Integration Considerations

Few financial institutions run one clean, homogeneous pipeline. Change-data-capture replication out of mainframe or Oracle-based source systems into cloud-native processing is the norm, and the self-healing loop needs to instrument both tiers. Cutting replication lag from hours to seconds by eliminating redundant source-side updates is one concrete lever at the legacy tier that directly determines how fast downstream anomalies can even be observed in the first place.

H. Evaluation Methodology

Two complementary approaches drive the evaluation. Synthetic fault injection introduces schema mismatches, delayed feeds, and out-of-range values into a controlled pipeline replica. Historical incident replay runs logged production incidents back through the proposed detection and diagnosis logic to check whether it would have caught and correctly classified each one. The evaluation reports two categories of evidence: verified quantitative reduction in false-positive exception volume and in replication-tier latency, drawn from production experience with correlated-instrument validation logic; and qualitative characterization of detection and recovery speed, since no verified time-to-detect or time-to-recover figure exists for the reference production system and none is estimated here.

IV. Results and Discussion

Table 4: Comparison of Pipeline Outcomes With and Without Dependency-Aware Self-Healing Automation

Failure Scenario	Outcome Without Self-Healing (Static Threshold)	Outcome With Self-Healing (Dependency-Aware)
Market-wide price movement (false-anomaly risk)	Flagged as exception, routed to manual review	Correctly attributed to correlated-instrument movement; not flagged
Feed-specific data corruption	Indistinguishable from market movement under static threshold; delayed manual triage	Isolated from correlated-peer behavior; flagged as true anomaly
Schema drift / upstream change	Detected only once downstream transformation fails	Detected at telemetry layer before propagating downstream
Infrastructure / replication bottleneck	Manual investigation of replication lag	Reduced lag through elimination of redundant source-side updates

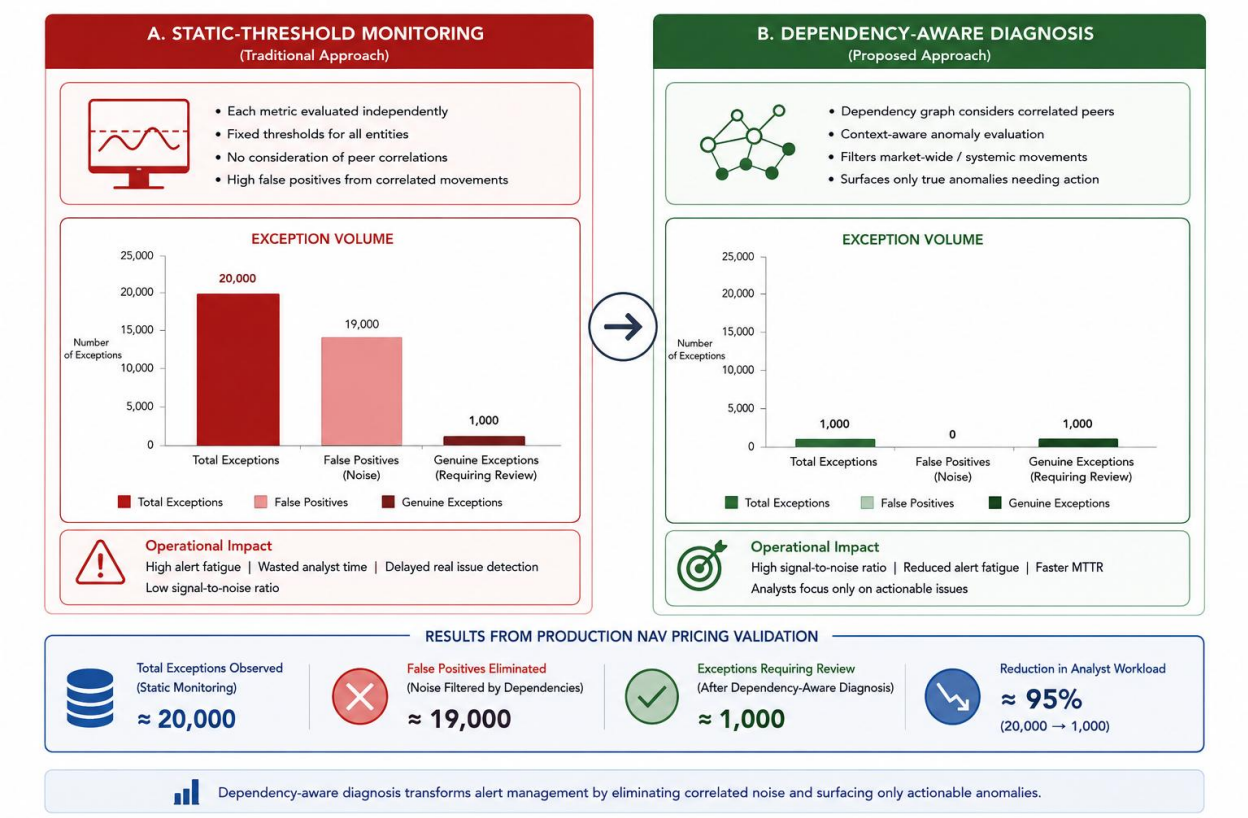

Fig. 3. Exception volume under static-threshold monitoring vs. dependency-aware diagnosis, drawn from production NAV pricing validation experience

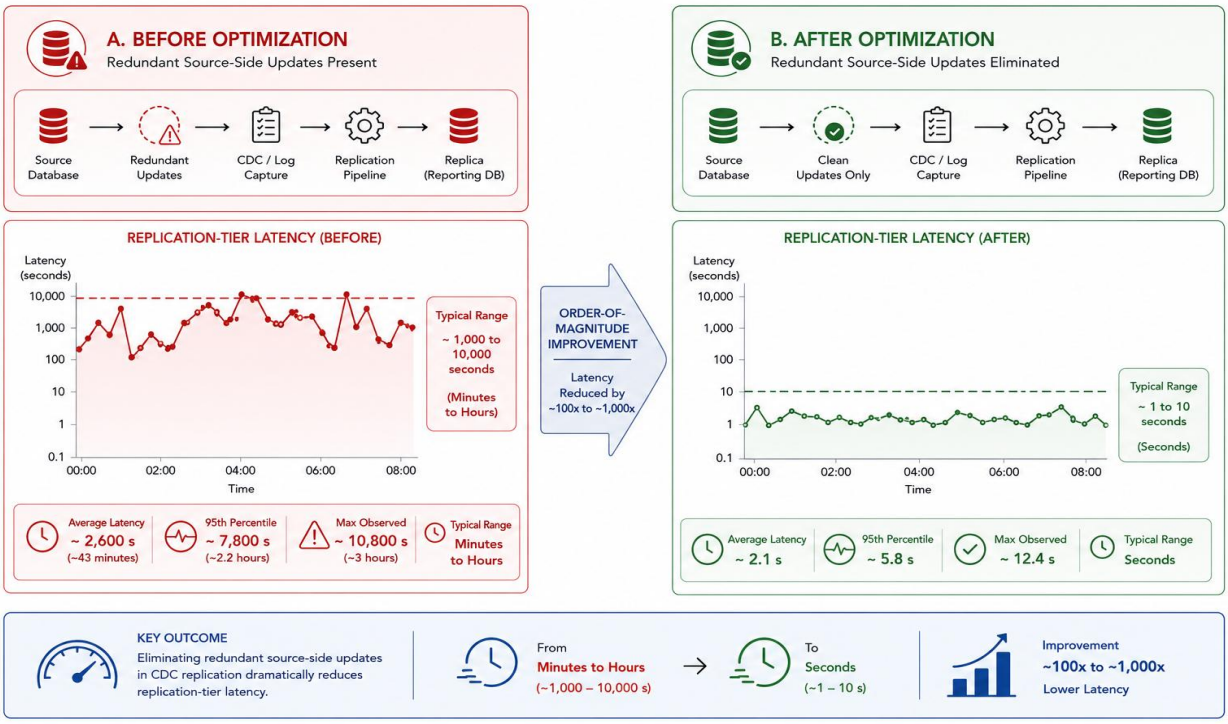

Fig. 4. Replication-tier latency, before and after eliminating redundant source-side updates in change-data-capture replication (illustrative; order-of-magnitude reduction from hours to seconds).

The most direct evidence for this paper's central claim comes from applying dependency-aware, correlation-based validation to production NAV pricing data: of approximately 20,000 exceptions generated under static-threshold monitoring, approximately 19,000, roughly 95%, were false positives attributable to legitimate, correlated market movement rather than genuine data faults. Filtering these through correlated-instrument comparison rather than univariate threshold scoring left approximately 1,000 exceptions genuinely requiring analyst review, a reduction that maps directly onto this paper's argument that dependency-aware diagnosis, not detection-model sophistication alone, is the more decisive lever for reducing false-positive load in regulated financial pipelines. Separately, at the replication tier, eliminating redundant source-side updates reduced change-data-capture lag from hours to seconds, addressing the infrastructure-bottleneck failure mode directly rather than through anomaly scoring.

Detection and recovery speed are reported qualitatively rather than with a specific figure: manual triage of the approximately 19,000 false-positive exceptions under the legacy threshold system consumed substantial analyst time during periods of market volatility, when review capacity was already constrained; automated dependency-aware filtering removes the majority of that triage burden before it reaches an analyst, which is markedly faster in practice than the legacy review cycle, though no verified time-to-detect or time-to-recover figure exists for this reference system and none is asserted here.

Three risks warrant explicit discussion. Cold-start conditions and an insufficient historical baseline for a newly onboarded feed or instrument limit the dynamic-threshold and correlated-peer approach until sufficient history accumulates; a conservative fallback to wider static thresholds during this period is a deliberate design choice. False remediation, where an automatic correction or quarantine is applied to data that was in fact legitimate, carries downstream reconciliation costs disproportionate to a rolled-back software deployment, which is why the governed control loop escalates rather than auto-remediates whenever a regulatory-reporting feed is affected. Model drift under changing market regimes, a correlation structure calibrated during one volatility regime performing poorly during another, requires periodic recalibration against recent market behavior rather than a one-time training pass.

These considerations connect directly to regulatory expectations around data aggregation quality and timeliness [17] and to system resiliency requirements applicable to market-critical infrastructure [18]; a self-healing pipeline's audit trail is a load-bearing component of its regulatory defensibility, not incidental tooling.

V. Conclusion and Future Work

This paper set out a self-healing pipeline architecture for financial data operations built on four separated, auditable stages: telemetry, ML-driven detection, dependency-aware diagnosis, and governed remediation and argued that dependency-aware diagnosis, not detection-model sophistication on its own, is the more decisive factor in reducing recovery time in regulated environments. Verified production evidence supports this: dependency-aware filtering of NAV pricing exceptions eliminated approximately 95% of false-positive review volume, and eliminating redundant replication updates cut change-data-capture lag from hours to seconds.

Two directions stand out for future work. One is extending the remediation layer with generative-AI-assisted incident narratives that summarize a detected anomaly and its supporting evidence in plain language for human reviewers. The other is exploring regulatory-aware autonomous agents that adjust remediation policy on their own as reporting deadlines approach. This shift mirrors a broader move toward AI-assisted operational decision-making already visible in adjacent practitioner domains, such as AI-driven predictive analytics applied to program management in retail supply chains [20], suggesting the underlying pattern generalizes well beyond financial data pipelines. Extending dependency-aware diagnosis beyond NAV and treasury reporting into real-time risk and exposure pipelines is a further avenue that deserves its own dedicated evaluation.

References

[1] A. Challa and M. R. Konatham, "Self-Healing CI/CD Pipelines with Feedback-Loop Automation," International Journal of Intelligent Systems and Applications in Engineering (IJISAE), vol. 12, no. 23s, pp. 3217–3229, 2024.

[2] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly Detection: A Survey," ACM Computing Surveys, vol. 41, no. 3, Art. 15, Jul. 2009, doi: 10.1145/1541880.1541882.

[3] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A Survey on Concept Drift Adaptation," ACM Computing Surveys, vol. 46, no. 4, pp. 1–37, 2014.

[4] A. H. Fawzy, K. Wassif, and H. Moussa, "Framework for Automatic Detection of Anomalies in DevOps," Journal of King Saud University – Computer and Information Sciences, vol. 35, pp. 8–19, 2023.

[5] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation Forest," in Proc. 2008 8th IEEE Int'l Conf. on Data Mining, 2008, pp. 413–422, doi: 10.1109/ICDM.2008.17.

[6] A. Hrusto, E. Engström, and P. Runeson, "Optimization of Anomaly Detection in a Microservice System Through Continuous Feedback from Development," in Proc. 10th IEEE/ACM Int'l Workshop on Software Engineering for Systems-of-Systems and Software Ecosystems, 2022, pp. 13–20, doi: 10.1145/3528229.3529382.

[7] S. S. Aljameel et al., "An Anomaly Detection Model for Oil and Gas Pipelines Using Machine Learning," Computation, vol. 10, no. 8, p. 138, Aug. 2022, doi: 10.3390/computation10080138.

[8] M. R. Sokkula, "Integrating Blockchain and AI for Data Encryption and Secure ETL Pipelines," IJISAE, vol. 13, no. 1, pp. 395–406, 2025.

[9] M. R. Sokkula, "The Role of AI in Strengthening Cybersecurity for Data Pipelines and ETL Systems," IJISAE, vol. 13, no. 1, pp. 357–368, 2025.

[10] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," IEEE Computer, vol. 36, no. 1, pp. 41–50, Jan. 2003, doi: 10.1109/MC.2003.1160055.

[11] A. Capizzi, S. Distefano, L. J. P. Araújo, M. Mazzara, M. Ahmad, and E. Bobrov, "Anomaly Detection in DevOps Toolchain," in Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment (DEVOPS 2019), Lecture Notes in Computer Science, vol. 12055, Springer, 2020, pp. 37–51, doi: 10.1007/978-3-030-39306-9_3.

[12] C. Bansal, S. Renganathan, A. Asudani, et al., "DeCaf: Diagnosing and Triaging Performance Issues in Large-Scale Cloud Services," in Proc. ACM/IEEE 42nd Int'l Conf. on Software Engineering, 2020, pp. 201–210.

[13] J. Soldani and A. Brogi, "Anomaly Detection and Failure Root Cause Analysis in (Micro) Service-Based Cloud Applications: A Survey," ACM Computing Surveys, vol. 55, no. 3, pp. 1–39, 2022, doi: 10.1145/3501297.

[14] Á. Brandón, M. Solé, A. Huélamo, D. Solans, M. S. Pérez, and V. Muntés-Mulero, "Graph-Based Root Cause Analysis for Service-Oriented and Microservice Architectures," Journal of Systems and Software, vol. 159, 110432, 2020, doi: 10.1016/j.jss.2019.110432.

[15] A. Ikram, S. Chakraborty, S. Mitra, S. Saini, S. Bagchi, and M. Kocaoglu, "Root Cause Analysis of Failures in Microservices Through Causal Discovery," Advances in Neural Information Processing Systems, vol. 35, pp. 31158–31170, 2022.

[16] M. Li, Z. Li, K. Yin, X. Nie, W. Zhang, K. Sui, and D. Pei, "Causal Inference-Based Root Cause Analysis for Online Service Systems with Intervention Recognition," in Proc. 28th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining, 2022, pp. 3230–3240.

[17] Basel Committee on Banking Supervision, Principles for Effective Risk Data Aggregation and Risk Reporting (BCBS 239), Bank for International Settlements, Jan. 2013.

[18] U.S. Securities and Exchange Commission, "Regulation Systems Compliance and Integrity," 17 C.F.R. §242.1000–1007, Release No. 34-73639, Nov. 2014.

[19] P. Cichonski, T. Millar, T. Grance, and K. Scarfone, "Computer Security Incident Handling Guide," NIST Special Publication 800-61 Rev. 2, Aug. 2012, doi: 10.6028/NIST.SP.800-61r2.

[20] C. L. Kappani, "Leveraging AI-Driven Predictive Analytics for Effective Program Management in Retail Supply Chains: A Program Manager's Perspective," IJISAE, vol. 14, no. 1s, pp. 295–303, 2026.