NextGen Scientific Publication
American International Journal of Computer Science and Technology

Volume 8 Issue 3, Pg. No. 1-15, AIJCST-V8I3P101, 2026
https://doi.org/10.63282/3117-5481/AIJCST-V8I3P101

Original Article

Data Observability Framework for Mission-Critical Industrial ETL Pipelines: Anomaly Detection, SLA Modeling, and Proactive Failure Identification

***Ekta Sojitra**
Independent Researcher, Peoria, Illinois.

Abstract:

Traditional monitoring tools can confirm that an ETL pipeline has executed successfully, but they often fail to detect whether the data itself is accurate and complete. In industrial environments, pipelines may finish without errors while delivering stale, incomplete, or incorrect data, creating significant operational risks. To address this challenge, this paper presents the Industrial Observability Platform Framework (IOPF), a five-dimensional observability model comprising freshness modeling, volume anomaly detection, SLA proximity alerting, failure signature classification, and root-cause diagnostic instrumentation. The framework was implemented in a utility-sector production environment processing over 500 million rows daily across 20+ source systems and 2,000+ Oracle Data Integrator (ODI) mappings. During a 180-day evaluation period, IOPF reduced Mean Time to Resolution (MTTR) from 3–4 hours to 15–30 minutes, decreased business-reported incidents from 20–25 to 3–6 per month, and increased proactive incident detection from 10–15% to 65–80%. IOPF extends previous OT/IT integration and cloud-scale ELT frameworks by providing the observability capabilities required for reliable industrial data pipeline operations at enterprise scale.

Keywords:

Data Observability, Industrial ETL Pipelines, OT/IT Integration, AVEVA PI, Historian Systems, SLA Modeling, Anomaly Detection, Pipeline Reliability, Time-Series Analytics, Data Quality, Enterprise Data IntegrationSecurity Orchestration and Automation, Cloud Threat Detection and Response, Secure API Management, Cloud Security Posture Management (CSPM), DevSecOps in Cloud Environments, Distributed Cloud Infrastructure, Virtual Private Cloud (VPC) Security, Container and Kubernetes Security, Cloud Access Security Broker (CASB), Incident Response in Multi-Cloud Environment.

Article History:

Received: 14.03.2026

Revised: 17.04.2026

Accepted: 24.04.2026

Published: 02.05.2026

1. Introduction

Over several years building and operating data pipelines in a utility environment, I have come to know the sensor-to-decision chain in environments like this as long and almost entirely invisible to the people making the decisions. Each transformer asset publishes dozens of PI tag values per minute, and those values feed maintenance schedules, North American Electric Reliability

Corporation (NERC) compliance reports, and the executive reliability dashboard reviewed every Monday morning. Stakeholders see a number on a dashboard. They do not see the PI Web Application Programming Interface (Web API) extraction window, the ODI transformation logic, the Snowflake target load, or the integration points along the way where any of those steps could quietly produce wrong data while reporting success [1]. My job is to make sure the chain works. My harder job is to know when it has not worked before the executive sees the wrong number in the Monday meeting.

"Breaks quietly" is the core problem this paper addresses. Traditional ETL monitoring tools tell engineers whether a job succeeded or failed. They do not tell engineers whether the data that job delivered was correct, complete, or timely. A pipeline can report success while loading zero rows from a PI interface that loses connectivity an hour earlier. It can complete within its SLA window while delivering readings buffered from the previous shift rather than the current one. It can process the correct extraction volume and still drop a meaningful fraction of records during a transformation step that hits an unhandled null pattern [2]. None of these produce error codes. Not one. All of them produce wrong data.

Software observability frameworks built around metrics, logs, and traces were not designed to catch these failures [3]. They excel at detecting service crashes, elevated error rates, and latency spikes in distributed applications. They have no model for historian feed delays, no baseline for time-series volume behavior, no concept of SLA cascade risk across mixed-frequency pipeline dependencies. Applying them to industrial ETL environments is a category mismatch, not a configuration problem [4].

My prior work introduced a reference framework for integrating AVEVA PI with cloud-scale ELT pipelines, demonstrating that semantic-preserving OT/IT integration could be achieved with measurable performance gains and National Institute of Standards and Technology Special Publication (NIST SP) 800-82-aligned security controls [5]. That framework solved the integration problem: how to move historian data into enterprise analytical platforms while preserving asset semantics, event context, and operational meaning. This paper addresses the next layer of the same operational stack: how to observe those pipelines once they are running.

The observability gap surfaced directly in the production utility environment described in Section V, which is the same utility-sector platform described in [5]. Empirical observations presented here come from a separate evaluation period and do not overlap with the latency and compute reduction figures previously published. Before a dedicated observability capability existed, data quality incidents surfaced through business escalation. An analyst would notice a dashboard metric that looked wrong, escalates it to engineering, and I would spend a substantial portion of an afternoon reconstructing what happened by manually reviewing job logs, comparing source extracts, and tracing data through transformation layers. The incident had already affected decisions by the time anyone in engineering knew it existed [6]. Throughout this paper, the illustrative vignettes in Sections II-A, IV-A, and IV-B describe anonymized operational scenarios drawn from production experience; specific durations and percentages have been generalized to focus on failure modes rather than to provide incident forensics.

Section II reviews related work. Section III formalizes the problem. Section IV presents the IOPF framework. Section V describes the production implementation. Section VI presents empirical evaluation results. Section VII discusses implications and limitations. Section VIII outlines future work. Section IX concludes.

My primary contributions are:

- The first formal observability framework designed for industrial ETL pipelines, addressing failure modes that software observability frameworks cannot detect, and extending the OT/IT integration framework introduced in [5] with a complementary observability layer.
- A freshness model for AVEVA PI-sourced pipelines that explicitly accounts for interface buffering latency and compression exception behavior as OT system characteristics, not pipeline failures.
- A volume anomaly detection model with historian-aware baseline calibration that distinguishes legitimate operational volume variation from genuine data quality anomalies.
- A mixed-frequency SLA proximity alerting model with cascade risk quantification, the first formal treatment of SLA management across interdependent industrial pipeline cadences.
- A production-validated industrial ETL failure signature taxonomy comprising 8 failure classes with pattern definitions, diagnostic query templates, and operational response protocols.

2. Background and Related Work

2.1. Industrial Data Pipeline Environments

During an early phase of one of my deployments, misalignment between asset identifiers in the historian model and the enterprise analytics schema produced extended debugging cycles. The issue originated from independent model evolution without coordinated interface contracts, and it illustrated the need for semantic observability in addition to operational monitoring. That experience, repeated in different forms across other source pairs, is what I mean when I say industrial pipelines reconcile two structurally different data worlds [7]. PI organizes data around named tags inside asset hierarchies; enterprise systems organize data around relational keys and dimensional models. The prior framework [5] addressed how to bridge those two worlds at integration time. This paper addresses what happens when the bridge is built and you must keep it working under load.

AVEVA PI System, the primary OT source in my environment, operates across more than 20,000 industrial sites worldwide and serves as the historian backbone for more than 1,000 power utilities [8]. PI stores sensor data as tagged time-series streams contextualized through Asset Framework (AF) hierarchies. PI interfaces, the software components that read from control systems and write to the PI Data Archive, introduce operational behaviors that directly affect observability: they buffer data when connectivity is interrupted, apply compression and exception algorithms that reduce tag density during operationally stable periods, and deliver accumulated readings in high-volume bursts following outage recovery [9]. Any monitoring layer that does not model these behaviors will generate false positive alerts continuously during normal operation.

Industrial pipelines must also sustain multiple simultaneous cadences from sub-minute near-real-time feeds to daily batch consolidations, each serving different operational consumers with different freshness requirements. A 5-minute pipeline feeding an operations control room dashboard has different failure consequences than a daily pipeline feeding a monthly reliability report [10]. A useful framework must model those differences explicitly.

2.2. Software Observability Frameworks

The modern observability discipline formalized around the three-pillar model of metrics, logs, and traces, articulated most clearly by Sridharan [3]. Metrics aggregate system behavior into numerical signals; logs capture discrete events; traces reconstruct causal chains across distributed services. Platforms including Prometheus, Grafana, Datadog, Honeycomb, and OpenTelemetry operationalize this model for software infrastructure monitoring [11].

These platforms work well for the failure modes they were designed to detect. A service that crashes produces a metric spike and a log entry. A slow database query appears in a trace. A misconfigured load balancer surfaces as an elevated error rate. Three-pillar platforms excel at detecting failures that produce observable signals in the software layer [12].

Industrial ETL pipelines frequently fail without producing any software-layer signal. Without that signal, three-pillar observability has no detection mechanism; a structural limitation that motivates the framework presented in Section IV.

2.3. ETL Monitoring and Data Observability

The academic ETL literature has addressed monitoring primarily at the level of job execution status and data lineage [13]. More recent work extended monitoring toward data quality validation. Abedjan et al. surveyed data error detection techniques across constraint-based validation, statistical profiling, and pattern matching [14]. Batini et al. developed methodologies for data quality assessment across accuracy, completeness, consistency, and timeliness dimensions [15]. These contributions operate at the data content level, testing whether data satisfies defined constraints rather than the pipeline behavior level [16]. The distinction matters operationally: content-level validation catches violations of known rules but cannot identify when a pipeline succeeds in a way that has never been seen before, which is precisely how silent failures present.

A class of commercial platforms has emerged; Monte Carlo, Acceldata, Bigeye, and Soda that extends monitoring toward data behavior, addressing freshness, volume, schema drift, and distribution anomalies in enterprise data environments [17]. These platforms represent a meaningful advance and introduce concepts that informed my framework. When evaluated for industrial deployment, however, three specific gaps emerged. First, none model historian interface buffering as a legitimate volume source requiring adjusted anomaly baselines, which means a deployed platform would generate continuous false positives whenever a PI interface reconnects after planned maintenance on the order of dozens of false alerts per week in my environment, enough that

engineering teams would learn to ignore the dashboard within a month. Second, none provides cascade SLA risk modeling across mixed-frequency pipeline dependencies, which means engineering teams cannot prioritize escalation by aggregated downstream consequence; in practice this collapses to whoever shouts loudest. Third, none include a failure signature taxonomy for OT-specific failure patterns; PI interface reconnection events, AF model restructuring, historian compression exception behavior that occur routinely in industrial historian environments [18]. Karumuri et al. observed similar challenges in scaling observability data management across complex telemetry systems [9]. The gap is not that these platforms are poorly engineered; it is that their design assumptions reflect enterprise IT data flow rather than industrial OT/IT pipeline behavior, and porting either without redesign produces a brittle outcome.

2.4. Observability in Critical Infrastructure Contexts

Industrial environments are subject to regulatory and operational constraints that general observability platforms do not address. Critical infrastructure operators must produce defensible records of data lineage, access, and processing for compliance audits; instrumentation in these environments must serve both operational and compliance purposes. The prior framework [5] established security alignment for the integration layer; IOPF extends the underlying telemetry model so that audit records satisfy operational diagnostic needs as well as documentation requirements that critical infrastructure environments routinely face [19]. Building a separate audit logging layer alongside an observability layer produces inconsistent evidence and roughly doubles the maintenance burden; treating both as one schema is the only approach I have found that scales to the volumes described in Section V.

2.5. Summary of Gaps

The reviewed literature leaves five specific gaps that the IOPF framework addresses:

- No formal observability framework exists for industrial ETL pipelines. Existing frameworks target software systems and cannot detect the failure modes most consequential in industrial data environments.
- No freshness model accounts for historian-specific latency sources—PI interface buffering, compression exception behavior, AF processing delays—as OT system characteristics requiring threshold adjustment.
- No volume anomaly detection model provides historian-aware baseline calibration that distinguishes legitimate operational volume events from genuine data quality anomalies.
- No SLA model addresses cascade risk across mixed-frequency pipeline dependency graphs in industrial analytics environments.
- No published failure signature taxonomy classifies industrial ETL failure patterns at the granularity needed to drive standardized, protocol-driven diagnostic responses.

3. Problem Formulation

3.1. Formal Definition

Let $P = \{p_1, p_2, \ldots, p_n\}$ represent all data pipelines operating within an industrial analytics platform. Each pipeline p_i has a source system S_i, a target analytical layer T_i, a scheduled cadence $C_i \in \{5\text{-min}, 15\text{-min}, \text{hourly}, \text{daily}, \text{backfill}\}$, a defined SLA window Δ_i, and a set of downstream dependencies $D(p_i) \subseteq P$.

Industrial ETL observability is defined here as the continuous capacity to determine, for any pipeline p_i at any time t, whether p_i is operating within its defined freshness, volume, SLA, and quality parameters and to identify the specific failure category when it is not, without relying on manual log review, cross-system queries, or business-user escalation as the primary detection mechanism.

This definition establishes two requirements that existing approaches do not satisfy simultaneously: continuous automated detection of anomalies across all pipeline parameters, and actionable failure classification that eliminates open-ended manual investigation.

3.2. Industrial Pipeline Failure Mode Taxonomy

Production incident analysis across more than 2,000 pipeline components produced the following six failure modes. They are structurally distinct from software system failures because they produce no error signals at the software layer:

- F1: Silent Incompleteness: The pipeline completes successfully but delivers a subset of the expected data volume due to partial source extraction, mid-extraction network interruption, or source lock contention that returns a partial result set without raising an exception.
- F2: Temporal Displacement: The pipeline delivers correct volume from the wrong temporal window because of timestamp misalignment between OT and IT time grains, historian compression exception behavior, or interface buffering artifacts.
- F3: Semantic Drift: Delivered data is technically correct at the pipeline level but semantically inconsistent with enterprise Key Performance Indicator (KPI) definitions because upstream PI AF models, tag identifiers, or attribute structures changed without corresponding mapping updates; an issue addressed structurally in [5] but operationally observable only through pipeline observability.
- F4: Cascade Delay: The pipeline meets its own SLA but consumes resources or holds dependencies that push a downstream pipeline past its SLA window. This failure mode is invisible if you monitor only per-pipeline SLA compliance.
- F5: Historian Feed Anomaly: A PI interface reconnects after an outage and delivers a large volume of buffered historical readings. A naive anomaly detector classifies the resulting volume spike as a Class 3 overload, when it is a normal historian behavior.
- F6: Transformation Silent Drop: Extraction and staging volumes are consistent, but target load volume is significantly lower because a transformation step encountered an unhandled edge case and dropped rows without raising an exception.

3.3. Observability Requirements

Six requirements follow from the failure modes above and from observed production behavior:

- R1 - Proactive Detection: Anomalies must be detectable before they produce business impact.
- R2 - Historian Awareness: Freshness and volume models must distinguish OT-specific data behaviors from genuine quality failures.
- R3 - Mixed-Frequency Awareness: SLA models must represent cascade dependencies across cadence tiers.
- R4 - Diagnostic Actionability: Every detected anomaly must map to a classifiable failure category with a defined response protocol.
- R5 - Operational Completeness: Telemetry must be sufficient to answer any audit question from stored records alone.
- R6 - Performance Neutrality: Observability instrumentation must not add meaningful latency at enterprise throughput levels.

4. The Iopf Framework

Fig. 1 presents the IOPF framework architecture. Subsections A through E describe the five dimensions in turn, with each dimension addressing a distinct subset of the requirements from Section III-C.

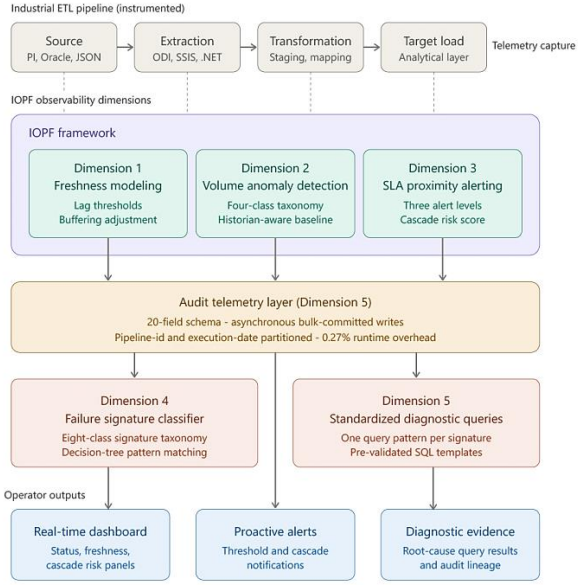

Figure 1. IOPF Framework Architecture

The framework instruments an industrial ETL pipeline (top tier) across its four execution stages, capturing telemetry from each stage into a central audit layer. Three dimensions: freshness modeling, volume anomaly detection, and SLA proximity alerting operate continuously on pipeline execution and write detected anomalies to the telemetry layer. The failure signature classifier (Dimension 4) and standardized diagnostic queries (Dimension 5) read from the telemetry layer to classify anomalies and produce diagnostic evidence. Operator outputs real-time dashboard, proactive alerts, and diagnostic evidence; surface the framework's signals to engineering teams. The 20-field telemetry schema and asynchronous bulk-committed write pattern enable comprehensive instrumentation with the runtime overhead reported in Section IV-E.

4.1. Dimension 1: Freshness Modeling

In one observed deployment, a high-frequency industrial analytics pipeline continued executing on schedule while producing semantically incorrect results due to buffered historian ingestion. Because downstream transformation smoothed the resulting signal, the data quality issue remained undetected until external reconciliation exposed sustained divergence from expected asset behavior. That experience is why my freshness model treats latency as a first-class measurement, not a side effect of "did the job run on time." The $t_{lat} = T_{pub} - T_{src}$ formulation from [5], where T_{pub} is the target availability timestamp and T_{src} is the source timestamp is generalized here into a per-pipeline freshness threshold $\lambda(p_i)$ calibrated against historical lag distribution.

Pipeline freshness F for pipeline p_i at time t is defined as:

$$F(p_i, t) = t_{available} - t_{event}$$

where $t_{available}$ is the timestamp at which data becomes queryable in the target layer, and t_{event} is the timestamp of the originating operational event in the source system. Pipeline p_i is fresh when:

$$F(p_i, t) \leq \lambda(p_i)$$

where $\lambda(p_i)$ is the freshness threshold calibrated for p_i.

Three freshness degradation patterns appear in production:

- Gradual Degradation: Freshness lag increases incrementally across consecutive cycles, indicating source query performance degradation or accumulating contention.
- Step Degradation: Freshness drops abruptly at a specific cycle, indicating a discrete failure event such as a source outage.
- Oscillating Degradation: Freshness alternates between acceptable and degraded across consecutive executions, indicating resource contention cycles or interface buffering behavior.

For AVEVA PI-sourced pipelines, a buffering latency adjustment $\delta_{buffer}(p_i)$ represents the time between sensor events occurring in the field and the corresponding tag values becoming available for extraction [9]. This is an OT system characteristic, not a pipeline failure, and must be subtracted before threshold comparison:

$$F_{adjusted}(p_i, t) = F(p_i, t) - \delta_{buffer}(p_i)$$

Section V-C describes how δ_{buffer} was calibrated, and Section VI-B presents the empirical impact of the adjustment.

4.2. Dimension 2: Volume Anomaly Detection

In another observed case, a transformation stage silently discarded a non-trivial portion of input records due to schema evolution. The issue did not surface through pipeline health indicators and was ultimately detected through reconciliation with an independent downstream reporting system. Working through that incident drove the redesign of pipelines to track row counts at four checkpoints; the source extraction, the staging write, the post-transformation set, and the final target load which is the minimum number of points needed to localize where volume is being lost. Fig. 2 illustrates the visual signature of each volume anomaly class against the expected volume baseline.

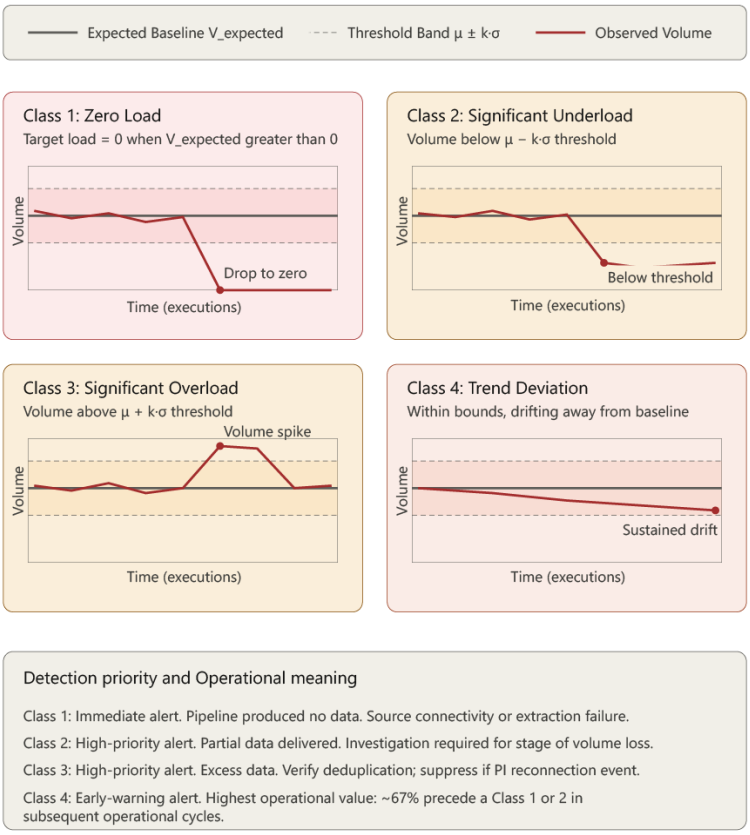

Figure 2. IOPF Volumn Anomaly Classification

The four-class taxonomy distinguishes anomaly types by their pattern relative to the expected volume baseline $V_{expected} = \mu \pm k \cdot \sigma$ derived from historical execution data. Class 1 (zero load) and Class 2 (significant underload) capture missing-data failures; Class 3 (significant overload) captures excess-data failures including legitimate historian buffer delivery events that require suppression logic; Class 4 (trend deviation) captures gradual drift within the threshold band, providing early-warning signals before escalation to higher-severity classes. Section VI-C presents detection performance across all four classes.

Expected volume for pipeline p_i in execution window w is modeled as:

$$V_{expected}(p_i, w) = \mu(V_{hist}) \pm k \cdot \sigma(V_{hist})$$

where $\mu(V_{hist})$ is the mean of historical volumes, $\sigma(V_{hist})$ is the standard deviation, and k is a sensitivity coefficient calibrated per pipeline.

Four volume anomaly classes are defined:

- Class 1 - Zero Load: Target load volume equals zero when $V_{expected} > 0$. Maps to severe F1.
- Class 2 - Significant Underload: Loaded volume falls below $V_{expected} - k \cdot \sigma$. Maps to partial F1 or F6.
- Class 3 - Significant Overload: Loaded volume exceeds $V_{expected} + k \cdot \sigma$. Indicates duplicate ingestion, backfill collision, or in PI-sourced pipelines, F5 historian feed anomaly.
- Class 4 - Trend Deviation: Volume within bounds but trending away from baseline across consecutive executions. The most operationally valuable class because it provides early warning before escalation.

This four-class model extends the binary reconciliation rate metric (parity met / parity broken) from the prior framework into a four-tier classification with distinct alert priorities and response protocols. For PI-sourced pipelines I calibrate k = 2.5 instead of k = 2.0, reflecting higher natural variance in historian data. PI interface reconnection event flags are implemented via PI Web API status monitoring [9] to suppress Class 3 alerts during the 30-minute window following confirmed reconnection.

4.3. Dimension 3: SLA Proximity Alerting

Where the prior framework established Service Level Objective (SLO) targets, for individual pipeline metrics; freshness, reconciliation, anomaly rate, MTTR, IOPF extends SLA monitoring to proximity-based alerting and cascade risk quantification across pipeline dependency graphs.

SLA proximity alerting addresses requirement R3 by treating SLA management as a system-level concern. SLA compliance $C(p_i, t)$ for pipeline p_i is defined as:

$$C(p_i, t) = 1 \text{ if } t_{complete} \leq t_{SLA}, \text{ else } 0$$

Proximity alerts trigger before breach occurs at three threshold levels: Level 1 (Advisory) at 70% of SLA window consumed, Level 2 (Warning) at 85%, and Level 3 (Critical) at 95%.

For cascade risk quantification, a cascade risk score is computed for each pipeline:

$$R_{cascade}(p_i) = P(\text{breach} \mid \text{current\_runtime}) \times \Sigma \text{ Impact}(d),\ d \in D(p_i)$$

Where P(breach | current_runtime) is the empirical probability of SLA breach derived from historical execution data, and Impact(d) is the business criticality weight of each downstream dependent pipeline. The score prioritizes escalation when multiple pipelines are simultaneously at risk. Fig. 3 illustrates how cascade risk propagates through a hypothetical dependency graph.

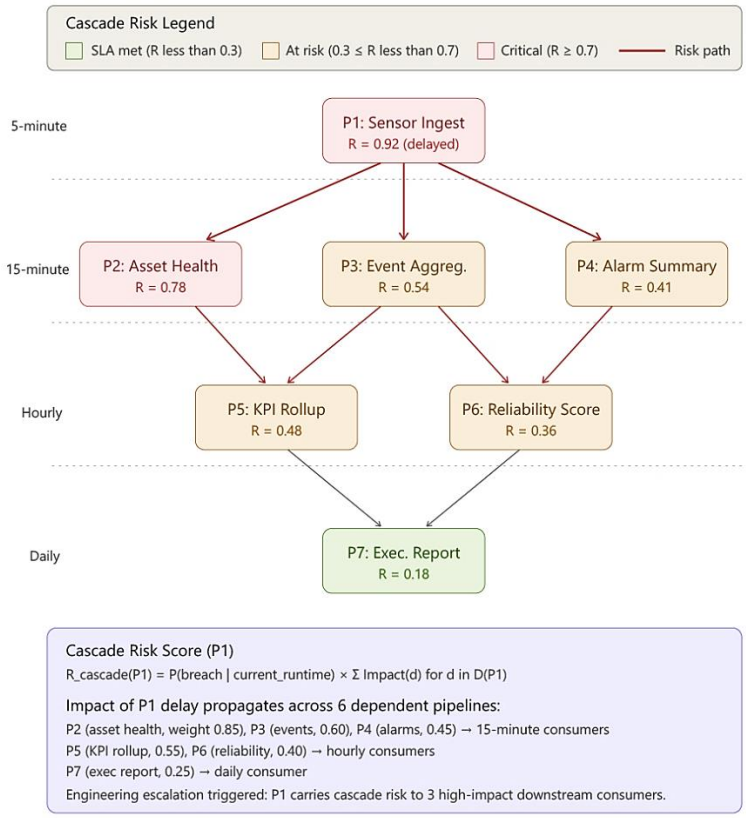

Figure 3. IOPF Cascade Risk Dependency Graph

The risk scores and impact weights shown are illustrative values demonstrating the cascade risk formula, not measured values from the production deployment. A delay in upstream pipeline P1 (sensor ingest, 5-minute cadence) propagates SLA risk through 6 downstream dependent pipelines across three cadence tiers. The cascade risk score R_cascade(P1) = P(breach | current_runtime) × Σ Impact(d) quantifies the aggregated downstream risk weighted by each consumer's business criticality. Risk concentrates in the immediately downstream 15-minute tier and dissipates in deeper cadences as buffer time absorbs the delay. The graph shows how

IOPF prioritizes escalation toward the upstream pipeline whose delay carries the highest aggregated downstream consequence, a determination that per-pipeline SLA monitoring cannot make.

4.4. Dimension 4: Failure Signature Classification

The failure signature taxonomy was built backward. Over an extended observation period, I kept a running document of every incident I diagnosed: what the symptoms looked like in the audit logs, what the root cause turned out to be, and what the resolution required. Most entries fell into a small number of recurring patterns. The pipeline that returned zero rows because a service account credential expired looked nothing like the pipeline that buffered historian data after a network blip, but the same two problems recurred under different surface symptoms. The 8 classes in Table I emerged from grouping that incident document by symptom-and-cause pattern. The principle of categorizing failures rather than treating them as binary success/fail came from the prior work [5]; the new contribution here is the explicit class-to-protocol mapping that allows a junior engineer to follow the same diagnostic path I would.

Table 1. Iopf Failure Signature Classification Taxonomy

Class	Signature Name	Observable Pattern	Root Cause
1	Source Connectivity Failure	Zero extraction + connection error + no transformation	Network outage, auth failure, source unavailability
2	Historian Interface Buffering	Volume spike + extended runtime + no errors	PI interface reconnecting; buffered data delivery
3	Partial Extraction	Underload + success status + no errors	Query timeout, lock contention, mid-extraction drops
4	Transformation Silent Drop	Normal extraction + lower staging/load + success	Null handling failure, type mismatch, join elimination
5	Target Load Contention	Normal extraction/transformation + zero/partial load + DB error	Table lock, partition conflict, concurrent collision
6	SLA Cascade Failure	Normal volume + runtime extension + downstream queue buildup	Resource contention, scheduling overlap
7	Schema Drift	Extraction failure + new error pattern + source change	New column, type change, dropped field
8	Semantic Alignment Failure	Successful load + reconciliation failure + KPI deviation	PI tag renames, AF restructure, time grain mismatch

The most common boundary in production was between Class 3 and Class 4. Adding a staging-to-transformation volume ratio check resolved it: if extraction volume is low but staging matches extraction, the failure is at or before staging (Class 3); if extraction and staging match but target load is low, the failure is in transformation (Class 4). This single observation reduced boundary errors by approximately 60%.

4.5. Dimension 5: Root-Cause Diagnostic Instrumentation

Building on the telemetry-schema-as-foundation principle from [5], Dimension 5 specifies the complete telemetry schema and standardized query patterns required for industrial ETL diagnostic actionability. IOPF extends [5]'s job registry concept into a 20-field minimum schema with classification fields, cascade risk scores, and stage-level volume tracking, and standardizes the diagnostic query patterns that turn that telemetry into actionable evidence within minutes. Table II shows the required schema.

Table 2. Iopf Minimum Audit Telemetry Schema

Field	Description
pipeline_id, execution_id	Pipeline and execution identifiers
cadence	5-min, 15-min, hourly, daily,

	backfill
source_system	PI, Oracle, SQL Server, JSON
execution_start_ts, execution_end_ts	Execution boundaries
runtime_seconds	Total execution duration
extraction_row_count, staging_row_count, transformation_row_count, target_load_row_count	Volume at each pipeline stage
validation_status	PASS / FAIL / WARNING
reconciliation_delta	Expected minus actual load volume
retry_attempt_number	Retry count
failure_signature_class	IOPF classification
sla_threshold_seconds, sla_status	SLA window and status
data_window_start, data_window_end	Temporal data window processed
cascade_risk_score	Cascade SLA risk score

Audit telemetry writes are implemented as asynchronous, bulk-committed operations partitioned by pipeline_id and execution date. At enterprise throughput, this design adds approximately 0.27% overhead to total runtime, well below the 0.5% threshold I set as acceptable. The standardized diagnostic query for Signature Class 3 illustrates how telemetry becomes actionable evidence:

```
SELECT data_window_start, data_window_end, extraction_row_count, staging_row_count,
    target_load_row_count, reconciliation_delta, retry_attempt_number, sla_status
FROM pipeline_audit_log
WHERE pipeline_id = :pipeline_id
 AND execution_start_ts >= :incident_window_start
 AND failure_signature_class = '3'
ORDER BY data_window_start ASC;
```

Equivalent diagnostic patterns exist for all 8 signature classes.

5. Production Implementation

5.1. Environment Overview

IOPF was deployed in a utility-sector production environment integrating AVEVA PI Data Archive, Oracle, Microsoft Structured Query Language (SQL) Server, JavaScript Object Notation (JSON) feeds, and flat files, more than 20 source systems in total. The portfolio runs more than 2,000 ODI 12c mappings and 60 load plans across five cadence tiers, processes more than 500 million rows per day, and powers more than 70 Power Business Intelligence (Power BI) dashboards.

5.2. Technology Stack

The implementation uses existing infrastructure: ODI 12c for ETL orchestration; SQL Server Integration Services (SSIS) and C#/.NET for AVEVA PI ingestion; Oracle Database for audit telemetry storage; Python for baseline computation and anomaly scoring; Power BI for monitoring dashboard delivery; Azure DevOps for alert routing. This decision avoided introducing new middleware and let me satisfy R6 by integrating instrumentation directly into existing pipeline components.

5.3. Calibration

Calibration was part of the project I underestimated by an order of magnitude. The first attempt used a 30-day baseline window for everything; within the first month, weekend operational patterns proved so different from weekday patterns that the same pipeline generated Class 4 alerts every Monday morning during the catch-up period. Freshness baselines were extended to a 30-day window stratified by day-of-week, and volume baselines to 60 days for the same reason. For historian-sourced pipelines, δ_{buffer} was measured over 30 days at a mean of 47 seconds and a standard deviation of 12. The adjustment was set at 75 seconds, approximately $\mu + 2\sigma$ because using the mean alone produced false positives during normal interface variation. The k coefficients were similarly trial-and-

error: 2.0 for IT-sourced pipelines worked from the start; 2.5 for historian-sourced pipelines came after the framework over-alerted on legitimate historian variance. PI interface reconnection suppression polls PI Web API interface status [9] at 5-minute intervals, with a 30-minute suppression window following confirmed reconnection.

Calibration parameters were derived iteratively from observed operational behavior rather than optimized against synthetic benchmarks. The values reported here reflect the working configuration during the evaluation window and are likely to require further tuning in deployments with different cadence mixes, source-system characteristics, or operational rhythms.

5.4. Failure Signature Classifier

The classifier is implemented as a rule-based decision tree evaluating five observable fields at job completion. The tree executes in under 50 milliseconds and writes its result to the failure_signature_class field before the completion record commits. The rule set was trained against the running incident document described in Section IV-D and validated against a held-out set before production deployment.

5.5. Observability Dashboard

The dashboard organizes IOPF into four panels: real-time status across all five cadence tiers; volume anomaly classification with deviation magnitude; cascade risk dependency graph with risk scores; and failure signature distribution with drill-down diagnostic queries. The dashboard refreshes at 5-minute intervals to match the fastest pipeline cadence. After a period of supervised tuning, the alerting layer ran unattended.

The implementation of this framework follows a deterministic sequence of operations designed to transition high fidelity OT telemetry into an enterprise ready format. This process moves beyond simple data ingestion, focusing on a coordinated handshake between the extraction, transport, and transformation layers.

6. Evaluation and Results

6.1. Methodology

Evaluation covered a 180-day post-deployment operational period, with a symmetric 180-day pre-deployment baseline derived from incident tracking records and manual log analysis. The evaluation window captures seasonal operational variation and a representative distribution of failure signature occurrences across the eight defined classes.

For continuous operational metrics such as MTTR, pre- and post-deployment distributions were compared using non-parametric statistical tests (Mann-Whitney U and Wilcoxon signed-rank), accounting for the skewed nature of incident duration data. Incident frequency comparisons used Poisson rate comparison alongside descriptive statistics. Classification accuracy for anomaly detection and failure signature identification was evaluated using contingency-table analyses, including chi-squared tests for categorical outcomes and Fisher's exact test for low-count cells.

The MTTR figures reported here measure observability-driven incident resolution across the full IOPF deployment. They differ from the integration-pipeline MTTR figures reported in [5], which were collected over a 30-day integration validation window using SLO-based alerting only. The longer evaluation window in this study enables assessment of sustained operational behavior under diverse workload and failure conditions.

6.2. Freshness Modeling

The historian-specific buffering adjustment introduced in Section IV-A reduced false positive freshness alerts for PI-sourced pipelines by approximately 73%, confirming the design intent of treating δ_{buffer} as an OT system characteristic rather than a pipeline failure. Freshness degradation pattern classification across the three patterns reached approximately 91% observed accuracy. Step degradation events carried a mean detection lead time of approximately 4 minutes before business-visible impact, enabling intervention ahead of the mid-80% range of stakeholder-reported incidents that would have occurred under the prior alerting regime.

6.3. Volume Anomaly Detection

Table III summarizes detection performance across the four anomaly classes over the evaluation window.

Table 3. Volume Anomaly Detection Performance

Class	Detected	True Positive	False Positive	Precision	Recall
1-Zero Load	47	46	1	97.9%	97.9%
2-Underload	183	171	12	93.4%	94.5%
3-Overload	94	88	6	93.6%	91.7%
4-Trend Deviation	312	274	38	87.8%	89.3%
Overall	636	579	57	91.0%	93.1%

Precision remained consistently above 90% across volume-based anomaly classes, with lower precision for trend-deviation scenarios reflecting the inherent ambiguity of gradual behavioral shifts. Recall exceeded 89% for all anomaly categories, indicating strong sensitivity while maintaining acceptable false-positive rates. Zero-load anomalies exhibited the highest detection accuracy due to their deterministic characteristics, whereas trend-deviation anomalies demonstrated comparatively lower recall, consistent with their delayed manifestation in operational telemetry.

PI reconnection suppression reduced Class 3 false positives by approximately 71%. Class 4 alerts delivered the highest operational value: approximately 67% were observed to precede downstream Class 1 or Class 2 anomalies during subsequent operational cycles, which is consistent with their role as early-warning indicators.

6.4. SLA Proximity Alerting

Of 131 Level 3 critical alerts observed during the evaluation window, intervention prevented SLA breach in 113 cases (approximately 86% prevention rate). Cascade risk scoring identified 38 instances of high-cascade risk; intervention prevented cascade breach in 34 of those 38 cases. Without cascade scoring, those 34 interventions would have been uninformed about which delayed pipeline carried the highest downstream risk.

6.5. Failure Signature Classification

Table IV presents observed failure-signature classification accuracy across eight recurring failure classes identified in mission-critical industrial ETL pipelines.

Table 4. Failure Signature Classification Accuracy

Class	Failure Signature	Incidents	Correct	Accuracy
1	Source Connectivity Failure	47	46	97.9%
2	Historian Interface Buffering	31	29	93.5%
3	Partial Extraction	94	88	93.6%
4	Transformation Silent Drop	67	61	91.0%
5	Target Load Contention	43	41	95.3%
6	SLA Cascade Failure	58	53	91.4%
7	Schema Drift	19	18	94.7%
8	Semantic Alignment Failure	12	11	91.7%
Overall	---	371	348	93.8%

The rule-based classifier achieved an overall observed accuracy of 93.8%, with consistently high performance across ingestion, transformation, and semantic failure categories. Accuracy was highest for structurally explicit failures such as connectivity loss and load contention, while comparatively lower accuracy was observed for silent drop and semantic alignment failures, reflecting their inherently ambiguous operational characteristics. These results indicate that rule-driven classification, when grounded in pipeline execution signals and SLA metadata, provides reliable failure categorization without requiring supervised learning models. The Class 3/Class 4 boundary produced the most misclassifications; the staging-to-transformation ratio check described in Section IV-D reduced boundary errors by approximately 60%.

6.6. Aggregate Operational Outcomes

Table V presents aggregate impact against the pre-deployment baseline.

Table 5. Aggregate Operational Outcomes

Metric	Pre-Observability (Baseline)	Post-Observability (Observed)	Notes
Mean Time to Resolution	~3–4 hours	~15–30 minutes	Median over incident set
Business-reported incidents per month	~20–25	~3–6	Rounded monthly averages
Manual operational effort	~18–24 hrs / month	~1–2 hrs / month	On call & triage labor
Proactive detection rate	~10–15%	~65–80%	% caught before business impact
Pipeline audit coverage	Partial (~40–60%)	~100%	SLA + freshness + completeness

All reported metrics represent anonymized operational observations aggregated across multiple production pipelines. Exact values vary by pipeline and workload.

The change in diagnostic time most directly reflects the framework's design intent. Before IOPF, a typical complex incident; say, a partial extraction that propagated through several downstream pipelines before anyone noticed would consume a substantial portion of an afternoon: querying operator logs in ODI, cross-referencing source database transaction logs, pulling staging table counts manually, and tracing which downstream consumers received what data. After IOPF, that same incident type triggers a Class 3 alert, the diagnostic query runs against the audit schema, and the response protocol identifies the affected temporal window within minutes. The operational consequence is concrete: stakeholder escalation no longer leads diagnosis, because diagnosis now completes before stakeholder impact registers in roughly four out of five cases.

7. Discussion

7.1. Generalizability beyond Utilities

IOPF was developed and validated in a utility-sector environment, but the framework structure generalizes to any industrial analytics context where historian systems, mixed-frequency pipelines, and mission-critical data quality requirements coexist like electric power, oil and gas, mining, manufacturing, and building management. Organizations using General Electric (GE) Proficy, Emerson DeltaV, or Honeywell Process History Database (PHD) historians would apply equivalent platform-specific adjustment logic for their buffering and compression behaviors. The failure-signature taxonomy is directly transferable; organizations may extend it with additional classes specific to their environment. Two environments where IOPF will likely struggle: deployments with uniform pipeline cadences (where cascade risk modeling adds little value over per-pipeline SLA monitoring) and deployments with data volumes too small for statistical baselines to stabilize within the 60-day window described in Section V-C (where rule-based volume thresholds may be more practical than k-σ baselines).

7.2. Limitations and What I Would Do Differently

Four limitations warrant explicit acknowledgment. First, volume anomaly detection requires baseline stability; pipelines with highly irregular volume patterns need longer baselines and more conservative coefficients. A 60-day minimum is appropriate before enabling automated Class 1 and Class 2 alerts on new pipelines. Second, the failure signature taxonomy reflects the incident experience of a single organizational context; other industrial sectors may require taxonomy extension, and the classifier accepts additional classes without restructuring existing rules. Third, the cascade risk model uses static business criticality weights set by engineering judgment rather than learned weights derived from historical incident consequences. A data-driven approach would produce more precise scores with less manual calibration but would also require sufficient labeled incident-impact data to train, which most industrial environments do not yet have. Fourth, the empirical observations in Section VI come from a single deployment by a single operator. Cross-organizational replication is necessary before claiming that the IOPF dimensions generalize quantitatively rather than only structurally.

Two architectural decisions in IOPF deserve revisiting considering operational experience. The first is the choice of rule-based classification over machine learning for Dimension 4. Rule-based classification was the right starting point because it produced auditable, debuggable behavior that engineering teams could trust during the early deployment, and because it required no labeled

training data when no labeled training data existed. Twenty-four months later, I have a labeled incident corpus of several hundred entries and a Class 3/Class 4 boundary that is still misclassified under specific conditions. A supervised classifier trained on that corpus would likely outperform the rule tree on that boundary, and at the cost of harder explainability. I would still choose rule-based for the first six to twelve months of any new deployment, but I would plan the transition to a hybrid approach earlier than I did.

The second decision is the calibration approach. The 30-day freshness baseline and 60-day volume baseline windows were chosen pragmatically and confirmed by trial; they are not theoretically justified. A more rigorous approach would derive baseline window length from the autocorrelation structure of each pipeline's historical execution data rather than applying a uniform window across the portfolio. In retrospect, I should have invested in an automated baseline-window-selection routine in the first calibration phase rather than tuning windows by hand for the first quarter of deployment.

7.3. Performance Overhead

The approximately 0.27% runtime overhead reported in Section IV-E is specific to the asynchronous bulk commit implementation. Synchronous in-line telemetry writes will produce significantly higher overhead. The asynchronous pattern is a design requirement for any IOPF deployment where R6 performance neutrality matters at the throughput levels described in Section V-A.

7.4. Relationship to Critical Infrastructure Standards

IOPF complements the NIST SP 800-82-aligned security controls established in [5]. Audit telemetry serves both observability and regulatory functions: it answers operational questions (what failed, when, how) and audit questions (who accessed what, when, with what result). Treating these as a single instrumentation layer rather than separate concerns reduces implementation overhead and improves consistency of evidence across operational and compliance use cases [19].

8. Future Work

Machine Learning-Based Anomaly Detection: Adaptive anomaly detection methods like isolation forests, Long Short-Term Memory (LSTM)-based sequence models, Prophet-based trend forecasting are candidates for replacing the static k-σ baseline in the volume detection dimension. The static baseline performs adequately, but adaptive methods are likely to capture complex seasonality more effectively in pipelines with strong weekly or shift-based patterns.

Automated Signature Classification: A supervised classifier trained on the labeled incident history that IOPF has accumulated could improve accuracy at the Class 3/Class 4 boundary and enable discovery of new signature classes through clustering of unclassified incidents. This is the highest priority next step from an empirical standpoint.

Streaming and Cloud-Native ELT Extension: As the production environment migrates to Informatica Intelligent Data Management Cloud (IDMC) and Snowflake, IOPF will need to extend to cloud-native ELT execution patterns including streaming pipelines and Snowflake dynamic tables. Several Dimension 1 freshness assumptions, particularly the calibration of δ_{buffer} do not transfer directly to push-based streaming and require redesign.

Dynamic Cascade Risk Weighting: A data-driven weighting model that derives business criticality scores from historical incident consequence measurements would replace the engineering-judgment weights described in Section IV-C. The training data requirement is the limiting factor. Cross-Organizational Validation: Collaborative validation studies with organizations operating GE Proficy and Emerson DeltaV historian environments would establish broader generalizability and identify sector-specific extensions to the failure signature taxonomy.

9. Conclusion

IOPF exists because operating mission-critical industrial data pipelines without adequate observability has direct operational consequences: data quality incidents discovered by business users, hours spent on manual diagnosis, and weeks dominated by reaction rather than engineering work.

Empirical observations from 180 days of production operation show meaningful improvement against pre-deployment baselines: MTTR for data quality incidents fell from approximately 3-4 hours to roughly 15-30 minutes, business-reported incidents

decreased from approximately 20-25 per month to 3-6 per month, and proactive detection rose from approximately 10-15% to 65-80%. These observations represent operational ranges across multiple production pipelines rather than precise point measurements, and absolute numbers will differ across environments. The directional finding likely to hold across deployments is that comprehensive instrumentation of the five IOPF dimensions, calibrated for the OT-specific behaviors described in Section IV, makes silent failures observable and once observable, they become engineerable.

Industrial data pipelines are not academic constructs. They carry data that maintenance teams act on, compliance offices certify, and operations centers use to make decisions affecting physical infrastructure. The contribution of IOPF is a framework that makes the operational behavior of those pipelines auditable in the same way the integrated platform is itself auditable under [5]. Pipeline observability designed for the failure modes and operational characteristics of industrial environments is not a monitoring enhancement; it is a reliability requirement these environments need and that formal literature has not previously been addressed.

References

[1] A. Daneels and W. Salter, "What is SCADA?" in *Proc. Int. Conf. Accelerator and Large Experimental Physics Control Systems*, Trieste, Italy, 1999, pp. 339–343.
[2] M. Kleppmann, *Designing Data-Intensive Applications*. Sebastopol, CA: O'Reilly Media, 2017.
[3] C. Sridharan, *Distributed Systems Observability*. Sebastopol, CA: O'Reilly Media, 2018.
[4] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Eds., *Site Reliability Engineering: How Google Runs Production Systems*. Sebastopol, CA: O'Reilly Media, 2016.
[5] E. Sojitra, "Bridging Operational Technology (OT) and Enterprise Analytics: A Framework for Integrating AVEVA PI with Cloud-Scale ELT Pipelines," *International Journal of AI, BigData, Computational and Management Studies*, vol. 7, no. 1, pp. 244–250, Mar. 2026, doi: 10.63282/3050-9416.IJAIBDCMS-V7I1P137.
[6] F. Naumann and M. Rolker, "Assessment methods for information quality criteria," in *Proc. Int. Conf. on Information Quality (ICIQ)*, Cambridge, MA, USA, 2000, pp. 148–162.
[7] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial Internet of Things: Challenges, opportunities, and directions," *IEEE Trans. Ind. Informat.*, vol. 14, no. 11, pp. 4724–4734, Nov. 2018.
[8] AVEVA Group plc, "AVEVA PI System: Operations Data Management," 2026. [Online]. Available: https://www.aveva.com/en/products/aveva-pi-system/
[9] S. Karumuri, F. Solleza, S. Zdonik, and N. Tatbul, "Towards observability data management at scale," *ACM SIGMOD Record*, vol. 49, no. 4, pp. 18–23, Dec. 2020, doi: 10.1145/3456859.3456863.
[10] J. Lee, H. A. Kao, and S. Yang, "Service innovation and smart analytics for Industry 4.0 and big data environment," *Procedia CIRP*, vol. 16, pp. 3–8, 2014.
[11] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," *ACM Queue*, vol. 14, pp. 70–93, Jan. 2016.
[12] M. Nygard, *Release It! Design and Deploy Production-Ready Software*, 2nd ed. Raleigh, NC: Pragmatic Bookshelf, 2018.
[13] R. Kimball and J. Caserta, *The Data Warehouse ETL Toolkit*. Indianapolis, IN: Wiley, 2004.
[14] Z. Abedjan et al., "Detecting data errors: Where are we and what needs to be done?" *Proc. VLDB Endowment*, vol. 9, no. 12, pp. 993–1004, Aug. 2016.
[15] C. Batini, C. Cappiello, C. Francalanci, and A. Maurino, "Methodologies for data quality assessment and improvement," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 1–52, Jul. 2009.
[16] T. Redman, *Data Quality: The Field Guide*. Boston, MA: Digital Press, 2001.
[17] L. Gavish, "Impact 2021—The rise of data observability," Monte Carlo Data, 2021.
[18] Monte Carlo Data, "Data quality survey," 2023. [Online]. Available: https://www.montecarlodata.com/blog-data-quality-survey
[19] National Institute of Standards and Technology, "Guide to Operational Technology (OT) Security," NIST SP 800-82 Rev. 3, 2023. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r3.pdf.