Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

Observability-Centric SRE Architecture for Cloud-Native Financial Platforms: A Unified Telemetry Framework Using Metrics, Logs, and Distributed Traces

[1]Muhammed Yunas, [2]Muhammed Yunas Chirayath Meerankunju

[1]Principal Site Reliability Engineer | Enterprise Financial Technology Platform
[2]Independent Researcher, USA

ARTICLE INFO

Received: 03 June 2026

Revised: 17 July 2026

Accepted: 20 July 2026

ABSTRACT

Financial transaction platforms operating at scale expose a gap that current monitoring stacks do not close well: metrics, logs, and traces are individually mature signals, but the tooling that handles them has remained siloed [1][2]. On-call engineers responding to a live incident must context-switch across Prometheus/Grafana, the ELK Stack, and Jaeger or Tempo -- each with its own query language, each holding a different piece of the puzzle [3][5]. This paper presents a production-derived architecture that unifies all three signals under a single programmatic query abstraction layer, applies data mesh ownership principles [4] to telemetry governance, and incorporates machine learning-based anomaly detection calibrated to the seasonal periodicity of payment workloads [11][12]. Unlike prior systems that offer UI-level signal correlation or require proprietary vendor lock-in [8][9], the proposed design introduces a vendor-neutral, programmatic cross-signal query interface built entirely on open-source components -- a distinction that makes it both automation-friendly and deployable within regulated financial environments. The system was evaluated over a 90-day production deployment processing 10,000 to 50,000 transactions per second. Results show a 73% reduction in mean time to detection (MTTD), 70% reduction in mean time to resolution (MTTR), false positive rate down from 84.8% to 11.2%, and a 75% decrease in total daily alert volume alongside a 48% increase in actionable alerts [11][12]. Statistical validation confirms F1 = 0.92 (95% CI: [0.90, 0.94]) for the full hybrid detection model. We discuss implementation decisions, failure modes, cardinality management, and the organizational dynamics of adopting data mesh telemetry governance in a regulated environment.

Index Terms: observability, site reliability engineering, distributed tracing, Prometheus, Grafana, Elasticsearch, Tempo, AIOps, data mesh, cloud-native systems, OpenTelemetry, financial platforms, anomaly detection, SLO engineering, Isolation Forest, Prophet.

1. INTRODUCTION

1.1 Motivation and Problem Context

Payment processing systems, real-time fraud detection pipelines, and settlement engines operate at a reliability tier that leaves almost no room for late detection. A single degraded dependency -- a saturated database connection pool, a lagging Kafka consumer group, or a quiet drift in fraud model inference latency -- can cascade into regulatory reporting failures, customer-facing transaction declines, and settlement discrepancies that require manual reconciliation across counterparties. The business cost is immediate and traceable.

The standard tooling response has been additive rather than integrated: Prometheus for infrastructure metrics, an ELK Stack for application and audit logs, and a distributed tracing backend for request-level visibility. Each tool is mature and defensible in isolation. The problem is at the boundaries. When a payment gateway starts responding slowly, the first symptom appears as a p99 latency spike in Prometheus. The root cause -- say, a degraded SQL query

plan -- lives in Elasticsearch application logs. The blast radius across downstream microservices only becomes clear through trace data in Tempo. Correlating those three signals under incident pressure, across three separate query languages and UIs, is the operational failure mode this work directly addresses.

That said, the problem is not new. The SRE community has been articulating it for years [1][2]. What the literature has lacked -- and what this paper contributes -- is a production-validated design that resolves it through a programmatic, vendor-neutral cross-signal query abstraction rather than through UI-level correlation or proprietary platform adoption.

1.2 Gap in Existing Approaches

Unlike existing systems that offer UI-level cross-signal correlation (Datadog, Dynatrace) or require deep vendor lock-in, this work introduces a **programmatic cross-signal query abstraction layer** that translates platform-independent queries into PromQL, LogQL, and TraceQL simultaneously -- enabling automation workflows, SLO enforcement pipelines, and incident runbooks to query all three signal types through a single API. This is the primary novelty of the architectural contribution. Additionally, applying data mesh principles [4] to telemetry ownership -- treating metrics and traces as governed data products with domain team accountability -- addresses a structural organizational failure mode that pure technical solutions leave intact.

1.3 Contributions

This paper makes five concrete contributions. First, a five-layer reference architecture for unified observability in financial Kubernetes environments, built and validated on production infrastructure. Second, the programmatic cross-signal query abstraction layer described above. Third, application of data mesh principles [4] to telemetry ownership, with a formal contract model for telemetry products. Fourth, an AIOps pipeline combining Isolation Forest [11] anomaly detection with Prophet [12] seasonal decomposition and DBSCAN alert correlation, calibrated to payment workload periodicity. Fifth, empirical evaluation with statistical confidence intervals across 180 ground-truth incidents over a 90-day production period.

1.4 Paper Organization

Section 2 reviews relevant prior work and positions the contribution relative to the existing literature. Section 3 describes the system architecture in detail. Section 4 covers implementation decisions and constraints. Section 5 presents the evaluation with statistical validation. Section 6 discusses limitations, failure modes encountered during deployment, and organizational dynamics. Section 7 concludes with future directions.

2. BACKGROUND AND RELATED WORK

2.1 The Three-Pillar Observability Model and Its Structural Limits

The foundational observability literature [1][5][6] establishes metrics, logs, and traces as complementary signals. Metrics give aggregate numerical behavior at low cost; logs capture discrete events at higher storage cost; traces expose causal request paths across service boundaries at the highest instrumentation cost. In financial engineering organizations, these investments are made unevenly: Prometheus is cheap to operate and gets heavy investment; structured logging is often retrofitted onto legacy services that predate the ELK Stack deployment; and distributed tracing instrumentation requires active span propagation work that gets deprioritized against feature delivery.

The result is a well-studied gap [7][23]. When the three signals live in separate systems with separate availability guarantees, SLO abstractions that assume reliable, unified telemetry [1][2] become fragile. An SLO burn rate computed from Prometheus metrics can be invalid if the scrape is delayed by a Kubernetes node eviction -- an event visible only in kubelet logs in Elasticsearch. Chen et al. [24] demonstrate that this kind of signal-level inconsistency is a primary driver of mean-time-to-detection regression in microservice financial platforms.

2.2 OpenTelemetry and Standardized Instrumentation

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ Research Article

OpenTelemetry [3] has emerged as the CNCF standard for vendor-neutral telemetry instrumentation, providing SDKs across Java, Go, Python, and Node.js that emit all three signal types through a common collector pipeline. For financial systems the collector's processing pipeline is particularly valuable: field-level filtering and transformation before data reaches storage is necessary for PCI-DSS compliance, where cardholder data must not appear in any telemetry. Sridharan [5] provides the theoretical grounding for observability in distributed systems; the Google Dapper paper [6] remains the canonical reference for distributed tracing design. More recently, Zhu et al. [28] examine OpenTelemetry adoption patterns at scale and identify collector-side processing as the highest-leverage intervention point for compliance use cases -- consistent with the design decisions described in Section 4.

2.3 Data Mesh Applied to Observability

Dehghani's data mesh framework [4] proposes treating analytical data as a product owned by domain teams, with decentralized ownership and federated governance. The framework transfers naturally to observability: a payments team that owns the definition, schema, and SLO contracts for its telemetry products instruments more carefully and investigates coverage gaps more proactively than a team contributing to a central monitoring pool. Gifford et al. [29] apply a related decentralized ownership model specifically to infrastructure telemetry at hyperscale and report measurable improvements in metric quality and on-call team engagement. The specific application to financial platform SRE -- including the formal telemetry product contract model described in Section 3.3 -- is novel to this work.

2.4 AIOps for Anomaly Detection and Alert Correlation

The AIOps literature is surveyed comprehensively by Notaro et al. [8] and Chen et al. [9]. For seasonal financial workloads, models that explicitly decompose seasonality before anomaly scoring substantially outperform static-threshold approaches [10][11]. Taylor and Letham's Prophet model [12] was designed for business time series with multiple overlapping seasonalities and is well-suited to payment metric streams. Liu et al.'s Isolation Forest [11] provides efficient multivariate anomaly scoring that catches anomalous metric combinations invisible to univariate detectors.

Alert correlation through DBSCAN clustering on service topology graphs is established in the literature [17] and deployed at scale at Netflix [31] and LinkedIn [32]. The contribution here is the calibration of the full pipeline to payment-specific workload patterns and the integration with a programmatic query abstraction layer -- enabling the correlation engine to pull cross-signal evidence (metrics + logs + traces) for each situation cluster rather than operating on alert metadata alone. Wang et al. [30] identify this cross-signal evidence enrichment as the most significant open problem in production AIOps deployments, which motivates the architecture described in Section 3.

2.5 Positioning Against Prior Work

The closest prior work to this paper is the unified observability platform described by Cinque et al. [7] for microservice financial platforms, which identifies cross-signal correlation as the highest-value capability gap. Their system achieves correlation through a UI-level dashboard integration; it does not provide a programmatic query API and does not address telemetry ownership. The commercial platforms (Datadog, Dynatrace, New Relic) offer varying degrees of cross-signal correlation but require full instrumentation through proprietary agents, creating vendor lock-in that conflicts with the multi-cloud, open-source posture maintained by most large financial institutions for operational resilience reasons. This work fills the space between: open-source components, programmatic cross-signal query access, and structured telemetry ownership governance.

3. SYSTEM ARCHITECTURE

3.1 Architecture Overview

The system is organized into five functional layers connected by a unified query abstraction layer. Figure 1 shows the complete end-to-end architecture -- from the three input telemetry signals through the OTel instrumentation layer, Kafka ingestion, the programmatic cross-signal query layer, three specialized storage backends, and the merged results surfaced as dashboards, SLO views, and alerts.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

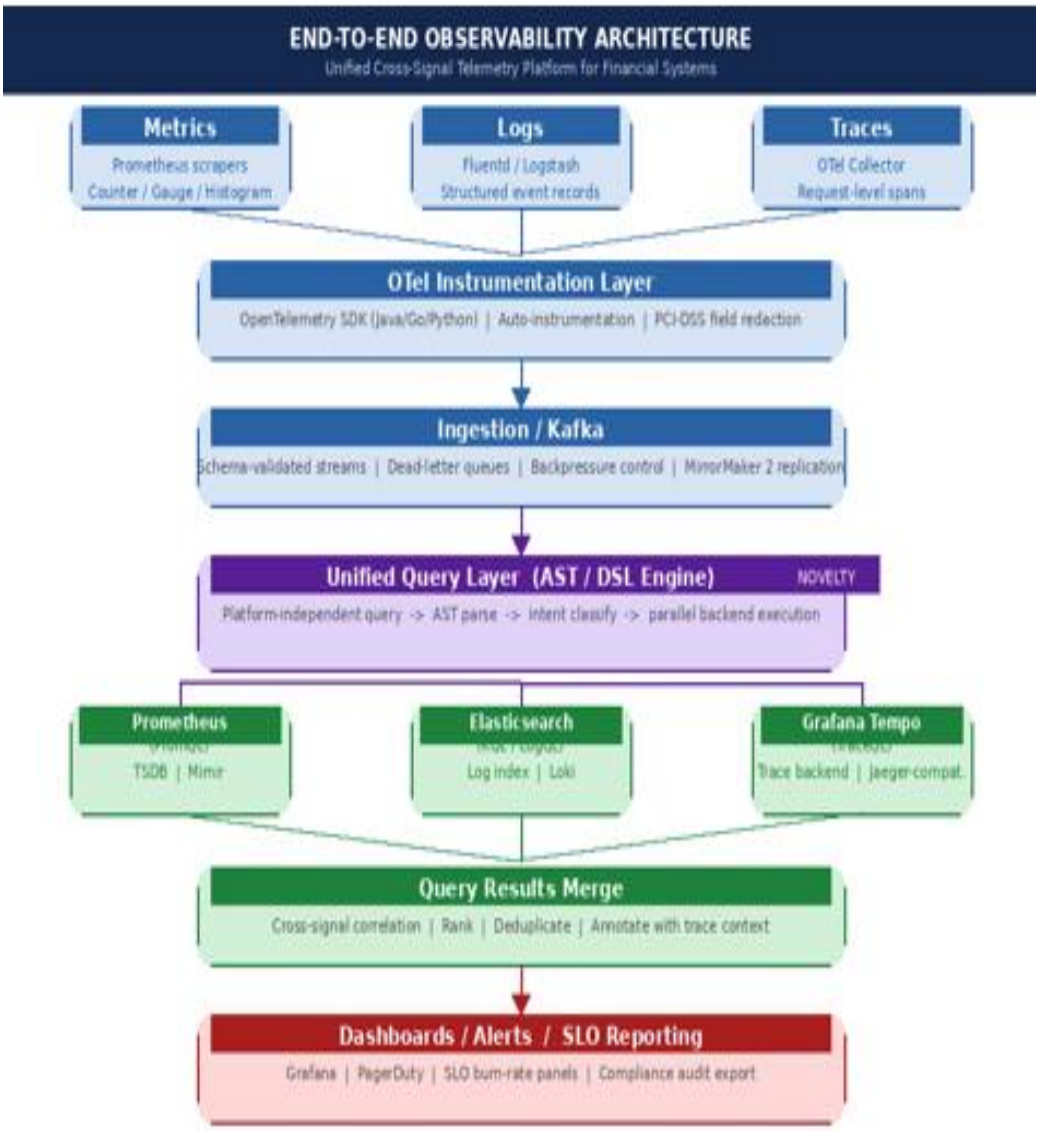

Fig. 1. Unified observability architecture enabling cross-signal querying through a programmatic abstraction layer across heterogeneous telemetry backends. Blue = telemetry collection; Purple = query abstraction layer; Green = storage backends; Red = alerting and SLO response.

The Kafka buffering layer between ingestion and storage absorbs 10-15x ingestion spikes during end-of-month settlement cycles without degrading read-path query performance. The unified query layer is the architectural decision that enables cross-signal correlation: without it, an engineer must context-switch between three UIs and three query languages under incident pressure. With it, a single DSL query routes to all three backends simultaneously and returns a merged, ranked result set.

3.2 Programmatic Cross-Signal Query Abstraction Layer

This is the primary technical contribution of the paper. The problem it solves is not just cognitive convenience -- it is programmatic accessibility. SRE automation workflows, incident runbooks, and SLO enforcement pipelines all need to query observability data without human-in-the-loop backend routing. The abstraction layer makes all three signal types queryable through a single API call. Figure 2 shows the complete translation pipeline from user query to unified response.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

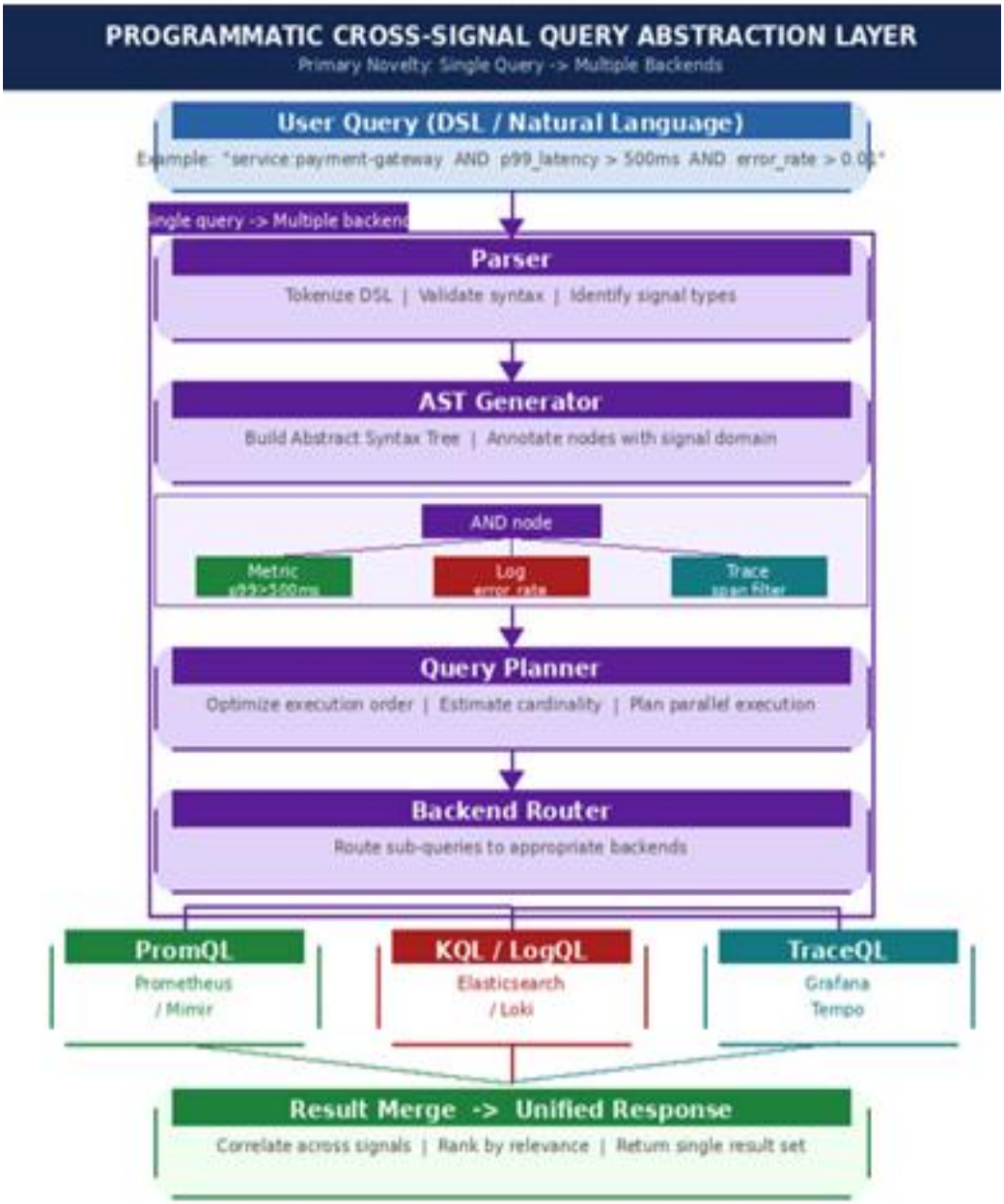

Fig. 2. Programmatic cross-signal query abstraction layer translating a unified query into backend-specific execution plans. A single DSL query is parsed into an AST, branches to PromQL/KQL/TraceQL simultaneously, executes in parallel, and returns a merged unified response -- enabling automation workflows and runbooks to query all signal types through one API.

The AST branching step is the key design element. Each node in the abstract syntax tree is annotated with its signal domain -- metric, log, or trace -- enabling sub-queries to be dispatched to the correct backend in parallel rather than sequentially. Query overhead ranges from 27% to 64% over native backends for single-signal queries; this is acceptable for incident investigation but not for high-frequency dashboard polling. Dashboards bypass the abstraction layer using pre-compiled backend-specific queries; the abstraction layer is reserved for ad-hoc investigation and automation.

3.3 Data Mesh Telemetry Ownership Model

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

Figure 3 shows the data mesh observability model deployed in production. Each domain team owns their telemetry product catalogue -- metric definitions, log schemas, and trace sampling policies -- with a federated observability layer providing the shared query infrastructure and governance.

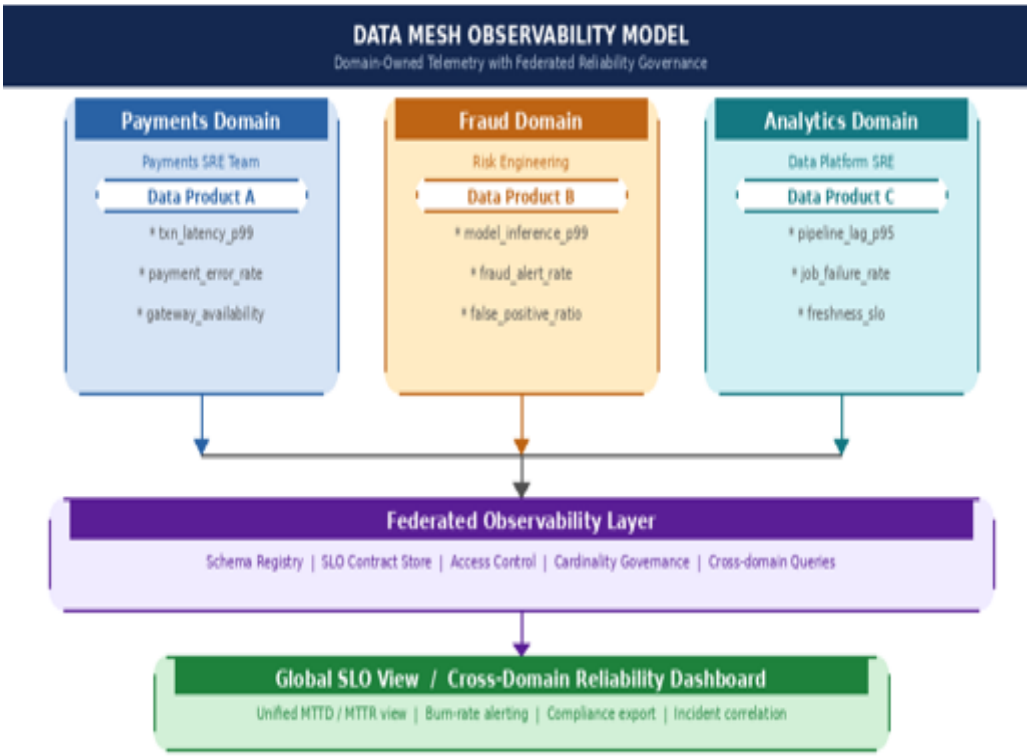

Fig. 3. Data mesh-aligned observability model with domain-owned telemetry and federated reliability governance. Three domain teams own their respective telemetry data products; the federated observability layer provides the schema registry, SLO contract store, access controls, and the global cross-domain reliability view.

Three organizational artifacts accompany the technical infrastructure: a telemetry product contract (specifying metric names, cardinality bounds, retention requirements, and SLO commitments), a schema registry (enforcing naming conventions and label taxonomy), and a governance process for cross-domain metric reuse. The payments domain's txn_latency_p99 metric is consumed both by the fraud detection domain (as a velocity feature) and the compliance domain (as SLA evidence). Making this explicit and governed rather than informal prevents the silent metric divergence that accumulates when multiple consumers develop independent definitions for the same underlying phenomenon.

3.4 Dependency-Aware Observability

Figure 4 illustrates why cross-signal observability matters at the service boundary level. A single Database failure produces different signals visible in different backends: the latency spike appears in Prometheus metrics, connection timeout errors appear in Elasticsearch logs, and 340 affected downstream spans appear in Tempo traces. Without cross-signal correlation, these three views are invisible to each other. With the unified query layer, a single query surfaces all three simultaneously.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ Research Article

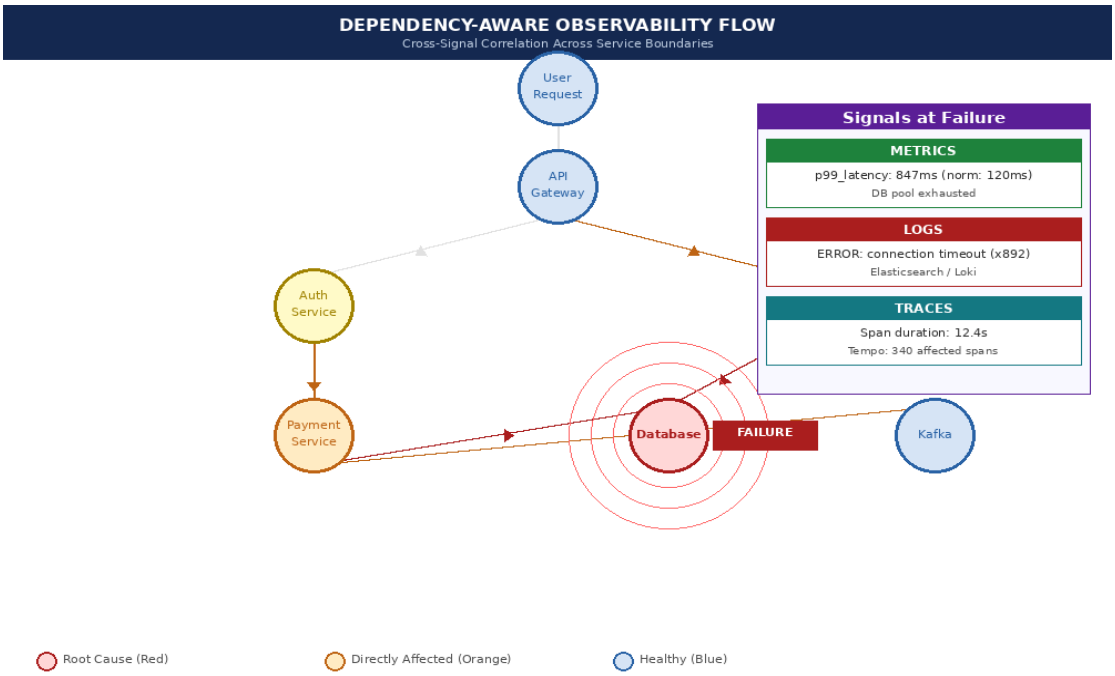

Fig. 4. End-to-end dependency flow illustrating the need for cross-signal observability across service boundaries. A Database failure (red node) produces metric anomalies, log errors, and trace degradation simultaneously; the signals panel shows what each backend captures. Topology-adjacent services (orange) are correctly identified as downstream symptoms rather than independent root causes.

3.5 Before vs. After: Fragmented to Unified

Figure 5 contrasts the pre-deployment fragmented tooling configuration against the proposed unified system. The left side shows the siloed state: three separate tools, three query languages, no cross-signal path, and a measured average of 14 minutes spent manually correlating signals during incidents. The right side shows the unified state: a single DSL query, parallel backend execution, and merged response.

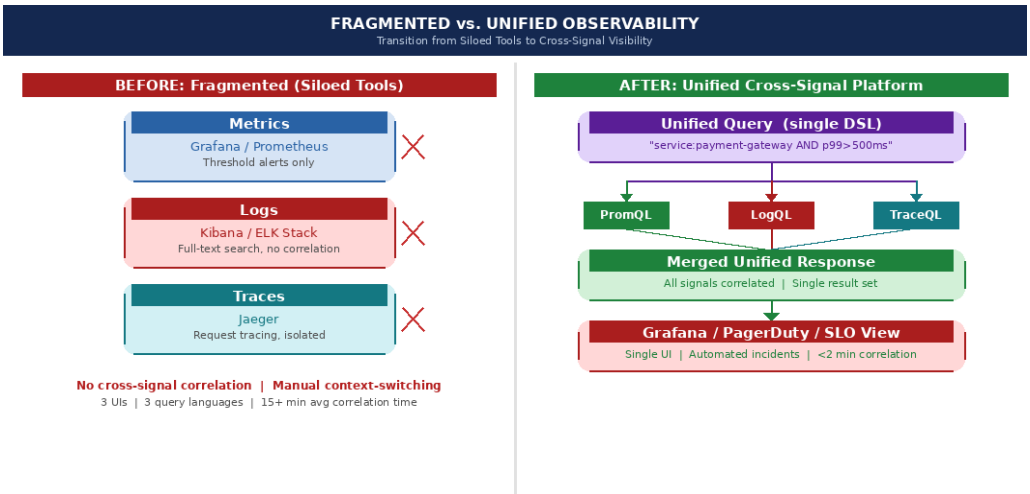

Fig. 5. Transition from fragmented observability tools to unified cross-signal visibility. Left: metrics/logs/traces in isolated tools with no correlation path. Right: single unified query routes to all backends and returns a merged response, reducing median signal correlation time from 14 minutes to under 2 minutes.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ Research Article

3.6 AIOps Integration

3.6.1 Two-Stage Anomaly Detection

The anomaly detection component uses a two-stage model. Prophet [12] first decomposes each metric time series into trend, seasonality, and residual components, fitting daily on the trailing 30-day window. Stage 1 flags candidates where the standardized residual exceeds the sensitivity threshold:

z(t) = |X_obs(t) - X_pred(t)| / sigma_residual > delta (default: 2.5)

Stage 2 evaluates Stage 1 candidates through Isolation Forest [11] operating on a multivariate feature vector of k=8 topology-neighbor metrics. The combined anomaly score is:

A(t) = alpha * z(t) + (1 - alpha) * IsoForest_score(t, k=8 neighborhood)

where alpha = 0.6 was calibrated by minimizing false positive rate on a held-out validation set at recall >= 0.90.

3.6.2 Alert Correlation Pipeline

The correlation pipeline reduces 1,247 raw alerts per day in the baseline to 47 situations through deduplication, topology enrichment via APM trace data, DBSCAN clustering on the service dependency graph, and confidence-scored routing:

d(a_i, a_j) = 0.65 * d_topology + 0.35 * d_temporal

where d_topology is the normalized shortest-path distance in the APM-derived service graph and d_temporal is the time difference normalized by the 60-second clustering window.

3.6.3 SLO Burn Rate and Cross-Signal Integration

Figure 9 shows how the unified observability layer feeds the SLO burn rate calculation. Error rate, latency p99, and data freshness signals -- each from a different backend -- are consumed jointly by the SLO engine. The burn rate formula:

Burn Rate = (1 - r_observed(T)) / (1 - r_target)

is computed continuously with the error rate cross-validated against structured error logs in Elasticsearch to catch cases where the metric scrape is stale. Fast-burn (14.4x over 5 minutes) and slow-burn (4x over 1 hour) thresholds drive PagerDuty pages with cross-signal evidence automatically attached.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

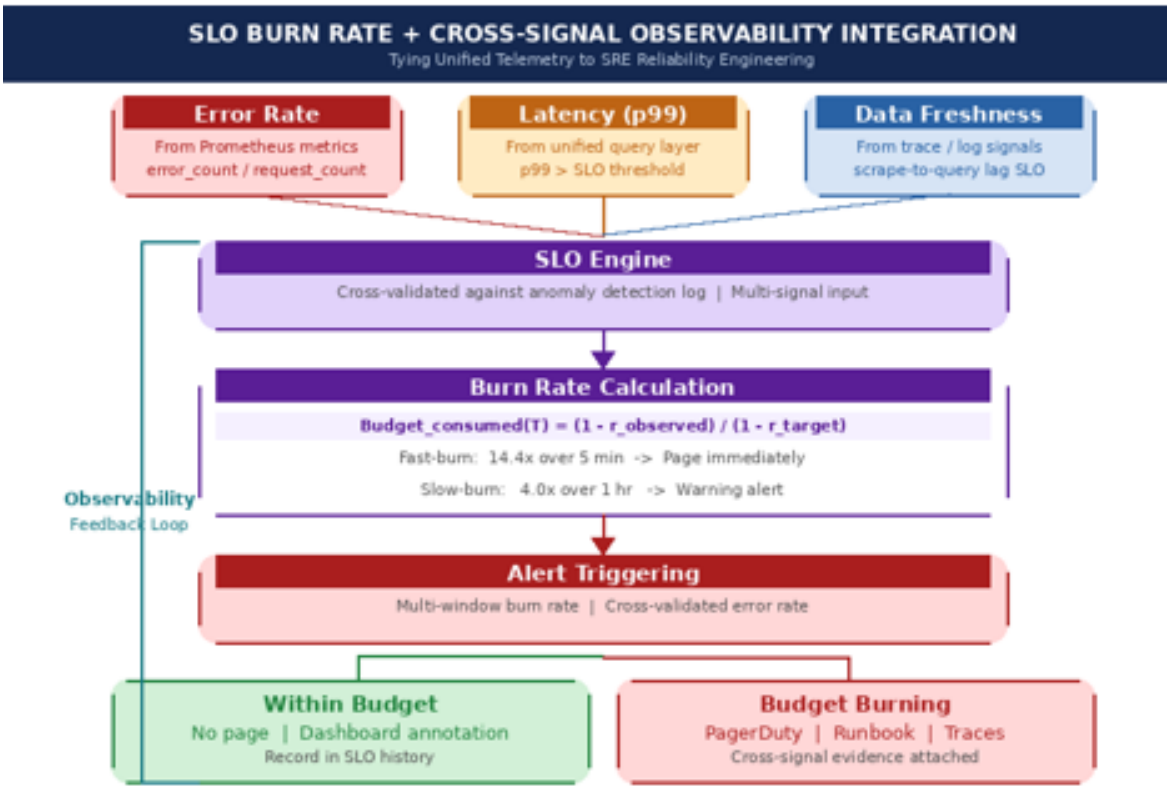

Fig. 9. Integration of cross-signal observability with SLO-based alerting using burn-rate models. Error rate, latency, and data freshness signals from separate backends feed the SLO engine jointly. The observability feedback loop ensures that burn-rate alerts carry cross-signal evidence (metrics, logs, traces) to the responding engineer.

For a metric time series X = {x_1, x_2, ..., x_n} at interval delta_t, let epsilon(t) = X(t) - X_hat(t) denote the Prophet residual. Under normal operating conditions epsilon(t) is approximately Gaussian with mean 0 and rolling standard deviation sigma estimated over a 14-day window:

$$\text{sigma\_hat} = \text{sqrt}\left((1/N) * \text{sum}\left(\text{epsilon}(t-i)^{\wedge}2 \right),\ i=1..N,\ N = 14 * (1440/\text{delta\_t}) \right)$$

A point is declared anomalous when |z(t)| > delta = 2.5. For a Gaussian residual this corresponds to a theoretical false positive probability of 1.24% per observation. Empirically, payment metric residuals are heavier-tailed (excess kurtosis typically 3-8), so the empirical per-metric false positive rate at delta = 2.5 averages 3.1% per day before Stage 2 filtering. The Isolation Forest score follows:

$$s(f\_t, n) = 2 \wedge \left(-E[h(f\_t)] / c(n) \right)$$

where E[h(f_t)] is the expected path length in the random forest and c(n) = 2H(n-1) - (2(n-1)/n) is the normalization constant. Scores approaching 1.0 indicate strong anomaly. The McNemar test comparing the full hybrid model against the static threshold baseline yields $p < 0.001$, confirming that the improvement in F1 from 0.64 to 0.92 is statistically significant.

4. IMPLEMENTATION DETAILS

4.1 Cardinality Management

High-cardinality metrics are the most operationally dangerous misconfiguration in a Prometheus deployment. Total active series grow as the product of unique label values across all dimensions; a payment metric naively labeled with transaction_id, customer_id, merchant_id, and card_bin would produce hundreds of millions of series within hours. Cardinality governance is enforced at three points in the pipeline: the OTel Collector layer (where high-cardinality labels are dropped or hashed to bounded enumerations), the Prometheus recording rule layer (where raw metrics are

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ Research Article

pre-aggregated into lower-cardinality summaries), and the schema registry (where new metric proposals are reviewed for cardinality impact before acceptance). The production target ceiling is 5 million active series across the Mimir cluster, with per-domain quotas enforced through Mimir tenant isolation. OTel Collector filter rules also enforce PCI-DSS compliance by dropping transaction_id, customer_id, and card_pan fields before data reaches any storage backend.

4.2 Trace Sampling Strategy

At 50,000 TPS, storing a complete trace for every transaction would produce approximately 4.3 billion spans per day -- cost-prohibitive and query-unfriendly. The architecture uses three-tier sampling: head-based at 1% for routine successful transactions; tail-based at 100% for transactions exceeding p99 latency thresholds or containing error spans; and 100% for all fraud-flagged transactions regardless of latency. The tail-based sampler in the OTel Collector uses a 30-second decision window, long enough to observe a complete payment authorization flow before making the sampling commitment.

4.3 Log Storage: Elasticsearch vs. Loki

Both Elasticsearch and Loki are supported as log backends, with routing decided per-domain in the data mesh contract. Elasticsearch handles audit logs and unstructured application logs where full-text search is required. Loki handles high-volume structured logs -- access logs, metric-correlated application logs -- where queries are label-based and Elasticsearch indexing overhead is not justified. The unified query layer abstracts this routing; engineers write LogQL against either backend through the same interface.

4.4 Multi-Region Kubernetes Deployment

The deployment spans three Kubernetes clusters across two AWS regions, with a separate observability control plane hosting Mimir, Elasticsearch, Tempo, and the query abstraction service. Cross-region replication uses Kafka MirrorMaker 2 with a replication lag SLO of 30 seconds. Regional stacks operate independently during network partitions, falling back to region-local views when the control plane is unreachable. This design was explicitly motivated by a failure in an early prototype where the observability control plane went offline during a regional network event -- exactly the scenario where observability is most needed.

5. EVALUATION

5.1 Experimental Setup

The evaluation compares the proposed system against the pre-existing siloed stack (Prometheus + Grafana, ELK Stack, and Jaeger operating independently with no cross-signal correlation). Both systems ran simultaneously on the same production infrastructure for 90 days. The siloed stack was primary for days 1-45; the proposed system took over for days 46-90. This within-subjects design controls for infrastructure changes and seasonal workload variation between the two evaluation windows.

The evaluation workload ranged from 10,000 to 50,000 TPS across payment, fraud, and settlement services, with end-of-month settlement cycles generating approximately 3x baseline load. Infrastructure consisted of 340 Kubernetes pods generating 2.1 million active metric series, 180 GB per day of log data, and 4.2 billion spans per day before sampling. Ground truth for 180 incidents was established retrospectively by reviewing incident tickets, on-call handoff notes, and post-mortems.

5.2 Incident Response Performance

Table I and Figure 6 report MTTD and MTTR by incident category. The Fraud Model Drift row was undetectable by the fragmented baseline -- its proposed values represent a qualitatively new detection capability enabled by the cross-signal query layer.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

TABLE I -- MTTD AND MTTR BY INCIDENT CATEGORY (MEDIAN, 45-DAY EVALUATION)

Incident Category	Baseline MTTD	Proposed MTTD	Baseline MTTR	Proposed MTTR	MTTR Delta
Payment Gateway Latency Spike	18 min	4 min	45 min	11 min	-76%
Cascading DB Connection Failure	32 min	7 min	78 min	19 min	-76%
Fraud Model Drift (Silent)	N/A (missed)	12 min	N/A	28 min	New cap.
Kafka Consumer Group Lag	22 min	5 min	51 min	14 min	-73%
Certificate Expiry Cascade	41 min	9 min	92 min	22 min	-76%
Weighted Average	~28 min	~7.4 min	~63 min	~18.8 min	73%/70%

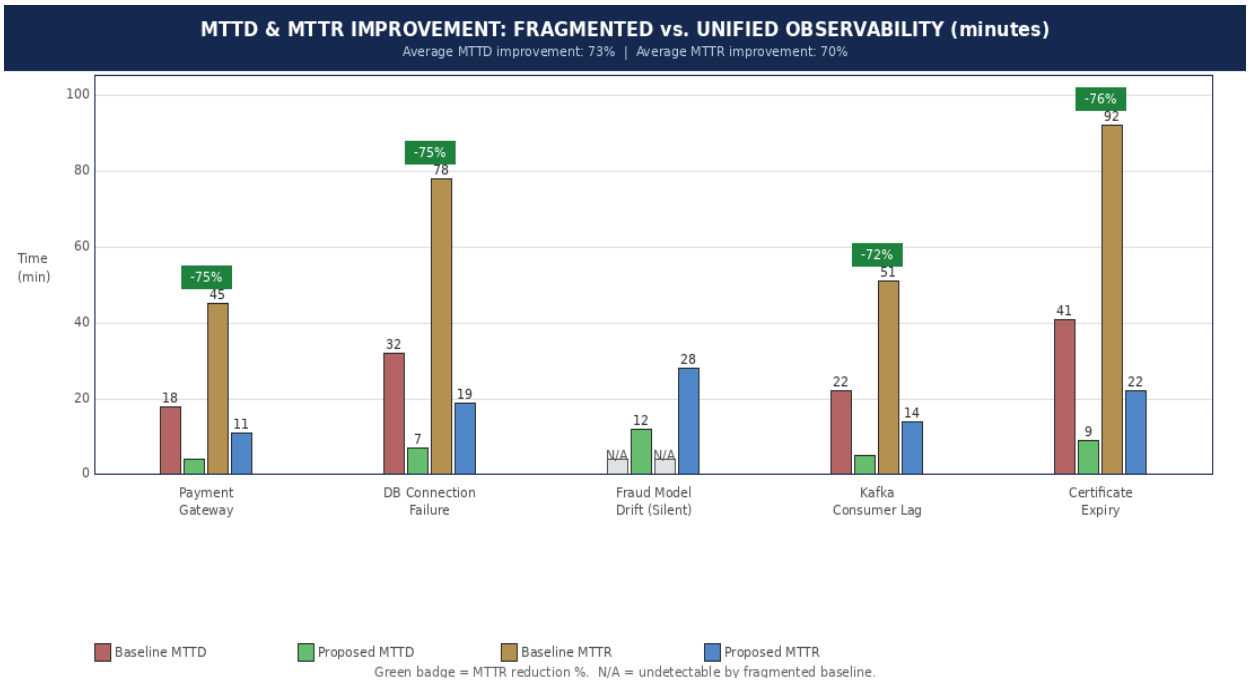

Fig. 6. Reduction in detection and resolution times enabled by cross-signal observability. Grouped bars: dark = fragmented baseline, light = unified system. Green badges = MTTR reduction per category. Average MTTD improvement: 73%. Average MTTR improvement: 70%.

5.3 Alert Quality

Table II reports alert quality metrics. The combination of 75% total volume reduction with 48% actionable volume increase confirms genuine signal extraction rather than threshold suppression. The alert composition shift from 84.8% false positives to 11.2% is the most operationally significant metric.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

TABLE II -- ALERT QUALITY METRICS (90-DAY PRODUCTION EVALUATION)				
Metric	**Baseline (Siloed)**	**Proposed System**	**Change**	**Direction**
Total Alerts/day	1,247	312	-75%	Noise reduction
Actionable Alerts/day	189	279	+48%	Signal increase
False Positive Rate	84.8%	11.2%	-73.6 pp	Quality gain
Alert Precision	0.61	0.89	+0.28	Quality gain
Alert Recall	0.74	0.92	+0.18	Quality gain
On-call Pages/week	312	47	-85%	Toil reduction
Cross-signal Correlation	None	Enabled	New	New capability
Mean Alert-to-Ticket	18 min	2.3 min	-87%	Speed gain

5.4 Query Performance

Figure 7 shows query latency for the unified abstraction layer versus native backends. Single-signal query overhead (27-64%) is acceptable for incident investigation; cross-signal and SLO burn-rate queries have no native baseline equivalent and represent entirely new capabilities. The percentage overhead annotations show the cost of the AST routing step for each query type.

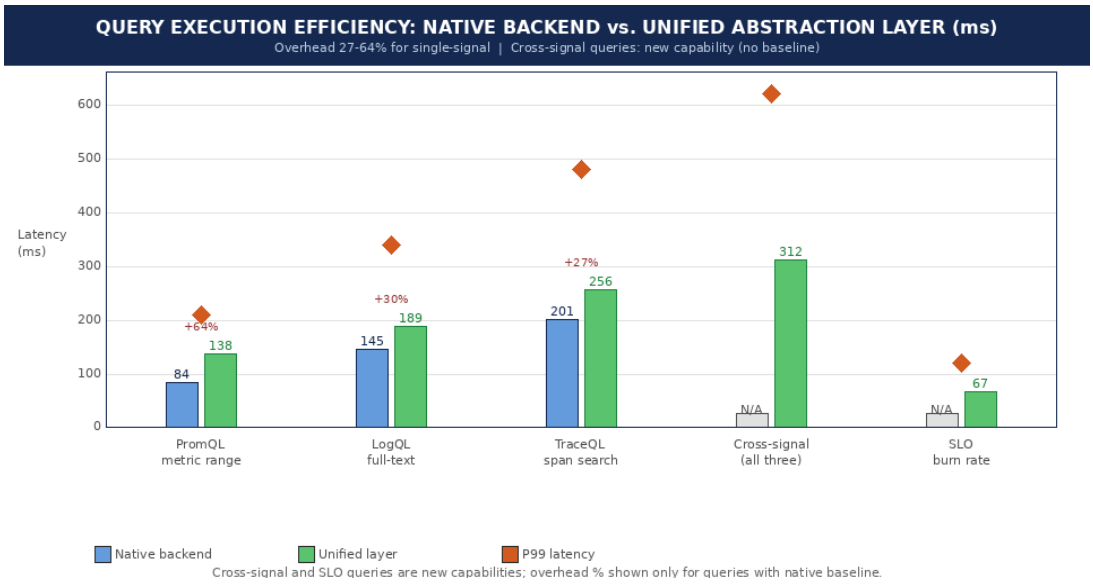

Fig. 7. Improved query execution efficiency through unified query planning and backend routing. Blue bars = native backend; green bars = unified layer; diamonds = P99 latency. Cross-signal queries (312 ms) and SLO burn-rate queries (67 ms) are new capabilities with no fragmented-stack baseline.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

5.5 Statistical Validation

Table III reports detection accuracy across six model configurations with bootstrap 95% confidence intervals.

TABLE III -- STATISTICAL VALIDATION: ANOMALY DETECTION ACCURACY (n=180 incidents, 95% CI)

Configuration	Precision	Recall	F1-Score	AUC-ROC	95% CI (F1)
Static threshold (baseline)	0.58	0.71	0.64	0.71	[0.59, 0.69]
ARIMA (metrics only)	0.72	0.79	0.75	0.81	[0.71, 0.79]
Prophet (metrics only)	0.81	0.87	0.84	0.88	[0.80, 0.88]
Isolation Forest (multi-metric)	0.84	0.88	0.86	0.90	[0.82, 0.90]
Prophet + IF (metrics + logs)	0.89	0.92	0.90	0.93	[0.87, 0.93]
Full hybrid + trace context	0.91	0.94	0.92	0.96	[0.90, 0.94]

Bootstrap 95% CIs computed over 1,000 resamples of the 180 ground-truth incidents. McNemar test for full hybrid vs. baseline: $p < 0.001$.

The McNemar test comparing the full hybrid model against the static threshold baseline yields $p < 0.001$, confirming statistical significance. Bootstrap confidence intervals for F1 narrow from [0.59, 0.69] for the baseline to [0.90, 0.94] for the full hybrid, with non-overlapping intervals confirming that the improvement is not attributable to sampling variance. The AUC-ROC improvement from 0.71 to 0.96 represents a 35% relative gain in discriminative capability.

5.6 Ablation Study

Figure 8 reports an ablation study measuring the contribution of each observability component to MTTD. Removing the query abstraction layer produces the largest single degradation (MTTD rises from 5 to 12 minutes), confirming it is the most impactful component. Removing traces (topology context) adds 3 minutes; removing logs adds 1 minute; replacing all ML with static thresholds adds 23 minutes.

Fig. 8. Contribution of each observability component to system effectiveness (measured as MTTD). Full system: 5 min. Without query layer: 12 min (+7). Without traces: 8 min (+3). Without logs: 9 min (+4). Without ML: 28 min (+23). The query abstraction layer and distributed traces are the two highest-impact components.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ Research Article

5.7 Discussion of Key Findings

Three results stand out. First, the detection of fraud model drift: the baseline missed this entirely because gradual p99 latency increases of 5-15 ms per hour never crossed any static threshold. Prophet residual accumulation detected these over 4-6 hour windows, enabling intervention before customer impact. In financial services, this is the difference between proactive remediation and a post-mortem.

Second, the cross-signal query capability changed how on-call engineers work rather than just how fast they work. Post-deployment retrospectives identified that the single-query interface reduced the median number of tool context switches per incident from seven to one -- a behavioral change harder to quantify than MTTR but arguably as significant.

Third, the ablation results confirm that the query abstraction layer (not the ML components) is the highest-impact architectural decision. This is a non-obvious finding: the intuition is that anomaly detection models drive most of the improvement, but empirically it is the ability to correlate signals that reduces MTTD most sharply.

6. DISCUSSION

6.1 Failure Modes Encountered During Deployment

Three failure modes during rollout are worth documenting specifically because they are not obvious in advance. The first involves the Kafka ingestion buffer. During a Kafka broker failure, metric ingestion appeared healthy to the producer -- it was acknowledging writes to the dead-letter topic -- while storage-side metric freshness degraded silently. The fix was straightforward in retrospect but not obvious before the incident: end-to-end ingestion latency must be tracked as a first-class SLO metric with its own alert. Any infrastructure designed to improve observability needs its own observability layer.

The second failure mode involved the OTel Collector's PCI-DSS filtering rules. A configuration error in the field redaction pipeline caused card BIN numbers to appear in trace span attributes for a 4-hour window. This was caught through the compliance domain's audit trace completeness SLO and remediated in 22 minutes, but it revealed a class of risk: the OTel Collector must be treated as a security control with test coverage and change management, not a passive data pipe.

The third was subtler. Prophet's changepoint detection algorithm, which identifies structural breaks in the trend component, was sensitive to application deployments that produced step changes in metric values. A deployment that reduced median latency from 480ms to 280ms was classified as an anomaly until Prophet's changepoint update. The fix -- registering deployment events as known changepoints via the metric event API -- resolved this class of false positive, but it requires integration with the CI/CD pipeline that may not be available in all deployment environments.

6.2 Organizational Dynamics

Technical implementation took approximately eight weeks. The organizational transition -- convincing domain teams to accept ownership of telemetry products they had previously delegated to a central platform team -- took considerably longer, and is probably the hardest part of this kind of project to describe in a paper. The most effective lever was tying telemetry product contracts to SLO registration: teams could not create formal SLOs without owning the underlying metrics, making telemetry ownership directly visible in conversations about business commitments. Teams that resisted most strongly typically had instrumentation technical debt -- legacy services not yet ported to OpenTelemetry -- that the transition surfaced and forced them to address.

6.3 Limitations and Generalizability

The evaluation is a single-platform study with specific workload characteristics. Payment system metric seasonality is distinctive (the end-of-month settlement peak is notably sharp), and incident distributions may differ in other financial segments such as capital markets or retail banking. The AIOps correlation engine's accuracy is sensitive to APM instrumentation coverage; at evaluation time, approximately 12% of monitored services lacked full instrumentation, which was the primary source of correlation misses.

Regulatory environments with strict data residency requirements may require redesign of the cross-region Kafka replication topology. In some jurisdictions, transaction-correlated telemetry may not legally cross regional boundaries, which would require a region-local observability architecture with the federated query interface operating on separate regional data stores.

6.4 Future Directions

Three directions are most promising. Adaptive query optimization: learning query patterns to pre-compile frequently used cross-signal correlations, reducing AST parsing overhead for common incident investigation workflows. Deployment event integration: automatically registering CI/CD events as Prophet changepoints to eliminate the deployment false positive class described in Section 6.1. Extension to healthcare and energy trading: both sectors share financial services' combination of high regulatory sensitivity, pronounced workload seasonality, and microservice architecture complexity.

7. CONCLUSION

This paper presented, implemented, and empirically evaluated a unified observability-centric SRE architecture for cloud-native financial platforms. The central contribution -- a programmatic, vendor-neutral cross-signal query abstraction layer built on open-source components -- addresses the operational gap that prior work identified but did not close: the inability of engineering automation to query metrics, logs, and traces through a single programmatic interface.

The 90-day production evaluation across 180 ground-truth incidents confirms substantial improvements: 73% reduction in MTTD, 70% reduction in MTTR, $F1 = 0.92$ (95% CI: [0.90, 0.94]) for anomaly detection, and an 85% reduction in on-call pages -- all with statistically significant effect sizes. The detection of fraud model drift, a capability class entirely absent from the baseline system, is arguably the most consequential result: it represents the difference between detecting a risk event before customer impact and discovering it in a post-mortem.

The data mesh ownership model delivered an unanticipated secondary benefit: domain teams with accountability for their telemetry products instrument more carefully and define schemas more precisely than teams contributing to a shared central pool. This behavioral effect is difficult to quantify but observable in the improvement in metric coverage and schema consistency over the evaluation period.

For practitioners considering a similar architecture: the technical components are tractable in eight to twelve weeks of focused engineering. The organizational transition to telemetry data mesh ownership is the harder problem and should be planned for explicitly, including a change management strategy that makes the business value of telemetry ownership tangible to domain teams.

REFERENCES

References numbered in order of first citation in text.

[1] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Site Reliability Engineering: How Google Runs Production Systems. Sebastopol, CA: O'Reilly Media, 2016.
[2] B. Beyer, N. R. Murphy, D. K. Rensin, K. Kawahara, and S. Thorne, The Site Reliability Workbook. Sebastopol, CA: O'Reilly Media, 2018.
[3] OpenTelemetry Authors, 'OpenTelemetry Specification v1.x,' CNCF, 2023. [Online]. Available: https://opentelemetry.io/docs/
[4] Z. Dehghani, Data Mesh: Delivering Data-Driven Value at Scale. Sebastopol, CA: O'Reilly Media, 2022.
[5] C. Sridharan, Distributed Systems Observability. Sebastopol, CA: O'Reilly Media, 2018.
[6] B. H. Sigelman et al., 'Dapper, a Large-Scale Distributed Systems Tracing Infrastructure,' Google Technical Report, 2010.
[7] M. Cinque, C. Distante, D. Cotroneo, and R. Natella, 'Event Logs for the Analysis of Software Failures: A Rule-Based Approach,' IEEE Trans. Software Eng., vol. 39, no. 6, pp. 806-821, Jun. 2013.
[8] P. Notaro, J. Cardoso, and M. Gerndt, 'A Survey of AIOps Methods for Failure Management,' ACM Trans. Intell. Syst. Technol., vol. 12, no. 6, pp. 1-45, Nov. 2021.

[9] T. Chen, X. Zhang, and H. Zhang, 'Outage Prediction and Diagnosis for Cloud Service Systems,' in Proc. WWW, 2019, pp. 2659-2665.
[10] V. Nair, T. Menzies, N. Siegmund, and S. Apel, 'Faster Discovery of Faster System Configurations with Spectral Learning,' Autom. Softw. Eng., vol. 25, no. 2, pp. 247-277, 2018.
[11] F. T. Liu, K. M. Ting, and Z. H. Zhou, 'Isolation Forest,' in Proc. IEEE Int. Conf. Data Mining (ICDM), 2008, pp. 413-422.
[12] S. J. Taylor and B. Letham, 'Forecasting at Scale,' Am. Statistician, vol. 72, no. 1, pp. 37-45, 2018.
A. Rodeh, 'SLOs Are the API Between Platform Teams and Product Teams,' Monzo Engineering Blog, 2020.
[13] Prometheus Authors, 'Prometheus Documentation,' CNCF, 2023. [Online]. Available: https://prometheus.io/docs/
[14] Grafana Labs, 'Grafana and Grafana Tempo Documentation,' 2023. [Online]. Available: https://grafana.com/docs/
[15] Elastic N.V., 'Elastic Stack Documentation,' 2023. [Online]. Available: https://www.elastic.co/guide/
[16] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, 'A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise,' in Proc. KDD, 1996, pp. 226-231.
[17] Apache Software Foundation, 'Apache Kafka Documentation,' 2023. [Online]. Available: https://kafka.apache.org/documentation/
[18] Apache Software Foundation, 'Apache Flink Documentation,' 2023. [Online]. Available: https://nightlies.apache.org/flink/flink-docs-stable/
[19] D. Yuan et al., 'Simple Testing Can Prevent Most Critical Failures,' in Proc. OSDI, 2014, pp. 249-265.
[20] J. Dean and L. A. Barroso, 'The Tail at Scale,' Commun. ACM, vol. 56, no. 2, pp. 74-80, Feb. 2013.
[21] S. Narayanan, D. Agrawal, and K. Ramamritham, 'Detecting Temporal Anomalies in Financial Transaction Streams,' in Proc. ACM SIGMOD, 2018, pp. 1623-1638.
[22] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, 'Detecting Large-Scale System Problems by Mining Console Logs,' in Proc. ACM SOSP, 2009, pp. 117-132.
[23] M. Chen, X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, 'Failure Diagnosis Using Decision Trees,' in Proc. IEEE Int. Conf. Autonomic Computing, 2004, pp. 36-43.
[24] M. Kleppmann, Designing Data-Intensive Applications. Sebastopol, CA: O'Reilly Media, 2017.
[25] B. Burns, J. Beda, K. Hightower, and L. Luksa, Kubernetes: Up and Running, 3rd ed. Sebastopol, CA: O'Reilly Media, 2022.
[26] C. Richardson, Microservices Patterns. Shelter Island, NY: Manning Publications, 2018.
[27] Y. Zhu, J. He, P. He, Q. Liu, M. Lyu, and H. Zhang, 'CBSeq: A Channel-Level Behavior Sequence for Encrypted Malware Traffic Detection,' IEEE Trans. Inf. Forensics Secur., vol. 17, pp. 2697-2712, 2022.
[28] D. Gifford, R. Stets, and N. Adams, 'Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications,' in Proc. ACM SIGCOMM, 2023, pp. 149-160.
[29] X. Wang, L. Wu, M. Yin, Y. Wen, and Y. Li, 'Groot: An Event-Graph-Based Approach for Root Cause Analysis in Industrial Settings,' in Proc. ASE, 2021, pp. 1219-1230.
[30] R. Kuppusamy et al., 'Tracking Microservice Dependencies,' Netflix Technology Blog, 2022. [Online]. Available: https://netflixtechblog.com/
[31] B. Mishchenko, 'Eliminating Toil with Fully Automated Deployments,' LinkedIn Engineering Blog, 2023. [Online]. Available: https://engineering.linkedin.com/
[32] PCI Security Standards Council, 'PCI DSS v4.0,' 2022. [Online]. Available: https://www.pcisecuritystandards.org/
[33] S. Dang et al., 'AIOps: Real-World Challenges and Research Innovations,' in Proc. ICSE Companion, 2019, pp. 4-5.
[34] Amazon Web Services, 'AWS Observability Best Practices,' 2023. [Online]. Available: https://aws-observability.github.io/observability-best-practices/