AML Observability: Rethinking Transaction Monitoring as a Debuggable Compliance System

An Architectural Position Paper with a Larger Synthetic Evaluation

Jürgen Schiller García
FinCrime Watchdog
ORCID: 0009-0001-5370-4972

April 2026 · v1.8 · Submitted to aixiv.science

Abstract

AML transaction monitoring systems are widely deployed across financial institutions, yet they often remain difficult to reconstruct, explain, and evaluate in production. This paper argues that a key limitation is not only insufficient detection logic, but insufficient system observability. It defines *AML Observability* as the capability to reconstruct detection-relevant decisions across the full processing lifecycle and reframes recurring AML control breakdowns as *diagnosability gaps* rather than isolated detection misses.

The paper makes five linked contributions. First, it proposes a five-layer AML observability architecture and the *Transformation Spiral* as a sequencing model linking data governance, system observability, AI observability, and AI enablement. Second, it interprets public enforcement cases through an OBASHI-informed lens and derives a compact anti-pattern library, including *Upstream Blindspot*, *Silent Transformation Drop*, *Threshold Suppression*, and *Broken Feedback Loop*. Third, it formalizes a minimal event-centric proof of concept in Python built around an `AMLTrace`/`AMLTraceEvent` model defined as an algebraic 7-tuple with explicit query semantics for `why_flagged`, `why_not_flagged`, and `what_changed` (with pseudocode and $O(|E|)$ complexity analysis), privacy-aware trace retention, and a design for handling stateful temporal dependencies. Fourth, it extends the same trace schema toward AI governance by embedding model-specific artifacts—including SHAP-based feature attribution, confidence distributions, GNN embedding vectors, and KL-divergence drift indicators—into the compliance trace as native Ω payloads queryable through the same diagnostic semantics. Fifth, it adds a production-oriented architecture sketch and a larger synthetic comparison between a monitoring-only baseline and an observable system.

The larger evaluation processes 1,000 synthetic transactions, including 300 injected faults distributed across six fault families spanning the five-layer stack. Across the fault cases, the observable system improves mean diagnosis completeness from 0.25 to 1.00, failure-layer attribution accuracy from 0.17 to 1.00, and reduces explanation steps from 4.0 to 1.0 relative to the baseline. A sizing model based on the current PoC event structure further indicates an average raw telemetry footprint of roughly 3.38 GB per one million transactions, falling to about 2.05 GB under the selective retention policy. These results remain synthetic and do not constitute production validation, but they provide stronger empirical and engineering support for the claim that observability improves diagnosability in AML systems. The paper concludes that observability is neither a substitute for governance nor an alternative to AI, but the architectural condition under which governance becomes evidence-based and AI becomes more governable.

Keywords: AML, Transaction Monitoring, Observability, Compliance Architecture, AI Governance, Data Lineage, OpenTelemetry, OBASHI, RegTech, AMLR, AMLA

---

1 Introduction

AML monitoring is typically framed as a detection problem: build better rules, train sharper models, reduce false positives. However, many documented failures suggest a deeper issue: institutions often cannot reconstruct how their systems actually behave. They know *what* was flagged, but not *why*—and, more critically, they cannot systematically determine what was *missed*.

This paper argues that AML weaknesses are frequently better understood as observability deficits rather than detection deficits. The distinction matters: detection can be improved incrementally through tuning, but observability requires architectural decisions about how systems describe themselves. Without that foundation, every downstream improvement—whether rule refinement, model deployment, or AI integration—operates on uncertain ground.

The paper draws on concepts from distributed systems engineering—where observability has been a first-class design principle for over a decade—and applies them to the AML domain. It builds on a four-part article series published on FinCrime Watchdog (April 2026), consolidating the market analysis, layer model, OBASHI case interpretation, and legacy migration strategy into a single, peer-reviewable position paper.

Version 1.7 extends that companion implementation in the direction most demanded by technically skeptical reviewers. The PoC is no longer only a compact executable blueprint; it now includes a larger synthetic workload with healthy and fault-injected transactions, a production-oriented telemetry and storage sketch, and a more concrete AI-governance extension of the event schema. These additions still do not amount to full production validation, but they move the paper from minimal comparative evidence toward a more explicit systems-engineering argument.

In compact form, the paper's claim is this: *monitoring records symptoms; observability reconstructs causes*. That distinction is what separates dashboards from diagnosability.

Contributions. This paper makes five contributions:

1. It reframes AML transaction monitoring as an observability problem, not only a detection problem.
2. It proposes a five-layer AML observability architecture together with the Transformation Spiral as a sequencing model.
3. It formalizes a minimal event-centric implementation blueprint around an AML trace model together with explicit query semantics for `why_flagged`, `why_not_flagged`, and `what_changed`.
4. It introduces a privacy-aware retention model that treats trace retention as a governance object rather than an implementation afterthought, and situates that model in a production-oriented telemetry architecture.
5. It extends the trace model toward AI governance by specifying how model-version, attribution, confidence, and drift artifacts can be embedded into the same compliance trace.
6. It provides a larger synthetic evaluation and a small anti-pattern library showing how observability improves diagnosability relative to a monitoring-only baseline.

2 Related Work

This paper draws on three distinct bodies of literature: observability in distributed systems, architectural patterns for financial transaction processing, and regulatory technology (RegTech). Positioning the contribution requires engaging with each.

2.1 Observability in Distributed Systems

The foundational distinction between monitoring and observability was operationalized for software systems by Majors et al. [Majors et al., 2022] and Sridharan [Sridharan, 2018], building on Kalman's control-theoretic definition [Kalman, 1960]. The OpenTelemetry project [OpenTelemetry, 2024] has since standardized the instrumentation model around logs, metrics, and traces. Recent work has extended observability into multi-cloud and federated environments: Jha [Jha, 2025] proposes federated learning for cross-cloud observability with privacy-preserving model aggregation, while Garnimitta [Garnimitta, 2025] develops a federated AI observability framework for multi-cloud microservices with differential privacy guarantees. These contributions demonstrate that observability architectures can operate under data sovereignty constraints—a finding directly relevant to GDPR-constrained AML systems (see Section 13).

2.2 Event-Driven Architectures for Financial Systems

Event-driven architecture (EDA) has been widely adopted for real-time financial transaction processing. Chilukala [Chilukala, 2025] demonstrates that EDA-based payment systems achieve sub-200ms latencies and 99.99% success rates, while Thompson and Davis [Thompson & Davis, 2023] propose observability-driven development as a design paradigm for distributed financial systems. These architectures share structural similarities with the five-layer stack proposed in Section 6.1: both decompose processing into discrete stages with inter-stage telemetry. However, existing EDA literature focuses on *operational* observability (latency, throughput, availability) rather than *compliance* observability (detection lineage, decision reconstruction, regulatory explainability). Our contribution is the explicit extension of observability from operational concerns to governance and AI governability.

2.3 RegTech and Compliance Automation

The RegTech literature has produced frameworks for standardizing compliance practices across institutions [Arner et al., 2017], and recent work applies advanced monitoring techniques to transaction integrity [Liu et al., 2025]. However, these contributions typically assume that the underlying monitoring infrastructure is adequate and focus on detection logic rather than system-level reconstructability. Schiller García [Schiller García, 2026c] identifies a related structural limitation in the context of RAG-based regulatory audits: the *coverage–verification gap*, where a system's ability to produce compliance assessments outpaces its ability to verify them. AML Observability addresses the architectural precondition that FinRegAgents assumes: without observable data pipelines, even well-designed verification architectures operate on uncertain inputs.

2.4 Positioning This Contribution

Table 1 summarizes how this paper relates to and extends prior work. The core novelty is not the application of observability *per se*, but the argument that observability is the *sequencing precondition* for both governance effectiveness and AI governability in AML—a claim not made in any of the cited works.

2.5 Conceptual Demarcation: What AML Observability Is Not

A legitimate concern is whether "AML Observability" merely relabels established concepts. Table 2 addresses this by distinguishing observability from six adjacent concepts and identifying what each covers and where it falls short.

The key argument is that AML Observability is not a seventh concept alongside the six above, but the *integration principle* that connects them across the full processing lifecycle. Data lineage

Table 1: Positioning relative to related work.

Prior Work	**Focus**	**This Paper Extends By**
Majors et al. (2022)	Observability for distributed systems	Applying observability to compliance/AML domain
Jha (2025), Garnimitta (2025)	Federated observability with privacy	Connecting to GDPR/banking secrecy constraints in AML
Chilukala (2025)	EDA for financial transaction processing	Shifting from operational to compliance observability
Liu (2025)	Advanced transaction monitoring techniques	Adding system-level reconstructability as prerequisite
Arner et al. (2017)	RegTech standardization frameworks	Focusing on architectural preconditions, not policy
Fed/OCC (2011)	Model Risk Management (SR 11-7)	Extending from model validation to full-pipeline observability

Table 2: AML Observability vs. adjacent concepts: what each covers and where it falls short.

Concept	**What It Covers**	**What It Does Not Cover**
Data Lineage	Where data originates and how it flows through systems	Detection logic, model behavior, case outcomes, feedback loops
Audit Trail	Retrospective record of who did what and when	Diagnostic capability: cannot answer *why* a decision was made or what was missed
Explainable AI (XAI)	Post-hoc interpretation of individual model predictions [Fed/OCC, 2011]	Upstream data quality, ingestion completeness, transformation correctness (Layers 1–3)
Process Mining	Discovery and conformance checking of actual process flows [van der Aalst, 2016]	Requires structured event logs that AML batch systems rarely produce; focuses on process, not data/decision quality
Model Risk Mgmt (SR 11-7)	Validation, governance, and controls for quantitative models [Fed/OCC, 2011]	System-level reconstructability; treats model as isolated component, not as part of a five-layer pipeline
Operational Monitoring	System availability, latency, throughput, error rates	Compliance correctness: a system can be "up" and processing transactions while silently missing detections
AML Observability	**End-to-end reconstructability of detection-relevant decisions across all five layers**	**Integrates the above as layer-specific capabilities within a unified diagnostic framework**

serves Layer 1–2, XAI serves Layer 4, audit trails serve Layer 5—but without an architecture that spans all five layers simultaneously, each operates in its own silo. Observability is the architectural property that makes the whole greater than the sum of its parts.

3 Core Propositions

The paper rests on three propositions that together form the argumentative spine.

P1: Observability deficits create epistemic uncertainty in AML systems.
When a system cannot describe its own internal state—which inputs it consumed, which transformations it applied, which signals it ignored—the institution operating it faces epistemic uncertainty, i.e. uncertainty about what the system actually does in production. This is not a theoretical concern: it manifests in the inability to explain false positives, the invisibility of false negatives, and the difficulty of validating detection effectiveness under audit.

P2: Governance without observability remains under-evidenced.
Regulatory frameworks—MaRisk, BAIT, DORA, the forthcoming AMLA regulatory technical standards—increasingly require institutions to *demonstrate* effectiveness, not merely *assert* it. Without system-level observability, governance remains declarative: institutions describe what *should* happen rather than evidencing what *does* happen.

P3: AI without observability is difficult to govern.
The integration of machine learning and AI into AML systems introduces additional complexity. If the system surrounding them is itself not observable, the challenge compounds: the institution cannot explain the model's behavior *or* the data pipeline that feeds it. Observability is therefore a precondition for responsible AI deployment in compliance, not an afterthought.

4 The Conceptual Gap: Monitoring vs. Observability

The distinction between monitoring and observability has not yet been systematically applied to AML. Table 3 makes the mapping explicit:

Table 3: Monitoring vs. Observability—conceptual mapping for AML systems.

Dimension	**Monitoring**	**Observability**
Design premise	Known failure modes are anticipated	Unknown failures must be discoverable
Instrumentation	Predefined dashboards, thresholds	Rich telemetry: logs, metrics, traces
Core question	"Did something happen?"	"Why did it happen—and what did we miss?"
Output	Alerts	Reconstructable understanding
Epistemology	Confirmation of expected patterns	Exploration of system state space

The concept of observability originates from control theory [Kalman, 1960] and was operationalized for distributed software systems by practitioners including Charity Majors, Cindy Sridharan, and the OpenTelemetry project.

5 Structural Limitations of Current AML Systems

5.1 Fragmented Data Flows

Multiple upstream systems deliver data through inconsistent transformations. End-to-end traceability from source record to detection decision typically requires manual reconstruction.

5.2 Opaque Detection Logic

Rules and models operate with limited visibility into feature engineering and decision boundaries.

5.3 Missing Feedback Loops

Case outcomes are rarely reintegrated systematically into model tuning or rule calibration.

5.4 No System-Level Debugging

False negatives remain structurally invisible. This is an architectural consequence of systems designed for detection output rather than self-description.

6 AML Observability: A Conceptual Architecture

We define **AML Observability** as:

> *The capability to reconstruct detection-relevant decisions across inputs, transformations, feature derivations, model inferences, and disposition outcomes—across the full processing lifecycle.*

6.1 The Five-Layer AML Observability Stack

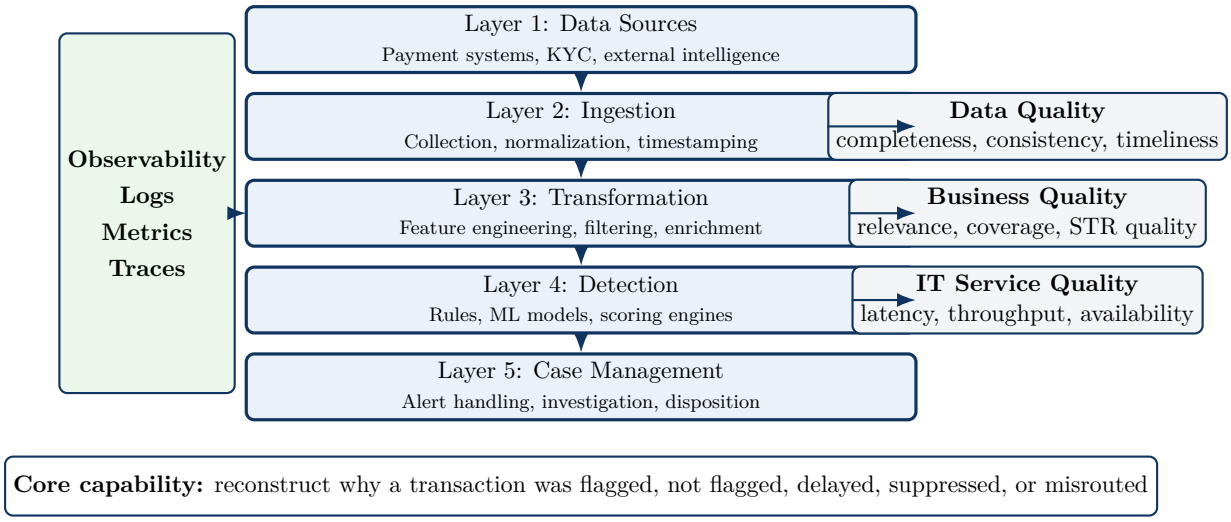


Figure 1: The Five-Layer AML Observability Stack.

6.2 Table View of the Stack

6.3 Observability Pillars

Analogous to distributed systems, AML observability relies on three pillars: **Logs** (structured records of what events occurred within the processing pipeline), **Metrics** (time-series data on system behavior—alert volumes, processing latency, data quality scores, model drift indicators), and **Traces** (end-to-end reconstruction of how a specific transaction flowed through all five layers from source to disposition).

The system must be able to answer four questions for any given transaction: Why was it flagged (or not)? Which data contributed to the decision? What transformations were applied? What signals were ignored or lost?

Table 4: The Five-Layer AML Observability Stack.

Layer	Function	Observability Requirement
1. Data Sources	Payment systems, KYC, external intelligence	Source completeness, freshness, schema validation
2. Ingestion	Collection, normalization, timestamping	Provenance tagging, deduplication metrics, latency
3. Transformation	Feature engineering, filtering, enrichment	Transformation lineage, data quality scores
4. Detection	Rules, ML models, scoring engines	Decision explanation, confidence scores, coverage
5. Case Management	Alert handling, investigation, disposition	Outcome tracking, feedback integration, SLA metrics

6.4 A Hypothetical AML Trace

To make the five-layer architecture concrete, consider a hypothetical transaction trace for a single wire transfer flagged by the AML system. Table 5 illustrates the data points that would be captured at each layer in an observable system.

In a non-observable system, only layers 4 (the alert) and 5 (the disposition) are typically visible. Layers 1–3 are opaque: the institution cannot reconstruct which data fed the decision, whether the PEP status was current, or whether the feature derivation was correct. This trace structure illustrates why observability must span all five layers to answer the four core questions posed in Section 6.3.

The trace also demonstrates the technical requirements: each layer must emit structured telemetry (log entries, metric updates, trace spans) that can be correlated via a shared transaction identifier. This is architecturally analogous to distributed tracing in microservice systems (e.g., OpenTelemetry's `trace_id`/`span_id` model [OpenTelemetry, 2024]), applied to the compliance processing lifecycle rather than the request-response lifecycle.

6.5 Comparative Capability View

One critique of earlier versions of this paper was that they positioned AML Observability against adjacent approaches without comparing them sharply enough. Table 6 makes the distinction explicit.

6.6 Three Quality Dimensions

The critical insight is that these three dimensions are not independent: a data quality failure may manifest as a business quality failure, which in turn may appear as an IT service quality failure.

Table 5: Hypothetical AML trace: a single wire transfer through all five layers.

Layer	Observable Data Points	Diagnostic Question Answered
1. Data Sources	Transaction ID, amount (€48,200), originator (Customer A, PEP status: false), beneficiary (Entity B, jurisdiction: IRN), source system: SWIFT gateway, timestamp: 2026-04-06T09:14:22Z	Was the transaction received? Was PEP/sanctions data current at ingestion time?
2. Ingestion	Normalization applied (ISO 20022 mapping), dedup check (no duplicate), ingestion latency: 340ms, provenance tag: `swift-gw-prod-01`	Did the transaction arrive intact? Was it deduplicated? How fresh is the data?
3. Transformation	Features derived: beneficiary_jurisdiction_risk=high, amount_percentile=97.2, customer_avg_monthly=€12,400, cross-border=true. Data quality score: 0.91 (1 field missing: beneficiary LEI)	Which features were computed? Were any inputs missing or stale?
4. Detection	Rule R-17 triggered (high-risk jurisdiction + amount $>$ 3$\times$ average). ML model score: 0.78. Combined alert score: 0.82. Confidence: high. Alternative rules evaluated but not triggered: R-04, R-11	Why was this flagged? What was the model's contribution? What rules did not fire?
5. Case Mgmt	Alert assigned to analyst, priority: medium. Investigation time: 2.4h. Disposition: escalated to MLRO. STR filed: yes. Feedback: true positive	What happened after the alert? Did the outcome feed back into model tuning?

Table 6: Capability comparison: traditional monitoring, operational observability, and AML observability.

Capability	Traditional Monitoring	EDA / Operational Observability	AML Observability
Latency / throughput visibility	Threshold dashboards	Detailed metrics and traces	Detailed metrics and traces
Source provenance	Limited	Sometimes available	Required as first-class evidence
Feature derivation trace	Rare	Rare	Required for Layer 3 diagnosis
Rule / model decision trace	Alert-focused	Operational only	Required with decision artifacts and confidence
Suppression visibility	Usually opaque	Usually opaque	Explicitly traceable
Case feedback integration	Manual and delayed	Out of scope	Required in Layer 5
SAR / disposition linkage	Often externalized	Out of scope	Built into end-to-end traceability
Compliance debugging question	"Did the job run?"	"Did the service degrade?"	"Why was this flagged, not flagged, or missed?"

Table 7: Three quality dimensions of AML Observability.

Dimension	Scope	Example Metrics
Business Quality	Alert relevance, detection coverage, STR quality	True positive rate, typology coverage ratio
Data Quality	Completeness, consistency, timeliness of inputs	Missing field rate, cross-source consistency
IT Service Quality	Processing latency, availability, throughput	P95 latency, batch completion rate, error rate

7 Technical Specification and Production-Oriented Implementation Blueprint

To address the criticism that the argument remains purely conceptual, this section formalizes a minimal implementation blueprint derived from the companion Beyond-AI AML observability PoC [Schiller García, 2026d]. The implementation is intentionally compact. Its purpose is not to claim production readiness, but to show that the proposed architecture can be expressed as executable data structures, deterministic processing stages, and reconstructable diagnostic queries.

7.1 Design Choice: An Event-Centric Trace Model

The PoC adopts an event-centric design inspired by OpenTelemetry [OpenTelemetry, 2024]. Instead of introducing a separate `AMLSpan` object, the current blueprint represents span-like structure through the tuple of `trace_id`, `span_id`, and `parent_span_id` on each trace event. This keeps the model compact while preserving causal ordering and inter-layer correlation.

The key design choice is therefore:

> *One business transaction corresponds to one `AMLTrace`; one reconstructable processing step corresponds to one `AMLTraceEvent`.*

This design is sufficient for the paper's central purpose: reconstructing decisions across the five-layer AML lifecycle.

7.1.1 Handling Stateful and Temporal Dependencies

A critical distinction between standard IT tracing and AML observability is the reliance on stateful, temporal logic—for example, "customer transaction volume over the last 30 days." The AML trace does *not* attempt to capture the entire 30-day event history inside a single trace. Instead, it captures the **state projection** at the moment of execution.

When Layer 3 evaluates a temporal feature, the `AMLTraceEvent` records the retrieved aggregated value and its temporal boundary directly into the Δ payload—e.g. `{"30_day_vol_eur": 45000, "window_end": "2026-04-06T10:00:00Z"}`. If a false negative occurs because a batch job failed to update the 30-day aggregate table, the trace explicitly surfaces the stale `window_end` timestamp. Diagnosability is achieved by recording the *inputs to the stateful decision*, rather than replicating the state store itself. This design keeps trace volume bounded while still making temporal staleness a first-class observable property.

7.2 Core Domain and Trace Objects

Table 8 summarizes the core objects currently implemented in the PoC.

Table 8: Core objects in the AML observability PoC.

Object	**Role in the PoC**	**Representative Attributes**
`TransactionRecord`	Raw transaction entering Layer 1	`transaction_id`, `amount_eur`, `beneficiary_jurisdiction`, `pep_flag`
`FeatureDerivation`	Transformation-layer output	`amount_multiple`, `beneficiary_jurisdiction_risk`, `data_quality_score`, `missing_attributes`
`DetectionDecision`	Detection-layer result	`triggered_rules`, `model_score`, `combined_score`, `alert_threshold`, `confidence`
`CaseDisposition`	Case-management feedback	`status`, `str_filed`, `feedback_label`, `investigation_time_hours`
`RetentionDecision`	Privacy-aware retention result	`mode`, `reason`, `summary`
`AMLTrace`	Transaction-level root object	`trace_id`, tuple of events
`AMLTraceEvent`	Reconstructable telemetry unit	`layer`, `event_type`, `derived_features`, `decision_artifacts`

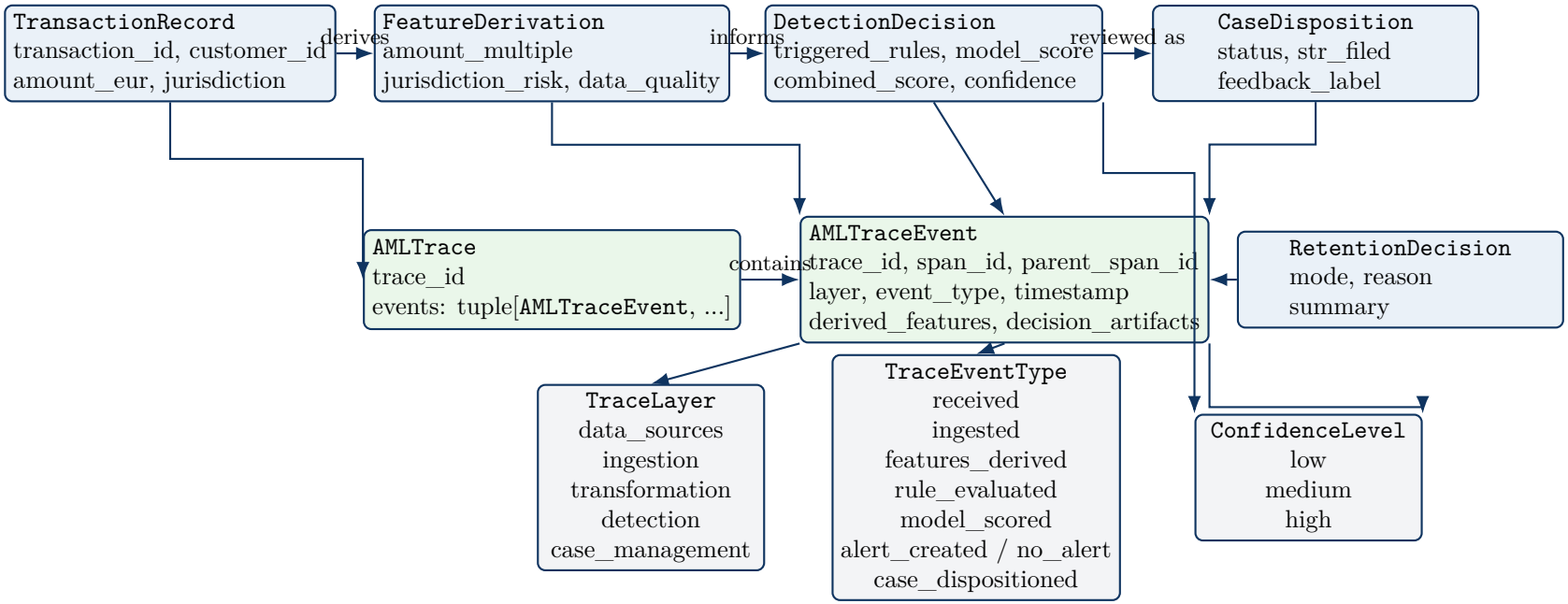


Figure 2: PoC data model: domain objects, trace objects, and semantic enums.

7.3 Formal Trace Schema

The minimal trace schema is intentionally small. Table 9 formalizes the current design.

Table 9: Formal schema of the event-centric AML trace model.

Element	Definition	Interpretation
`AMLTrace`	`(trace_id: str, events: tuple[AMLTraceEvent, ...])`	Root object for one transaction lifecycle
`trace_id`	Shared identifier across all events	Correlates all five layers for one business object
`span_id`	Event-local identifier	Encodes span-like ordering without a separate span class
`parent_span_id`	Optional pointer to prior event	Preserves causal chain across layers
`layer`	One of `DATA_SOURCES`, `INGESTION`, `TRANSFORMATION`, `DETECTION`, `CASE_MANAGEMENT`	Maps telemetry to the five-layer stack
`event_type`	Semantic event label	Distinguishes receipt, ingestion, feature derivation, rule evaluation, model scoring, alert state, and disposition
`derived_features`	Key-value payload	Captures transformation outputs and missing attributes
`decision_artifacts`	Key-value payload	Captures rules, scores, threshold state, priority, and case outcome
`confidence`	Optional normalized confidence	Allows common confidence semantics across decision and event layers

7.3.1 Formal Trace Model as a 7-Tuple

To move beyond a textual schema, we define the trace model algebraically. Let $\mathcal{E}$ be the set of all trace events. An event $e \in \mathcal{E}$ is a 7-tuple:

$$e = \langle t_{id},\ s_{id},\ p_{id},\ L,\ \tau,\ \Delta,\ \Omega \rangle$$

where t_{id} is the global transaction trace identifier, s_{id} is the unique span identifier for the event, p_{id} is the optional parent span identifier establishing a partial order $e_1 \prec e_2 \iff e_2.p_{id} = e_1.s_{id}$, $L \in \{1, 2, 3, 4, 5\}$ denotes the AML processing layer, τ is the execution timestamp, Δ is the dictionary of derived features (e.g. computed risk scores, missing-attribute flags), and Ω is the dictionary of decision artifacts (e.g. triggered rules, model confidences, case outcomes).

An `AMLTrace` is then a pair $T = (t_{id},\ \langle e_1, \ldots, e_n \rangle)$ where the events are ordered by the partial order induced by p_{id}.

7.3.2 Pseudocode for `what_changed`

The diagnostic query `what_changed` compares two traces T_A and T_B for the same transaction entity and returns the set of causal divergence points. Algorithm 3 specifies the procedure.

Complexity. Since the maximum depth of an AML trace is strictly bounded by the number of architectural layers and sub-components (typically $|E| \leq 20$), this query executes in $O(|E|)$ time, making it suitable for real-time diagnostic retrieval. The symmetric difference on each dictionary pair is $O(k)$ where k is the number of keys, which is similarly bounded by the finite set of features and artifacts per layer.

7.4 End-to-End Processing Flow

Figure 4 mirrors the current PoC pipeline. It is not merely illustrative: each node corresponds to an implemented stage or object in the companion repository.

Algorithm 1: `what_changed`(T_A, T_B)

Input: Trace T_A (baseline), Trace T_B (current)
Output: List of divergence records D

$D \leftarrow [\,]$
for each layer $L \in \{1, 2, 3, 4, 5\}$ **do**
  $e_A \leftarrow \textsc{GetEvent}(T_A, L)$; $e_B \leftarrow \textsc{GetEvent}(T_B, L)$
  $\delta_\Delta \leftarrow e_A.\Delta \triangle e_B.\Delta$ *// symmetric difference on features*
  if $\delta_\Delta \neq \emptyset$ **then** $D.\textsc{Append}(\langle L,$ `FEATURE_DRIFT`$, \delta_\Delta \rangle)$
  $\delta_\Omega \leftarrow e_A.\Omega \triangle e_B.\Omega$ *// symmetric difference on artifacts*
  if $\delta_\Omega \neq \emptyset$ **then** $D.\textsc{Append}(\langle L,$ `DECISION_DRIFT`$, \delta_\Omega \rangle)$
end for
return D

Figure 3: `what_changed` trace comparison algorithm.

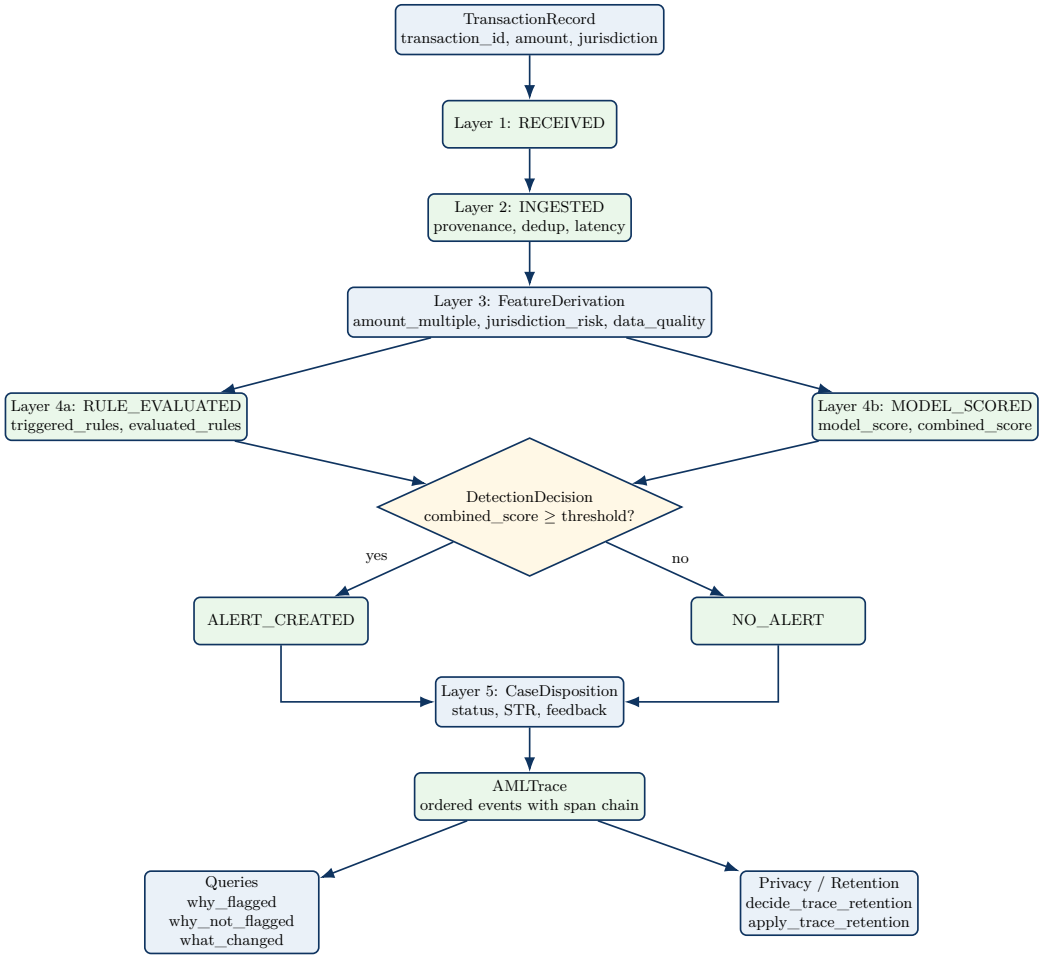


Figure 4: PoC execution flow from transaction input to queryable trace and retention decision.

7.5 Query and Retention Semantics

The PoC implements three compliance-oriented diagnostic queries:

- `why_flagged(trace)` reconstructs which rules fired, what the model score was, and which threshold condition created the alert.
- `why_not_flagged(trace)` reconstructs why a transaction remained below threshold and

surfaces the absence of sufficient evidence as an explicit trace result rather than a silent non-event.

- `what_changed(previous, current)` compares two traces and summarizes changes in features, scores, and alert state.

In addition, the PoC implements privacy-aware retention with two modes:

- **Full retention** for alerted transactions or statistical samples.
- **Summary retention** for non-alert transactions, where only a minimized subset of events is retained.

This is directly relevant to the GDPR objection often raised against observability: the architecture does not require that every trace be retained in full indefinitely. Instead, the retention decision itself becomes observable and auditable.

7.6 Technical Novelty in the AML Application

The observability primitives used in the PoC are intentionally borrowed from distributed systems engineering rather than invented from scratch. The technical novelty of this paper therefore does not lie in proposing a new tracing primitive. It lies in how those primitives are adapted, constrained, and recombined for AML compliance pipelines. Table 10 makes this differentiation concrete by mapping the `AMLTraceEvent` schema against two established standards: OpenTelemetry (OTel) spans and IEEE XES process mining logs.

Table 10: Schema comparison: `AMLTraceEvent` vs. OpenTelemetry and IEEE XES.

Attribute	**OpenTelemetry Span**	**IEEE XES Log**	**AMLTraceEvent (Proposed)**
Causal ordering	`trace_id`, `parent_span_id`	Flat `trace_id`, temporal order only	`trace_id`, `parent_span_id` with layer-enforced DAG
Domain layering	Generic string `span.name`	Generic string `concept:name`	Strict $L \in \{1, 2, 3, 4, 5\}$ enforcing pipeline stage validation
State payloads	Generic `attributes` (mixed key-value)	Flat event attributes	Explicit Δ (derived features) separated from Ω (decision artifacts)
Governance artifacts	Mixed into standard logs	Usually absent	Dedicated Ω carrying XAI, SHAP, rule logic, and drift indicators
Missing evidence	Implicit (span simply absent)	Implicit (event absent)	Explicitly queryable via `why_not_flagged` semantics
Retention governance	Not modeled	Not modeled	First-class `RetentionDecision` object with auditable mode and reason

The strict payload separation (Δ vs. Ω) is necessary because, in AML, *how* a feature was derived is a fundamentally distinct regulatory object from *how* a model weighted that feature. Generic tracing frameworks collapse both into a single attribute bag, making compliance-specific queries impossible without post-hoc parsing.

First, AML observability requires *causal multi-layer debugging for regulatory failures*. Standard distributed tracing is optimized for latency, throughput, and service health. By contrast, an AML trace must explain why a regulatory decision was made, not made, delayed, or suppressed. That requirement leads to domain-specific query semantics such as `why_not_flagged`, which have no direct equivalent in ordinary operational tracing.

Second, AML observability requires *privacy-compliant retention as a governance object*. In this setting, retention is not merely a storage optimization. The decision to keep a full trace, retain only a minimized summary, or sample a non-alert transaction becomes itself a governed and auditable outcome. This is a domain-specific response to the tension between GDPR data minimization and compliance reconstructability.

Third, AML observability introduces *architectural anti-pattern detection* as a diagnostic goal. The anti-patterns derived later in the paper—*Upstream Blindspot*, *Silent Transformation Drop*, *Threshold Suppression*, and *Broken Feedback Loop*—are not generic observability concepts. They are a compact taxonomy of recurring compliance-system failures obtained by interpreting AML enforcement cases through a traceability lens.

In short, the contribution is not "OpenTelemetry for banking" in a narrow sense. It is the domain-specific integration of trace semantics, privacy-aware retention, and failure-pattern analysis into a compliance architecture that can answer regulatory and governance questions rather than purely operational ones.

7.7 Production Architecture and Scalability

The current PoC uses an in-memory collector because its primary purpose is explanatory. A production deployment would replace that collector with a distributed telemetry path: instrumentation in each AML layer emits events to a collector tier, the collector batches and forwards events to a partitioned transport, and the retained traces are stored in a columnar analytical backend or equivalent observability store. One concrete realization would use OpenTelemetry-compatible SDKs, a collector process, a partitioned message bus, and a query-optimized warehouse keyed by `trace_id`. The architectural point is less the specific vendor choice than the separation between event emission, event transport, and compliance-oriented querying.

The PoC now allows a first sizing estimate grounded in actual serialized traces rather than only intuition. Across a synthetic workload of 1,000 transactions, the current event structure produces an average of 6.1 events per transaction and roughly 3.63 KB of raw JSON telemetry per full trace. Extrapolated linearly, this corresponds to about 3.38 GB of raw telemetry per one million processed transactions. Under the implemented selective-retention policy—full retention for alerts plus a 5% sample of non-alerts—the retained footprint falls to approximately 2.05 GB per one million transactions in the current PoC serialization. These values are not production benchmarks; they are engineering sizing indicators derived from the companion implementation.

Query cost is similarly small at the PoC layer. Over 200 traces processed in memory, the mean local execution times were approximately 0.0064 ms for `why_flagged`, 0.0058 ms for `why_not_flagged`, and 0.0073 ms for `what_changed`. In production, end-to-end latency would of course be dominated by storage, indexing, and transport rather than by the query logic itself. Still, the local measurements support the narrower claim that the diagnostic semantics introduced in this paper do not require computationally exotic algorithms; they are lightweight query patterns over a modest per-trace event set.

These results suggest that scalability is not primarily blocked by the complexity of the diagnostic queries. The harder engineering problems lie elsewhere: instrumenting heterogeneous source systems, integrating batch and near-real-time stages, choosing retention and sampling policies, and operating the storage tier under regulatory hold periods. That is precisely why this paper treats production architecture as part of the observability argument rather than leaving it as an implementation afterthought.

8 Minimal Proof of Concept and Walkthrough

The PoC is implemented as a small Python package with separated responsibilities. Table 11 summarizes the current module layout.

Table 11: Minimal implementation blueprint of the companion PoC.

Module	Responsibility
`models.py`	Domain objects for transaction input, feature derivation, detection decision, case disposition, and retention decision
`shared/observability/trace.py`	Event-centric trace primitives: `AMLTrace`, `AMLTraceEvent`, `TraceLayer`, `TraceEventType`
`collector.py`	In-memory trace collector used by the demo pipeline
`detector.py`	Feature derivation, rule evaluation, model scoring, combined threshold logic
`case_mgmt.py`	Priority assignment and default case feedback for alerted transactions
`privacy.py`	Selective retention and trace minimization
`pipeline.py`	Thin orchestration layer for the end-to-end walkthrough
`queries.py`	Compliance-oriented diagnostic questions over traces
`evaluation.py`	Comparative evaluation harness for small and large synthetic fault batteries
`scripts/run_evaluation.py`	Reproducible command-line entry point for evaluation runs and markdown summaries
`scripts/run_demo.py`	Executable demo on synthetic JSONL fixtures
`notebooks/walkthrough.ipynb`	Interactive walkthrough of flagged, non-flagged, and changed-trace cases

8.1 A Simulated False-Negative Walkthrough

One of the review requests was to explain how observability improves AI governance beyond isolated model monitoring. The PoC addresses this with a simple but concrete scenario: two otherwise similar transactions are processed, one with a complete high-risk jurisdiction value and one with that field missing.

Table 12: Simulated cross-layer false-negative walkthrough.

Step	Complete Transaction	Degraded Transaction
Layer 1 input	`beneficiary_jurisdiction = IRN`	`beneficiary_jurisdiction = None`
Layer 3 transformation	`jurisdiction_risk = high`; higher data quality	`jurisdiction_risk = unknown`; lower data quality; missing attribute logged
Layer 4 detection	Rule path can fire; model score remains elevated	Rule path no longer fires; model score falls below threshold
Layer 5 outcome	Alert or escalated case remains explainable	No alert, but the reason for the miss is still reconstructable
Governance insight	Typology captured	False negative becomes diagnosable as an upstream data-quality failure rather than an unexplained model miss

This example matters because isolated model monitoring would typically observe only the changed score at Layer 4. By contrast, the full trace shows that the score changed for an upstream reason: the transformation layer explicitly records `missing_attributes`, lower `data_quality_score`, and changed feature semantics. This is precisely the kind of cross-layer evidence needed for AI governance in AML. It links the model output back to data provenance and transformation behavior rather than treating the model as a self-contained black box.

8.2 Why This Matters for AI Governance

The technical implication of Proposition P3 can now be stated more concretely. Observability supports AI governance in at least three ways:

1. It captures training- and scoring-relevant feature provenance at Layers 1–3, making degraded or stale input states visible.
2. It preserves rule-based and model-based artifacts in the same trace, allowing hybrid decisions to be reconstructed rather than split across disconnected systems.
3. It links downstream feedback from case management back to upstream processing states, enabling evidence-based recalibration rather than intuition-driven tuning.

In other words, observability does not replace XAI or SR 11-7-style model governance. It provides the missing system context within which those practices become operationally meaningful.

Table 13 shows one concrete extension path for the `AMLTraceEvent` schema. The point is not that every AML deployment must store every artifact shown below. The point is that the same event model can carry model-specific explanation payloads inside `decision_artifacts`, allowing XAI and MLOps signals to be analyzed in the same historical trace as the surrounding pipeline state.

Table 13: Illustrative `AMLTraceEvent` extension for AI governance.

Field	Example	Governance Use
`model_version`	`xgb_v2.3.4`	Identifies the exact model state used for scoring and supports rollback detection
`feature_importance`	`{"amount_pctl": 0.78}`	Embeds local SHAP-like attribution inside the transaction trace
`confidence_distribution`	`[0.10, 0.70, 0.20]`	Preserves uncertainty information for borderline or escalated cases
`training_data_version`	`q1_2026`	Links a scoring event back to the training corpus version and provenance controls
`drift_score`	`0.15`	Captures divergence from the training distribution at scoring time

This extension also makes the AI-governance claim more operational. Consider a model whose alert confidence for a high-risk jurisdiction falls materially between two quarters while the nominal model version remains unchanged. Without observability, the analyst only sees the changed score. With observability, `what_changed` can link that lower score to a transformation-layer mapping change, a drift signal, or both. The result is not just a model-monitoring alert, but a causal debugging path that connects AI behavior back to the pipeline states that produced it.

8.2.1 Beyond Simple Missing Features: Complex Model Artifacts

The false-negative walkthrough above illustrates a simple case—a missing field. But real AML deployments increasingly use non-linear models (Gradient Boosted Trees, Graph Neural Networks, deep sequence models) whose behavior cannot be explained by a single absent input. The Ω dictionary in the trace schema is designed to accommodate this complexity.

For a complex model at Layer 4, the `AMLTraceEvent` does not merely record `score=0.88`. It embeds the top-k SHAP (SHapley Additive exPlanations) values or attention weights directly into Ω—for example, `{"top_shap": {"rapid_movement": 0.45, "dist_to_cluster": 0.22}}`. For Graph Neural Networks operating on customer-entity networks, the trace can capture the local subgraph embedding vector or the top contributing neighbor nodes.

Furthermore, model drift indicators can be recorded at scoring time. A natural choice is the Kullback–Leibler divergence $D_{\mathrm{KL}}(P_{\mathrm{current}} \| P_{\mathrm{train}})$ between the current transaction's feature distribution and the training baseline, computed over a sliding window and stored as `drift_kl` in Ω. When `what_changed` detects a rising `drift_kl` concurrent with falling model scores, it can surface this as a `DECISION_DRIFT` divergence linked to the specific transformation-layer states that produced the shifted features.

The architectural point is that XAI and MLOps signals are not external add-ons. They are native payloads within the same trace structure used for compliance diagnostics, queryable through the same `what_changed` semantics, and subject to the same privacy-aware retention governance.

8.3 A Larger Synthetic Comparative Evaluation

The PoC now supports a larger controlled comparison between a *monitoring-only baseline* and a fully *observable system* built on the proposed trace model. The aim remains narrow: not to benchmark production throughput or detection quality, but to test the architectural claim that observability improves *diagnosability*. Both systems process the same synthetic workload. The baseline exposes only the final alert state and score, while the observable system exposes full cross-layer traces and the diagnostic queries `why_flagged`, `why_not_flagged`, and `what_changed`.

8.3.1 Setup and Hypotheses

Three hypotheses are tested:

1. **H1:** The observable system achieves higher diagnosis completeness than the monitoring-only baseline.
2. **H2:** The observable system identifies the correct failure layer more reliably than the baseline.
3. **H3:** The observable system reduces explanation steps relative to the baseline.

The workload contains 1,000 synthetic transactions. Of these, 700 are healthy no-fault transactions and 300 are fault-injected transactions distributed evenly across six fault families. Table 14 summarizes the fault families.

8.3.2 Operational Definition of Diagnosability

Let $E_{\mathrm{system}}(t)$ denote the set of events actually exposed by a system for transaction t, and let $E_{\mathrm{required}}(f)$ denote the minimal event set an expert would need to diagnose failure mode f. Diagnosability then depends on how much of the required causal record is visible:

$$\mathrm{DiagnosisCompleteness}(f,t) = \frac{|E_{\mathrm{system}}(t) \cap E_{\mathrm{required}}(f)|}{|E_{\mathrm{required}}(f)|}.$$

Table 14: Fault families used in the larger synthetic evaluation.

Fault Family	Count	Expected Layer	Injected Condition
False positive via rule floor	50	Detection	Rule R-17 forces the score across threshold and downstream case feedback closes the alert as false positive
False negative from missing jurisdiction	50	Transformation	A required jurisdiction field is removed upstream, degrading Layer 3 feature semantics and suppressing escalation
Transformation mapping error	50	Transformation	Jurisdiction mapping silently downgrades a high-risk feature into a standard-risk state
Stale PEP state at source	50	Data sources	A stale source-side PEP state removes model contribution before detection logic is evaluated
Silent suppression of R-17	50	Detection	Rule logic matches a risky pattern but the decisive rule is configured as suppressed
Case backlog without feedback closure	50	Case management	Alert creation is correct, but downstream case handling stalls and feedback remains pending

In the observable system, $E_{\text{system}}(t)$ spans all five layers via structured trace events. In the monitoring-only baseline, $E_{\text{system}}(t)$ is intentionally much smaller, typically limited to the final alert outcome and a small number of detection-layer signals. The claim tested in this section is therefore architectural: systems with only end-state monitoring should be systematically weaker at reconstructing upstream, suppressed, or downstream feedback failures than systems that retain cross-layer traces.

For the synthetic evaluation, diagnosis completeness is operationalized as the overlap between expected causal markers and visible markers:

$$\text{DiagnosisCompleteness} = \frac{|M_{\text{expected}} \cap M_{\text{visible}}|}{|M_{\text{expected}}|}.$$

Failure-layer accuracy records whether the system reveals the expected causal layer for the scenario. *Explanation steps* is a simple analyst-effort proxy: the baseline is assigned four reconstruction steps because it requires manual inference beyond the alert outcome, while the observable system is assigned one step because the required explanation is returned directly from the trace-aware query layer.

8.3.3 Results

Table 15 reports aggregate results both for the full 1,000-transaction workload and for the 300 fault-injected transactions only. This split is important. Healthy transactions should be easy for both systems; the real question is how each system behaves when diagnosis is actually needed.

The fault-only slice is the more important result. Across all 300 injected failure cases, the monitoring-only baseline remains substantially incomplete: it exposes only 25% of the required causal markers on average, identifies the correct failure layer in only 17% of cases, and never reaches high diagnosis completeness. By contrast, the observable system consistently exposes the required causal markers and the correct failure layer through the trace-aware query path.

Table 15: Aggregate results of the larger synthetic evaluation.

Scope	System	Mean Diagnosis Completeness	Failure-Layer Accuracy	Mean Explanation Steps	Completeness ≥ 0.90
All transactions	Monitoring-only baseline	0.78	0.75	4.00	0.70
All transactions	Observable system	1.00	1.00	1.00	1.00
Fault-only	Monitoring-only baseline	0.25	0.17	4.00	0.00
Fault-only	Observable system	1.00	1.00	1.00	1.00

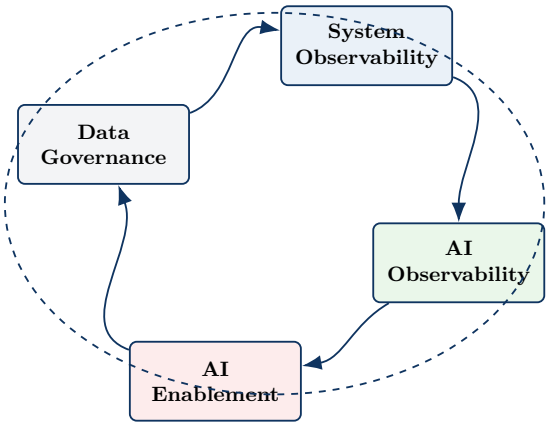
These numbers should still be interpreted narrowly. They do not prove that observability improves real-world detection effectiveness, nor do they demonstrate banking-scale production behavior. They do, however, provide stronger empirical support than the earlier three-scenario comparison for the paper's central architectural claim: when the same fault families are injected into the same pipeline, end-to-end observability materially improves the system's ability to explain why an alert fired, failed, or changed.

8.3.4 Falsifiability and Scope

The architectural claim tested here is falsifiable. If a monitoring-only system could reliably reconstruct Layer 1–3 causal failures without Layer 1–3 instrumentation, the present argument would require revision. Likewise, if rule suppression or downstream feedback absence could be diagnosed with the same completeness from end-state alert telemetry alone, the case for full observability would weaken materially. The current synthetic results do not establish such counterexamples. Instead, they support the narrower proposition that diagnosability gaps in non-observable systems are structural rather than incidental.

9 The AML Transformation Spiral

A recurring question in the industry is sequencing: should institutions invest in AI, in governance, or in infrastructure first? We propose a **Transformation Spiral** as a heuristic sequencing model—not an empirically validated maturity framework, but a tool for making implicit dependencies between governance, observability, and AI explicit and discussable.



Maturity increases outward. Institutions may enter at any stage, but stable AI adoption requires closing the observability loop.

Figure 5: The AML Transformation Spiral.

Institutions may enter the spiral at any point. However, entry flexibility does not imply equal stability: the further institutions progress without establishing observability, the greater the accumulation of opacity, control debt, and rework. Maturity requires closing the loop.

10 Case Interpretation and Recurring Anti-Patterns

Methodological note: These interpretations are illustrative, not causal proof. The selected cases represent widely documented, high-impact enforcement actions with publicly available descriptions of control breakdowns, allowing cross-case comparison despite limited architectural detail.

Table 16: OBASHI-informed observability anti-patterns in five enforcement cases.

Case	**Fine**	**Anti-Pattern**	**Observability Gap (Interpreted)**
Danske Bank Estonia	$2B	Upstream Blindspot	No provenance for non-resident flows across subsidiary boundary
Westpac (AUSTRAC)	$1.3B AUD	Silent Transformation Drop	IFTI reporting gap in batch processing; silent failure
Capital One (FinCEN)	$390M	Broken Feedback Loop	SAR backlog without systematic triage or feedback
Swedbank	$386M	Upstream Blindspot	Baltic subsidiary data not integrated into group monitoring
NatWest	£265M	Threshold Suppression	Cash deposit alerts suppressed by static thresholds

The recurring pattern is that each enforcement action can be mapped not only to a layer, but to a repeatable anti-pattern of non-observable system behavior. Danske Bank and Swedbank share an *Upstream Blindspot*: subsidiary or cross-boundary data did not flow into group-level monitoring, a Layer 1–2 failure that no amount of detection tuning at Layer 4 could have compensated. Westpac's IFTI gap represents a *Silent Transformation Drop*: the system processed transactions but silently lost a reporting-relevant obligation inside transformation logic. Capital One illustrates a *Broken Feedback Loop*: the SAR backlog was not only a detection issue but a case-management bottleneck that failed to recalibrate upstream controls. NatWest shows *Threshold Suppression*: a detection-layer configuration that actively prevented alerts despite risky underlying patterns. These are not merely isolated implementation failures; they are recurring diagnosability gaps that an observability-first design would likely have made more visible.

10.1 A Small Anti-Pattern Library

The anti-pattern framing matters because it moves the paper from isolated case commentary to a reusable diagnostic vocabulary:

1. **Upstream Blindspot.** Inputs required for meaningful detection never become visible at group or pipeline level. This produces structurally silent failures before detection logic is even reached.

2. **Silent Transformation Drop.** The system appears operational, but transformation logic silently removes, corrupts, or fails to enrich a detection-relevant field. The alerting layer then appears to underperform for reasons it cannot itself explain.

3. **Threshold Suppression.** Detection logic matches a risky pattern, but thresholding, suppression rules, or configuration choices prevent escalation. Without explicit decision traces, the miss looks like ordinary non-alert behavior.

4. **Broken Feedback Loop.** Case-management outcomes do not feed back into upstream rules, features, or models. The system therefore accumulates decision debt because it cannot connect operational outcomes to the processing states that produced them.

These anti-patterns are not offered as a complete taxonomy. They are a compact pattern library showing how observability deficits recur across institutions and layers. Their value is explanatory: they help distinguish accidental implementation defects from architectural blind spots that monitoring-only systems are poorly equipped to diagnose.

11 Migration Strategy: The Strangler Fig Pattern

Table 17: Strangler Fig migration phases for AML observability.

Phase	Action	Value Created
1. Instrument	Add observability probes without changing behavior	Immediate visibility; baseline for improvement
2. Extract	Move processing steps into observable microservices	Parallel validation; gradual capability transfer
3. Replace	Redirect traffic once parity is validated	Full observability; legacy decommission

Phase 1 creates immediate value—visibility into the current system's actual behavior—before any replacement occurs.

12 Implications

12.1 Regulatory Compliance

AML observability directly supports traceability, data lineage, and explainability requirements from AMLA's RTS and the AMLR effectiveness mandate. These expectations are consistent with broader international supervisory trends, including FATF guidance on effectiveness and risk-based supervision.

12.2 Operational Efficiency

Observable systems enable systematic false positive reduction through root cause analysis, faster investigation through transaction-level tracing, and evidence-based model improvement.

12.3 Risk Management

An observable AML system can identify blind spots, detect systemic data flow weaknesses, and answer: "How do you know your system works?"

13 Discussion and Limitations

The proposed framework carries inherent limitations that must be addressed transparently.

The current evaluation is still synthetic. Version 1.7 moves beyond a purely illustrative three-scenario comparison by evaluating 1,000 synthetic transactions with 300 injected faults, but the resulting evidence is still controlled and programmatic. The reported gains show improved diagnosability under the fault families encoded in the PoC; they do not yet establish detection-performance gains, human-analyst time savings in live operations, or robustness under the full complexity of real banking environments. The next research step is therefore validation over richer synthetic workloads, de-identified institution data, or industry partnerships.

Scalability has only been sized, not validated in production. The new production-architecture subsection grounds the discussion in measured PoC trace sizes and local query latencies, but these remain sizing indicators rather than production benchmarks. Real deployments would need to account for heterogeneous transports, storage compaction, back-pressure, failure recovery, and regulatory hold periods. However, recent work on federated observability [Jha, 2025, Garnimitta, 2025] suggests that privacy-preserving aggregation techniques can reduce the data volume that must be centralized, potentially making observability viable even under strict data minimization constraints.

GDPR and data minimization. A tension exists between the observability goal (reconstruct everything) and the GDPR principle of data minimization (store only what is necessary). The PoC suggests two design strategies: (1) *selective trace retention*, where full traces are retained only for transactions that trigger alerts or that fall into statistical sampling windows, with summary metrics retained for all transactions; and (2) *on-demand trace reconstruction*, where raw telemetry is retained in encrypted, time-limited storage and full traces are assembled only when needed for investigation or audit. These strategies require careful legal analysis, but they demonstrate that observability and data minimization are not inherently contradictory.

Vendor lock-in and closed-box systems. The Strangler Fig migration strategy (Section 11) assumes that institutions can instrument their existing systems. In practice, many AML platforms (e.g., SAS, NICE Actimize, Oracle FCCM) are proprietary and offer limited API access to internal processing states. Phase 1 instrumentation in such environments must rely on *peripheral observability*: capturing inputs to and outputs from the closed-box system, monitoring batch job completion, and comparing expected versus actual alert volumes. This provides partial but immediate visibility. Full observability requires either vendor cooperation (exposing internal telemetry via APIs) or eventual replacement of opaque components—which is precisely what Phases 2 and 3 of the Strangler Fig pattern address.

AI governance integration is still a schema extension, not a full MLOps implementation. The paper now makes the AI-governance claim more concrete by specifying how attribution, confidence, and drift artifacts can be embedded in `AMLTraceEvent`. Even so, the PoC does not yet implement deep sequence models, graph features, embedding stores, or full lineage across training and inference pipelines. The paper's claim remains narrower: observability provides the structural context into which those artifacts can be inserted and queried.

Cultural shift. Moving from compliance-as-checkbox to engineering-driven approaches requires organizational transformation. The Transformation Spiral acknowledges this: maturity is iterative, not binary.

Open questions. How can observability standards be harmonized across institutions, potentially through AMLA's supervisory convergence mandate? Can federated observability architectures—building on the cross-cloud frameworks proposed by Jha [Jha, 2025]—enable cooperative compliance across institutions without centralizing sensitive data? What role should AI play not only in detection but in interpreting observability data, moving toward autonomous compliance diagnostics? And finally: can the coverage–verification gap formalized by Schiller García [Schiller García, 2026c] be narrowed systematically through observability-driven feedback loops?

14 Conclusion

The sequencing principle is clear:

> *First make the system legible. Then govern it. Then make it smarter.*

The five enforcement cases analyzed suggest that several high-impact AML failures of the past decade can plausibly be interpreted, at least in part, as observability failures—failures to know what the system was actually doing.

Version 1.7 strengthens that argument one step further. By formalizing an event-centric trace model, extending it toward AI-governance artifacts, grounding it in a production-oriented telemetry architecture, and comparing it against a monitoring-only baseline over a larger synthetic workload, the paper now shows both *how* observability can be built and *why* its absence produces repeatable diagnosability gaps. That is still not the final empirical word, but it is a clearer bridge from architectural thesis to engineering evidence.

Observability First is not a competing paradigm. It is a sequencing principle, a diagnostic discipline, and arguably one the AML industry has not yet fully adopted.

References

[Majors et al., 2022] Majors, C., Fong-Jones, L., Miranda, G. *Observability Engineering*. O'Reilly Media, 2022.

[Sridharan, 2018] Sridharan, C. *Distributed Systems Observability*. O'Reilly Media, 2018.

[Alder & Wallis, 2017] Alder, F., Wallis, P. *OBASHI: Business & IT Alignment and Governance*. Van Haren Publishing, 2017.

[Fowler, 2004] Fowler, M. "StranglerFigApplication." martinfowler.com, 2004.

[Kalman, 1960] Kalman, R. E. "On the General Theory of Control Systems." *IFAC Proceedings*, 1(1), 1960.

[FATF, 2021] FATF. *Guidance on Risk-Based Supervision*. 2021.

[EU, 2024a] Regulation (EU) 2024/1624 (AMLR). Official Journal of the EU, 2024.

[EU, 2024b] Regulation (EU) 2024/1620 establishing AMLA. Official Journal of the EU, 2024.

[BaFin, 2024] BaFin. *Auslegungs- und Anwendungshinweise zum GwG*. Updated 2024.

[OpenTelemetry, 2024] OpenTelemetry Project. "OpenTelemetry Specification." opentelemetry.io, 2024.

[Schiller García, 2026a] Schiller García, J. "AML Observability Series (Parts I–III + OBASHI Analysis)." *FinCrime Watchdog*, April 2026. watchdog.endvater.de

[Schiller García, 2026b] Schiller García, J. "272 Datenpunkte und ein Paradigmenproblem." *BKR* (forthcoming), C.H. Beck Verlag, 2026.

[Jha, 2025] Jha, N. N. "Federated learning for cross-cloud observability: Privacy-preserving model aggregation across distributed cloud platforms." *World Journal of Advanced Research and Reviews*, 2025.

[Garnimitta, 2025] Garnimitta, B. "Federated AI Observability in Multi-Cloud Microservices: A Secure and Scalable Federated Learning Perspective." *International Journal of Engineering and Advanced Technology Studies*, 13(3), 20–31, 2025.

[Chilukala, 2025] Chilukala, S. "Event-Driven Architectures in FinTech: Enabling Real-Time Payment Processing and Settlement." 2025.

[Thompson & Davis, 2023] Thompson, H., Davis, F. "Observability-driven development: A new paradigm for distributed financial systems." 2023.

[Liu et al., 2025] Liu, Y. et al. "Advanced Techniques in Real-Time Monitoring for Financial Transaction Integrity." 2025.

[Arner et al., 2017] Arner, D., Barberis, J., Buckley, R. "FinTech, RegTech, and the Reconceptualization of Financial Regulation." *Northwestern Journal of International Law and Business*, 37(3), 2017.

[Schiller García, 2026c] Schiller García, J. "FinRegAgents: A Multi-Agent RAG Framework for AI-Assisted Financial Regulatory Audits with Confidence-Aware Validation." *aiXiv*, aixiv.260228.000002, February 2026.

[Schiller García, 2026d] Schiller García, J. *Beyond-AI: AML Observability Proof of Concept*. GitHub repository, 2026. https://github.com/endvater/beyond-ai

[van der Aalst, 2016] van der Aalst, W. M. P. *Process Mining: Data Science in Action*. 2nd ed., Springer, 2016.

[Fed/OCC, 2011] Board of Governors of the Federal Reserve System / Office of the Comptroller of the Currency. "Supervisory Guidance on Model Risk Management (SR 11-7)." April 2011.

[Baesens et al., 2021] Baesens, B., Hillard, R., Soetemans, M. *Analytics and AI for AML and Financial Crime Detection*. Wiley, 2021.

This paper was developed with AI-assisted drafting under human editorial oversight by the author. The underlying research, conceptual framework, case interpretation, Transformation Spiral model, PoC architecture, and all editorial decisions are the author's own work.