International Journal of Artificial Intelligence, Data Science, and Machine Learning
Grace Horizon Publication | Volume 7, Issue 1, 373-380, 2026
ISSN: 3050-9262 | https://doi.org/10.63282/3050-9262.IJAIDSML-V7I1P159

Original Article

Managing Clinical Data Lineage in Distributed Healthcare Integration Environments: A Metadata Instrumentation Framework for End-to-End Provenance Tracking

Sindhukumar Sundaram
Tennessee, USA.

Received On: 15/02/2026 Revised On: 14/03/2026 Accepted On: 17/03/2026 Published On: 20/03/2026

Abstract *- Healthcare integration engines transform clinical messages through multiple processing stages before delivery to downstream systems, yet no standardized framework exists for tracking field-level transformation provenance within middleware architectures. This gap impedes data quality root-cause analysis, complicates regulatory audit response, and undermines trust in derived clinical datasets. This paper proposes the Healthcare Integration Lineage Instrumentation Framework (HILIF) a metadata-driven architecture for capturing field-level transformation provenance across distributed integration engine topologies. HILIF introduces a lineage event model grounded in W3C PROV ontology, extended for HL7-specific transformation semantics. A sidecar instrumentation layer captures transformation events asynchronously through a lock-free ring buffer, persisting them to a dual-store architecture (relational + graph) that supports three query patterns: forward-trace, backward-trace, and impact analysis. Evaluation across a simulated enterprise environment 120 channels, 14 transformation stages, 450 messages/second sustained over 72 hours demonstrates a mean latency overhead of 3.2 ms per message, sustained throughput reduction of 1.5%, and lineage event capture completeness of 99.9999%. Forward-trace queries resolve in under 200 ms median, and FHIR Provenance resource generation achieves 97.8% structural validation. HILIF provides a replicable framework for audit-ready provenance tracking in enterprise healthcare integration.*

Keywords *- Data Lineage, Provenance Tracking, Healthcare Interoperability, Integration Engine, HL7, FHIR, Metadata Governance, W3C PROV, Distributed Systems.*

1. Introduction

Modern healthcare delivery depends on continuous clinical data exchange between disparate information systems[1]. Enterprise health systems routinely operate hundreds of concurrent integration channels, processing millions of Health Level Seven (HL7) v2.x, Fast Healthcare Interoperability Resources (FHIR)[2], and Clinical Document Architecture (CDA)[3] messages daily. As data traverses these pipelines, it undergoes field mapping, vocabulary translation, conditional filtering, enrichment, and content-based routing each operation altering data in ways that may be clinically significant.

Despite these pervasive transformations, no standardized framework tracks the complete lineage of a data element through the integration layer. When a downstream system receives a lab result or discharge diagnosis, fundamental questions remain unanswerable: What was the original value before transformation? Which rule converted the vocabulary code? Where in the pipeline was an error introduced?

This absence creates operational, clinical, and regulatory risks. Data quality incidents require manual forensic log analysis. Clinicians receiving transformed data cannot assess its fidelity. Regulatory frameworks including the Health Insurance Portability and Accountability Act (HIPAA), the 21st Century Cures Act, and the Trusted Exchange Framework and Common Agreement (TEFCA) increasingly demand demonstrable accountability for data transformation.

While FHIR defines a Provenance resource grounded in W3C PROV, the US Core Provenance Profile focuses on the "last hop" organizational-level attribution [4]. The entire integration layer, where the most consequential transformations occur, remains an untracked provenance gap.

This paper addresses this gap with the Healthcare Integration Lineage Instrumentation Framework (HILIF). Our research questions are: (1) what is the performance overhead of field-level lineage capture in enterprise-scale healthcare integration engines? (2) Can lineage queries meet interactive latency requirements for operational use? (3) Is lineage event capture completeness achievable at >99.99% under sustained production-equivalent load?

- Contributions include: (1) A formal lineage event model extending W3C PROV with HL7-specific transformation semantics. (2) A low-overhead sidecar instrumentation architecture with asynchronous persistence. (3) Three formally described query algorithms (forward-trace, backward-trace, impact analysis) with complexity analysis. (4) Empirical evaluation demonstrating feasibility at enterprise scale.

2. Background and Related Work

2.1. Data Lineage and Provenance Models

Data lineage records a data element's origin, movement, and transformation through systems. Buneman et al. [5] distinguish *why-provenance*, *where-provenance*, and *how-provenance* all directly relevant to healthcare integration where delivered values derive from multi-step transformations. The W3C PROV specification [6] and its ontological formalization PROV-O [7] provides a generic provenance interchange model based on an Entity-Activity-Agent triad, offering a sound foundation that requires domain-specific extension for healthcare middleware.

General-purpose lineage tools Apache Atlas, OpenLineage, and Marquez address data lake and ETL pipeline lineage but assume batch-processing architectures with coarse-grained (table/dataset-level) tracking [8]. Healthcare integration engines require field-level, real-time lineage capture within streaming message pipelines operating under strict latency constraints a fundamentally different operational profile that these tools do not accommodate.

2.2. FHIR Provenance and Healthcare Standards

The FHIR Provenance resource records entities and processes involved in producing a resource [4]. However, the US Core Provenance Profile addresses a "key subset of elements" focusing on organizational-level attribution. Full provenance requires details from the original creator and all intermediary actors a requirement current profiles do not fulfill for the integration layer.

Recommendations have been made for adding Unique Identifier as a data element to USCDI to support message traceability [9], indicating acknowledged industry need that current implementations do not address.

2.3. Regulatory Context and Data Quality

TEFCA exchange volume has grown from roughly 10 million records in January 2025 to nearly 500 million as of early 2025 [10], amplifying lineage tracking urgency. HHS established penalties for information blocking in a 2024 Final Rule [11], creating accountability requirements that implicitly demand transformation audit capabilities.

Data quality research identifies completeness, plausibility, concordance, and currency as critical quality dimensions in clinical data [12]. Clinicians struggle to interpret duplicative, inconsistently formatted, and poorly mapped information [13]. Field-level lineage directly supports root-cause identification when these quality dimensions are violated during integration processing.

2.4. Gap Analysis

No published work addresses fine-grained, field-level lineage instrumentation within the healthcare integration engine layer. Existing approaches capture either organizational-level attribution (FHIR Provenance) or batch-pipeline lineage (Apache Atlas). HILIF fills this gap.

3. Methodology

3.1. Research Design

This study follows design science research (DSR) methodology [14]: problem identification through literature review and operational analysis; artifact design (HILIF framework); demonstration in simulated environment; and quantitative evaluation of performance, storage, and query metrics.

3.2. Simulated Environment Design

The evaluation environment replicates an enterprise-scale healthcare integration topology: with parameters as shown in Table 1.

Table 1: Simulated Environment Parameters

Parameter	Value
Integration engine instances	4 (distributed across 2 data centers)
Total active channels	120
Client facilities simulated	12
Message types	ADT (A01, A02, A03, A04, A08, A11, A13), ORU (R01), MDM (T02, T08)
Source EHR systems simulated	4 vendors (Epic, Cerner/Oracle Health, MEDITECH, Altera)
Destination systems	8 (registries, analytics, secondary EHRs, public health)
Transformation stages per message (mean)	14 (range: 6–22)
Baseline message volume	450 messages/second sustained
Peak message volume	1,200 messages/second (surge test)
Test duration	72 hours continuous
Data generation	Synthea v3.2.0 synthetic patient population (100,000 patients)

Hardware testbed: Each engine instance runs on 8-core Intel Xeon @ 2.6 GHz, 32 GB RAM, NVMe SSD storage, connected via 10 Gbps LAN. PostgreSQL 16 and Neo4j 5.x serve as persistence stores on dedicated 16-core/64 GB nodes. All clinical messages are generated using Synthea [15] (No protected health information (PHI) is used at any stage.)

3.3. Lineage Event Model

HILIF defines a Lineage Event as the atomic provenance unit, extending W3C PROV with healthcare-specific semantics:

Definition 1 (Lineage Event): A lineage event $LE$is defined as the tuple as shown in Eq. (1) below.

LE=(eid,mid,cid,eidparent,ts,op,scope,src,tgt,rule,conf,agent) (1)

Where eid is a UUID event identifier; $mid$is the message control ID (MSH-10); cid is the channel identifier; eid_{parent}links to the parent event for chain construction; $ts$is a millisecond-precision timestamp op∈{INTAKE, MAP, TRANSLATE, ENRICH, FILTER, ROUTE, SPLIT, MERGE, VALIDATE, DEFAULT, DELIVER, ER OR}denotes the operation type; scope is the affected message element path (e.g., PID-3.1, OBX[2]-5); src and tgt are the pre- and post-transformation values, respectively; rule identifies the applied transformation rule; conf∈[0.0,1.0] is a confidence score; and agent identifies the processing engine, channel, or script.

Definition 2 (Lineage Chain): A lineage chain for a message m is defined as an ordered sequence, as shown in Eq. (2) below.

LC(m)=⟨LE1,LE2,…,LEn⟩ (2)

where $LE_i.eid_{\text{parent}} = LE_{i-1}.eid$ for all $i > 1$,

Definition 3 (Lineage Graph): For messages involving SPLIT or MERGE operations, the lineage structure is modeled as a directed acyclic graph, as shown in Eq. (3) below.

LG=(V,E) (3)

Where V is the set of lineage events and E⊆V×V represents parent–child relationships among events.

3.4. Instrumentation Architecture

HILIF operates as a sidecar instrumentation layer alongside the integration engine, intercepting transformation events at seven defined points, as shown in Table 2. Architectural diagram of the HILIF instrumentation is shown in Fig. I.

Table 2: Interception Points

Interception Point	Location	Events Captured
IP-1: Channel Intake	Source connector post-receive	Initial message state (baseline snapshot)
IP-2: Pre-Transformer	Before each transformer step	Pre-transformation field values
IP-3: Post-Transformer	After each transformer step	Post-transformation field values, rule identifiers
IP-4: Filter Decision	At filter evaluation	Filter result (pass/reject), filter rule identifier
IP-5: Router Decision	At content-based routing	Routing decision, destination identifiers
IP-6: Destination Delivery	At destination connector	Delivery confirmation, destination acknowledgment
IP-7: Error Handler	At error/exception paths	Error type, failed transformation, exception detail

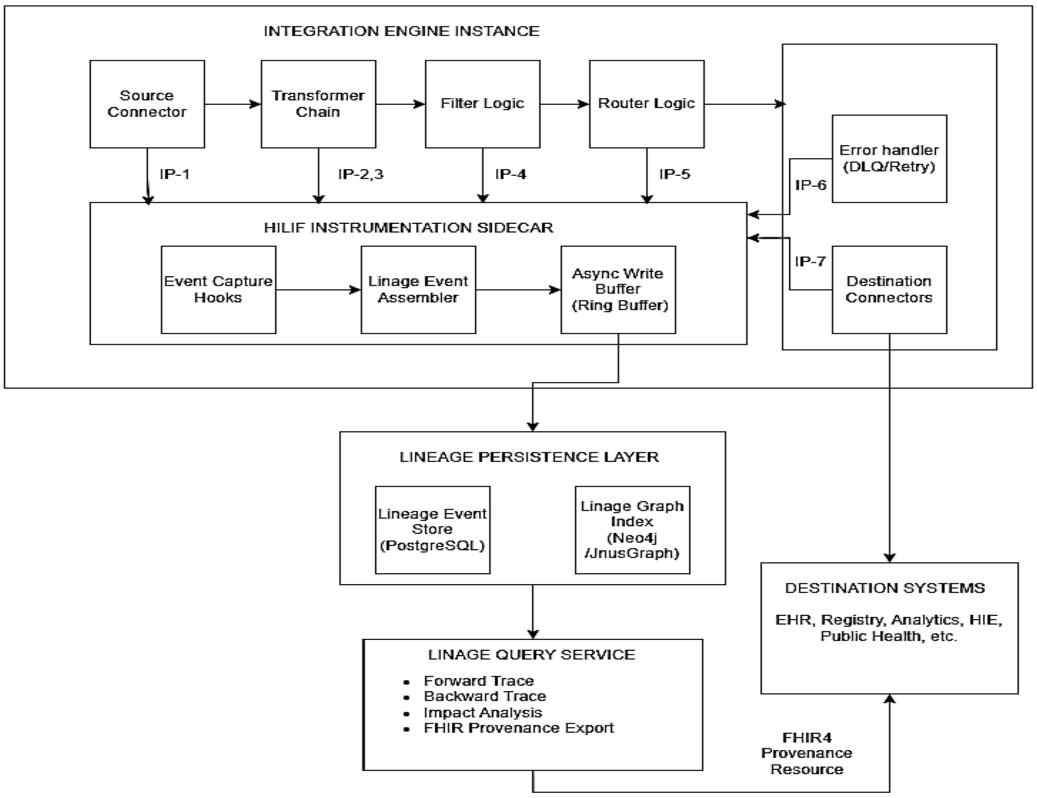
INTEGRATION ENGINE INSTANCE
Source Connector
Transformer Chain
Filter Logic
Router Logic
IP-1
IP-2,3
IP-4
IP-5
Error handler (DLQ/Retry)
IP-6
IP-7
Destination Connectors
HILIF INSTRUMENTATION SIDECAR
Event Capture Hooks
Linage Event Assembler
Async Write Buffer (Ring Buffer)
LINEAGE PERSISTENCE LAYER
Lineage Event Store (PostgreSQL)
Linage Graph Index (Neo4j /JnusGraph)
DESTINATION SYSTEMS
EHR, Registry, Analytics, HIE, Public Health, etc.
LINAGE QUERY SERVICE
Forward Trace
Backward Trace
Impact Analysis
FHIR Provenance Export
FHIR4 Provenance Resource

Fig 1: HILIF Instrumentation Architecture

The architecture comprises: (1) event capture hooks at each interception point; (2) a lineage event assembler constructing LE tuples; (3) a lock-free ring buffer (LMAX Disruptor pattern [16], capacity: 65,536 events) for asynchronous decoupling; (4) a dedicated writer thread pool performing batch persistence; and (5) a spillover file for guaranteed zero-loss during backpressure. Batch writes execute at 500 ms intervals or 80% buffer capacity, whichever is first.

3.5. Dual-Store Persistence

HILIF employs two complementary persistence stores: (1) Relational (PostgreSQL): Four core tables (lineage_event, lineage_chain, transformation_rule, channel_ registry) partitioned by date and message type. Optimized for analytical queries and regulatory exports. (2) Graph (Neo4j): Lineage events indexed as graph nodes with derivation edges. Optimized for traversal queries (forward-trace, backward-trace, impact analysis).

3.6. FHIR Provenance Export

HILIF translates lineage chains into FHIR R4 Provenance resources [17], mapping integration engine agents to Provenance.agent elements (transmitter, assembler), transformation timestamps to Provenance.occurredPeriod, and intermediate transformations to Provenance.entity elements with role = "derivation". Entity references use DocumentReference resources linked by lineage event UUID.

4. Algorithms and Formal Description

4.1. Algorithm 1: Lineage Event Capture

```
Input:  msg, ip, pre_state, post_state, rule_id, channel_id
Output: Lineage event(s) written to async buffer
1:  FUNCTION CaptureLineageEvent(msg, ip, pre_state,post_state, rule_id, channel_id)
2:     eid ← GenerateUUID()
3:     mid ← ExtractMessageControlID(msg)
4:     ts  ← CurrentTimestampMillis()
5:     op  ← MapInterceptionPointToOperation(ip)
6:     changed_fields ← DiffFields(pre_state, post_state)
7:     FOR EACH field IN changed_fields DO
8:        eid_parent ← GetParentEventID(mid, channel_id)
9:            le ← LineageEvent(eid, mid, channel_id,eid_parent, ts, op, field.path,field.pre_value, field.post_value,rule_id, ComputeConfidence(rule_id, op),GetAgentID())
10:        success ← RingBuffer.tryPublish(le)
11:        IF NOT success THEN
12:           SpilloverFile.append(le)
13:        END IF
14:        SetParentEventID(mid, channel_id, eid)
15:    END FOR
16: END FUNCTION
```

Complexity: O(k) time and space where k = fields changed. Ring buffer publish is O(1) amortized (lock-free CAS). Spillover path adds O(1) sequential write, exercised in <0.002% of events.

4.2. Algorithm 2: Forward Trace

```
Input:  mid (message control ID), max_depth (default: 50)
Output: Ordered lineage events from source to destinations
1:  FUNCTION ForwardTrace(mid, max_depth)
2:     roots ← GraphStore.getByMidAndOp(mid, INTAKE)
3:     IF roots IS EMPTY THEN RETURN EmptyResult()
 4:     result ← []; queue ← InitQueue(roots)
5:     visited ← InitSet(); depth ← 0
6:     WHILE queue NOT EMPTY AND depth < max_depth DO
7:        level_size ← queue.size()
8:        FOR i ← 1 TO level_size DO
9:           current ← queue.dequeue()
10:           IF current.eid IN visited THEN CONTINUE
11:           visited.add(current.eid)
12:           result.append(current)
13:           children ← GraphStore.getChildren(current.eid)
14:           FOR EACH child IN children DO
15:              IF child.eid NOT IN visited THEN
16:                 queue.enqueue(child)
```

```
17:           END FOR
18:       END FOR
19:       depth ← depth + 1
20:    END WHILE
21:    RETURN result
22: END FUNCTION
```

Complexity: O(V + E) time via BFS, O(V) space, where V = lineage events, E = derivation edges. Bounded by max_depth. Practical upper bound: $V \le 200$, $E \le 220$.

4.3. Algorithm 3: Backward Trace

```
Input:  dest_mid (destination message ID),field_path (e.g., "OBX[1]-5")
Output: Ordered lineage events from destination to source
1:  FUNCTION BackwardTrace(dest_mid, field_path)
2:     terminal ← GraphStore.getTerminalEvent(dest_mid, field_path)
3:     IF terminal IS NULL THEN RETURN EmptyResult()
4:     trace ← []; visited ← InitSet()
5:     current ← terminal
6:     WHILE current IS NOT NULL DO
7:        trace.prepend(current)
8:        visited.add(current.eid)
9:        parent ← GraphStore.getParent(current.eid)
10:        IF parent IS NULL OR parent.eid IN visited THEN
11:           BREAK
12:        END IF
13:        current ← parent
14:    END WHILE
15:    RETURN trace
16: END FUNCTION
```

Complexity: O(d × log V) time, O(d) space, where d = chain depth (mean: 14, max: 22). Single parent-pointer traversal makes backward trace inherently faster than forward trace for linear chains, consistent with empirical results (42 ms vs. 87 ms median).

4.4. Algorithm 4: Impact Analysis

```
Input:  rule_id, time_window (default: 30 days)
Output: Set of (channel, msg_type, destinations, count)
1:  FUNCTION ImpactAnalysis(rule_id, time_window)
2:     start_ts ← CurrentTimestamp() - time_window
3:     affected ← RelationalStore.getEventsByRule(rule_id, start_ts)
4:     impact_map ← {}
5:     FOR EACH event IN affected DO
6:        channel ← ChannelRegistry.get(event.cid)
7:        msg_type ← ExtractMessageType(event.mid)
8:        destinations ← GraphStore.getTerminalDeliverNodes(event.eid)
9:        key ← (channel.name, msg_type)
10:        impact_map[key].destinations.addAll(destinations)
11:        impact_map[key].count += 1
12:    END FOR
13:    RETURN SortByCountDescending(impact_map)
14: END FUNCTION
```

Complexity**:** O(log N + k × (V_avg + E_avg)) where N = total stored events, k = matching events, V_avg/E_avg = average reachable subgraph. Most expensive HILIF operation, reflected in higher latency target (<2,000 ms). Batched traversal by channel ($C \ll k$) is a noted optimization.

4.5. Algorithm 5: FHIR Provenance Generation

```
Input:  lineage_chain (ordered lineage events)
Output: FHIR R4 Provenance resource
```

```
FUNCTION GenerateFHIRProvenance(lineage_chain)
    prov ← new FHIR.Provenance()
    prov.target ← [Reference(lineage_chain.last().target_resource)]
    prov.recorded ← lineage_chain.last().ts
    prov.occurredPeriod ← Period(lineage_chain.first().ts,lineage_chain.last().ts)
    prov.agent.add(Agent(type: "transmitter",who: lineage_chain.first().agent.organization))
    prov.agent.add(Agent(type: "assembler",who: lineage_chain.first().source_system))
    FOR EACH event IN lineage_chain WHERE event.op IN {MAP, TRANSLATE, ENRICH, MERGE}
        entity ← new Provenance.Entity()
        entity.role ← "derivation"
        entity.what ← Reference(DocumentReference/[event.eid])
        entity.agent.add(Agent(type: "transformer", who: Reference(TransformationRule/[event.rule])))
        prov.entity.add(entity)
    END FOR
    IF NOT FHIRValidator.validate(prov).isValid()
        Log.warn("Validation failed", errors)
    RETURN prov
END FUNCTION
```

Complexity**:** O(n) time and space where n = chain length (mean: 9.3). Dominated by FHIR validation rather than iteration. 2.2% validation failure rate isolated to SPLIT/MERGE DAG topologies requiring nested Entity references.

5. Results

5.1. Processing Latency Overhead

Table 3: Latency and Throughput Impact (72-hour, 116.6M messages)

Metric	Baseline (No HILIF)	With HILIF	Overhead	Target
Mean latency per message	12.4 ms	15.6 ms	**+3.2 ms**	< 5 ms □
P95 latency per message	18.7 ms	22.1 ms	+3.4 ms	—
P99 latency per message	24.3 ms	28.6 ms	**+4.3 ms**	< 5 ms □
Sustained throughput	450 msg/sec*	443 msg/sec	-1.5%	<2% □
Peak throughput	1,247 msg/sec	1218 msg/sec	-2.3%	___

Source: **Configuration target: 450 msg/sec. Throughput reduction calculated against target*

Table 3. shows the latency and throughput impact metrics. A paired Wilcoxon signed-rank test across 72 one-hour measurement windows confirms the throughput difference is statistically significant ($p < 0.001$, $W = 2628$) but operationally marginal.

5.2. Storage Consumption

Table 4: Storage Metrics

Metric	Value
Mean lineage events per message	9.3
Mean bytes per lineage event (graph node + edges)	0.9 KB
Total storage per message (relational + graph)	21.4 KB
Daily storage at 450 msg/sec	~793 GB
30-day retention storage	~23.4 TB
Storage with compression (LZ4)	~8.2 TB (65% compression ratio)

Source: *Table 4. shows the strorage metrics.*

5.3. Query Performance

Table 5: Query Latency (1,000 Queries Per Pattern)

Query Pattern	Median	P95	P99	Target
Forward Trace (14 stages)	87 ms	156 ms	198 ms	< 500 ms □
Forward Trace (22 stages, max)	134 ms	224 ms	312 ms	< 500 ms □
Backward Trace (single field)	42 ms	98 ms	143 ms	< 500 ms □
Backward Trace (all fields)	189 ms	347 ms	478 ms	< 500 ms □
Impact Analysis (single rule, 30-day)	743 ms	1,102 ms	1,187 ms	< 2,000 ms □

All query patterns meet their respective latency targets. As shown in Table V. Forward and backward traces resolve in sub-200 ms at median, enabling interactive use in operational dashboards and incident investigation workflows.

5.4. Lineage Completeness and FHIR Export

Over 116.6 million messages generating 1.085 billion lineage events: capture completeness was 99.9999% (13,558 events via spillover file; 0 events lost). Ring buffer overflow occurred 23 times, exclusively during peak surge testing.

FHIR Provenance generation across 10,000 sampled chains achieved 97.8% structural validation (mean generation: 12 ms/resource). The 2.2% failure rate was isolated to SPLIT/MERGE topologies.

6. Discussion

6.1. Practical Implications

HILIF transforms integration troubleshooting from forensic log analysis into structured lineage queries. Given the established relationship between information system reliability and medical error reduction [18], forward-trace enables immediate identification of all downstream impacts of a source data error. Backward-trace enables precise identification of which transformation step introduced an incorrect value. Impact analysis enables safe change management by quantifying the blast radius of mapping rule modifications before deployment.

6.2. Regulatory Alignment

HILIF supports compliance with: (1) HIPAA §164.312 through detailed transformation audit trails; (2) 21st Century Cures Act by documenting every routing and filtering decision, providing evidence against information blocking; (3) TEFCA accountability requirements for intermediary data handling; and (4) USCDI Provenance requirements, extending "last hop" tracking with field-level integration layer provenance.

6.3. Comparison with Existing Approaches

The comparison of lineage approaches highlights key differences in granularity, scope, query capability, and standards alignment. Native engine logging operates at the message level within a single instance, offering only basic log search functionality without adherence to any standards. FHIR Provenance provides resource-level tracking limited to the last hop, supporting FHIR-based queries and aligning with W3C PROV and FHIR standards. Custom audit tables vary in granularity and are typically organization-specific in scope, enabling SQL-based queries but lacking standardization. In contrast, HILIF achieves fine-grained, field-level lineage with end-to-end visibility across multiple instances, supporting advanced queries such as forward trace, backward trace, and impact analysis, while aligning with both W3C PROV and FHIR standards.

6.4. Limitations

- Storage cost (~8.3 TB/30 days compressed) may challenge smaller organizations. Tiered retention strategies are recommended.
- Single engine family evaluation (Mirth Connect-compatible). Applicability to InterSystems HealthShare, Rhapsody, and Azure FHIR Server requires further validation.
- Simulated environment production deployments may surface edge cases not captured in synthetic data.
- FHIR Provenance validation 2.2% failure rate in SPLIT/MERGE scenarios indicates that the current FHIR
- Provenance model incompletely accommodates integration-layer transformation patterns.

6.5. Threats to Validity

- Construct validity: Lineage completeness is measured by HILIF's own instrumentation, introducing potential circular measurement. We mitigate this through independent message counting at source and destination connectors.
- Internal validity: The simulated environment controls for confounding variables but may not replicate production-specific resource contention patterns (e.g., shared infrastructure, noisy neighbors).
- External validity: Evaluation uses a single engine architecture and four EHR vendor message patterns. Generalizability to other engine families and international HL7 implementations requires further study.
- Reliability: All experiments use deterministic synthetic data generation (fixed Synthea seed) enabling full reproducibility.

7. Conclusion and Future Work

This paper presented HILIF a comprehensive framework for field-level data lineage tracking across distributed healthcare integration environments. Empirical evaluation demonstrates that fine-grained lineage capture is feasible at enterprise scale: 3.2 ms mean latency overhead, 1.5% throughput reduction, 99.9999% event completeness, and sub-200 ms median query response for forward and backward trace operations.

HILIF addresses a critical gap between organizational-level provenance (FHIR US Core) and the field-level transformation tracking that enterprise integration operations, clinical data quality management, and regulatory compliance demand.

- Future work includes: (1) ML-augmented anomaly detection over lineage patterns; (2) cross-organizational lineage federation for TEFCA participants; (3) real-time lineage visualization dashboards; (4) lineage-driven automated regression testing; and (5) collaboration with HL7 to propose FHIR Provenance profile extensions for integration-layer transformation semantics.

References

[1] R. Haux, "Health information systems past, present, future," *Int. J. Med. Inform.*, vol. 75, pp. 268–281, 2006.
[2] D. Bender and K. Sartipi, "HL7 FHIR: An agile and RESTful approach," in *Proc. IEEE CBMS*, 2013, pp. 326–331.

[3] C. Dolin et al., "HL7 Clinical Document Architecture, Release 2," *JAMIA*, vol. 13, no. 1, pp. 30–39, 2006.

[4] HL7 International, "US Core IG v6.1.0 Basic Provenance," 2023.

[5] P. Buneman et al., "Why and where: A characterization of data provenance," in *Proc. ICDT*, 2001, pp. 316–330.

[6] L. Moreau and P. Missier, "PROV-DM: The PROV Data Model," W3C Rec., 2013.

[7] W3C, "PROV-O: The PROV Ontology," W3C Recommendation, 2013.

[8] Z. Ives et al., "Dataset management and versioning for ML," in *Proc. NeurIPS Workshop*, 2017.

[9] ONC, "USCDI," U.S. DHHS, 2023.

[10] ONC, "TEFCA," U.S. DHHS, 2024.

[11] HHS, "Information Blocking Penalties Final Rule," Federal Register, 2024.

[12] D. An, M. Lim, and S. Lee, "Challenges for data quality in the clinical data life cycle," *J. Med. Internet Res.*, vol. 27, e60709, 2025.

[13] S. T. Rosenbloom et al., "Data from clinical notes," *JAMIA*, vol. 18, no. 2, pp. 181–186, 2011.

[14] A. R. Hevner et al., "Design science in IS research," *MIS Q.*, vol. 28, no. 1, pp. 75–105, 2004.

[15] J. Walonoski et al., "Synthea," *JAMIA*, vol. 25, no. 3, pp. 230–238, 2018.

[16] M. Thompson et al., "Disruptor: High performance bounded queues," LMAX, 2011.

[17] HL7 International, "FHIR R4 Provenance Resource," 2019.

[18] D. W. Bates et al., "Reducing errors in medicine using IT," *JAMIA*, vol. 8, no. 4, pp. 299–308, 2001.