International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(23s) 2026 pp. 1693-1706
https://doi.org/10.70917/ijcisim-2026-5722
Article

Deterministic Data Pipelines for Regulated Industries: Reproducibility Guarantees, Automated Validation at Ingestion, and Auditable Lineage Across Mission-Critical Healthcare Workflows

Sandeep Rao Udupi

Independent Researcher, India. Email:- sandeeprao003@gmail.com

Abstract: A data pipeline is deterministic when the same input data, transformation logic, configuration, and execution conditions always produce the same output. Ordinary businesses benefit from this property. Regulated ones require it, because a payment, a claim, a quality report, or a compliance decision may have to be reproduced and independently verified months or years after it was made, and an auditor asking how a published number was produced is entitled to an answer that does not depend on who happens to remember. This paper sets out what determinism costs to achieve in a healthcare payer environment and how it is delivered. It ranks the sources of non-determinism by the production trouble each causes, placing late-arriving and corrected data first, and derives from that ranking a single governing rule: every value capable of influencing the output must be an explicit, versioned input rather than an invisible dependency. It describes a metadata-driven source-to-target validation framework in which the objects, business keys, comparison columns, data types, null-handling rules, and validation criteria are configuration rather than code, so that the same reconciliation logic serves any pair of datasets with comparable records. It distinguishes reproducing a business result from recreating a byte-identical file, which are different guarantees with different costs. In one reconciliation cycle the framework identified more than five thousand record-level and field-level discrepancies between source and target before they could reach downstream payment processing and reporting. The paper reports that count as measured and states explicitly which other quantities were not measured.

Keywords: Deterministic data pipelines, reproducibility, data lineage, source-to-target reconciliation, metadata-driven validation, ingestion validation, audit trail, regulated data processing, healthcare payer systems, claims processing, data governance, exception handling, idempotent processing

1. INTRODUCTION

1.1 The obligation

In a regulated setting the output of a data pipeline is not only used, it is answered for. A provider payment, a claims reconciliation, a quality submission, or a compliance report may be examined long after the processing cycle that produced it, and the institution has to be able to demonstrate not merely what number was published but exactly how that number was produced and validated.

That obligation has a technical consequence that is easy to underestimate. It means a historical processing state has to be reconstructable, which means every input to that state has to have been preserved, and which in turn means the pipeline has to know what all of its inputs actually were. Most pipelines do not, because most pipelines have inputs their authors do not think of as inputs.

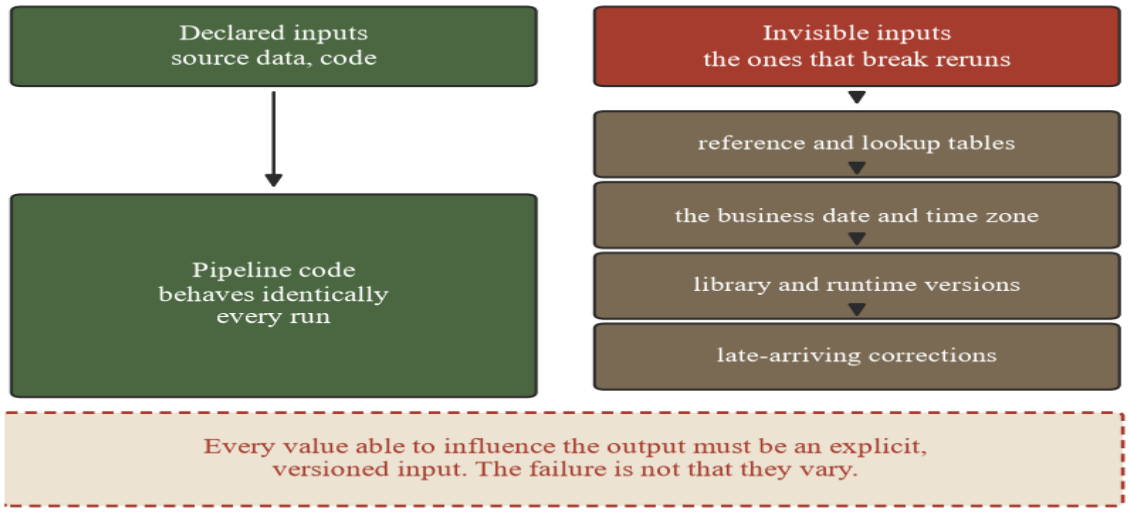

Figure 1. Where determinism is lost. The pipeline code behaves identically every run; what varies are the inputs nobody classified as inputs.

1.2 What determinism means here

A pipeline is deterministic when the same input data, transformation logic, configuration, and execution conditions always produce the same output. Achieving that requires controlled schemas, versioned business rules, stable transformation logic, reproducible execution, idempotent processing, consistent reference data, and auditable run metadata. A deterministic pipeline should additionally capture lineage showing where data originated, how it was transformed, which rules were applied, and what output resulted.

Ordinary businesses benefit from these properties. Regulated organisations face materially greater consequences from their absence: compliance violations, incorrect payments, financial exposure, audit failures, and loss of institutional trust. The difference is not that determinism is more useful in regulated industries. It is that the cost of its absence is borne differently.

1.3 Scope and contribution

The work described was carried out during 2024 and 2025 in a regulated healthcare payer environment, supporting provider, claims, fee, reconciliation, and payment-related data flows, in a data engineering and lead data engineering capacity covering pipeline architecture, ingestion, transformation, validation, reconciliation, exception handling, and production deployment.

This paper contributes:

1. A ranking of the sources of non-determinism by the production trouble each causes, and the control appropriate to each.

2. A metadata-driven source-to-target validation architecture in which validation logic is configuration rather than code, and which is therefore reusable across data domains rather than built per feed.

3. A distinction between reproducing a business result and recreating a byte-identical physical output, with the controls each requires.

4. A lineage model sufficient to trace a published number back through validation to its originating source record.

5. A report of a real reconciliation defect and the controls added in response.

What this paper does not do. It reports one measured quantity, the count of discrepancies identified in a reconciliation cycle, and is explicit about the rest. There is no verified reproducibility percentage, no documented count of production pipelines under management, and no audited count of findings opened and closed. Section 12 states the evidence position in full, including one figure that circulates alongside this work and is explicitly not a measurement.

On naming. The environment, the source system, and the target platform are described by function throughout. This follows the disclosure practice appropriate to protected healthcare data and does not affect any technical claim in the paper.

2. Background and Related Work

Reproducibility as an engineering property. The reproducibility problem is usually posed in scientific computing terms, where the concern is that a published analysis can be re-executed. The regulated data-processing case is stricter in one respect and looser in another: stricter because the re-execution may be demanded adversarially and years later, looser because the requirement is usually to reproduce a business result rather than a bitwise-identical artifact. Section 5 treats that distinction directly, because conflating the two produces either an unachievable requirement or an inadequate one.

Lineage and provenance. Data lineage systems record the derivation of a value through a processing graph. The literature concentrates on capturing lineage automatically from execution. The requirement in a regulated payer setting is narrower and heavier: not lineage for every value, but lineage sufficient to answer a specific question about a specific published number, retained for as long as the regulatory record is retained, and demonstrable to someone outside the engineering function.

Reconciliation rather than trust. The design position underlying Section 4 is that agreement between a source and a target is something to be demonstrated per cycle rather than assumed from the correctness of the transformation. This is the same reasoning that motivates explicitly reconciled activities in place of distributed transactions in large-scale systems: where two stores cannot be updated atomically, the correct response is to make the reconciliation a first-class process rather than to seek an atomicity that is not available.

Idempotency as the enabling property. Determinism and idempotency are frequently conflated and are different requirements. A deterministic pipeline produces the same output from the same inputs. An idempotent one produces the same state whether it runs once or three times. A regulated pipeline needs both, and it needs the second specifically because reruns are routine: a failed batch is reprocessed, a corrected file is redelivered, a checkpoint is resumed. Without idempotent writes, the act of recovering from a failure changes the result, which means the recovery procedure itself becomes a source of non-determinism.

Metadata-driven processing. Configuration-driven frameworks, in which behaviour is expressed as data rather than code, are established practice in data engineering. Their relevance here is specific: when validation criteria are configuration, they are also an auditable artifact with a version, which turns a governance requirement into a property of the architecture rather than an additional documentation burden.

The gap this paper addresses is the composition of these into a working control. Reproducibility literature explains why determinism matters, lineage literature explains how to record derivation, and metadata-driven design explains how to avoid rebuilding validation per feed. What is less developed is a concrete account of what breaks determinism in a production regulated pipeline, in what order, and what each fix costs.

3. Where Non-Determinism Enters

Sources of non-determinism are not equally troublesome, and treating them as a checklist obscures which ones actually cause production incidents. Table 1 ranks them by production risk.

Table 1. Sources of non-determinism, ranked by production risk.

Rank	Source	Why it causes trouble	Control
1	Late-arriving or corrected data	Most common reason a rerun differs. Technically a change in the input set rather than computational non-determinism	Fixed cutoffs, watermarks, batch identifiers, immutable source snapshots
2	Mutable external and reference data	Provider, eligibility, pricing, and reference tables change between runs. These are hidden inputs	Version or snapshot reference data; record the version used per run
3	Processing order and concurrency	Parallel jobs, duplicate events, and non-deterministic joins produce different outcomes	Stable business keys, deterministic ordering, idempotent writes, deduplication, transactional processing
4	Wall-clock time and time zones	Logic keyed to current timestamp or local "today" makes output depend on when it ran	Parameterised business date, UTC normalisation, persisted run timestamps
5	Library and runtime versions	Package, connector, engine, or runtime changes alter processing behaviour	Version-pinned dependencies, versioned code, controlled deployments, reproducible runtimes

The ranking carries more information than the list. The first two entries are not computational non-determinism at all: the pipeline is behaving identically and the inputs have changed. That is why they cause the most trouble. A team investigating a differing rerun looks first at the code, which is where the problem is not, and the actual cause is a reference table someone updated legitimately in the interim and which nobody regarded as an input.

This yields the governing rule for the whole design: every value capable of influencing the output must be treated as an explicit, versioned input rather than an invisible dependency. The word invisible is the operative one. Reference data, the business date, the runtime version, and the library set are all inputs in the sense that they determine the output, and the failure is not that they vary but that they vary without being recorded.

4. Metadata-Driven Validation at Ingestion

4.1 The design

The core of the implementation is a generic metadata-driven source-to-target validation framework. Metadata defines the source and target objects, the business keys, the comparison columns, the data types, the null-handling rules, and the validation criteria. At runtime the framework executes record-level and field-level reconciliation dynamically from that definition.

Because the validation logic is configuration-driven rather than hard-coded, the same framework serves any source and target datasets with comparable records, across claims, provider, payment, financial, and other regulated data domains. This is the property that makes the approach economic: validation coverage that would otherwise be built per feed, and therefore built inconsistently or not at all, becomes a matter of supplying a configuration.

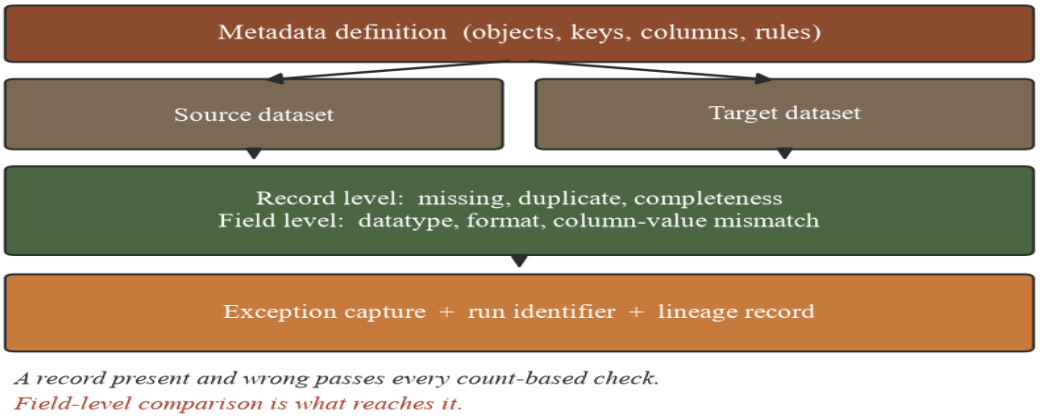

Figure 2. The metadata-driven validation framework. One definition drives record-level and field-level reconciliation for any pair of comparable datasets.

Table 2. What the framework validates at runtime.

Check	**What it detects**
Missing-record detection	Records present in source and absent in target, or the reverse
Duplicate detection	Business keys appearing more than once where uniqueness is expected
Completeness validation	Required fields absent or null where the rule forbids it
Datatype and format checks	Values that do not conform to the declared type or format
Column-level mismatch analysis	Field values differing between source and target for a matched key

Record-level checks establish whether the right rows arrived. Field-level checks establish whether the right values arrived in them, and it is the field-level layer that catches the defects most likely to reach a payment, because a record that is present and wrong passes every count-based reconciliation.

4.2 What the framework does with what it finds

Detection is only half of the control. Each identified discrepancy is captured as an exception carrying the business key, the rule that failed, the source and target values, and the run identifier, which makes the exception addressable rather than merely counted.

Two properties of that design matter. Exceptions are captured rather than blocking, so a reconciliation cycle produces a complete picture of every discrepancy rather than halting at the first, which is what allows a downstream team to triage by materiality instead of by discovery order. And an exception carries enough context to be investigated without returning to the source files, which is the difference between a finding someone can act on and a report that starts another search.

The alternative arrangement, where validation raises an alert and the investigation begins from the raw data, is common and produces a predictable failure: the volume of exceptions exceeds the capacity to investigate them, so thresholds get raised until the alerts are manageable, and the control quietly stops controlling.

4.3 Why configuration is also governance

Expressing validation criteria as configuration has a second consequence beyond reuse. Validation rules and source-to-target mappings become centrally defined and consistently applied rather than handled through ad hoc

checks distributed across jobs. That produces standardisation, accountability, traceability, and auditability as properties of the architecture rather than as documentation maintained alongside it.

The practical difference appears when someone asks which rules were applied to a specific historical run. Where rules are code, the answer requires reconstructing the deployed version. Where rules are versioned configuration recorded against the run, the answer is retrievable.

5. Reproducibility: Two Different Guarantees

A question that sounds like one question is really two, and the distinction matters because they cost different amounts to deliver.

Reproducing the business result means replaying the same source data with the same configuration and obtaining the same reconciliation outcome: the same missing records, the same field-level mismatches, the same verdicts. Because the framework is metadata-driven and deterministic, this holds when the original input batch, the source and target snapshots, the metadata configuration, the business keys, and the validation rules are preserved.

Recreating a byte-identical physical output is a stronger requirement and needs additional controls, because processing timestamps, row ordering, file formatting, compression, and runtime versions can all change the physical file while the business result is unchanged.

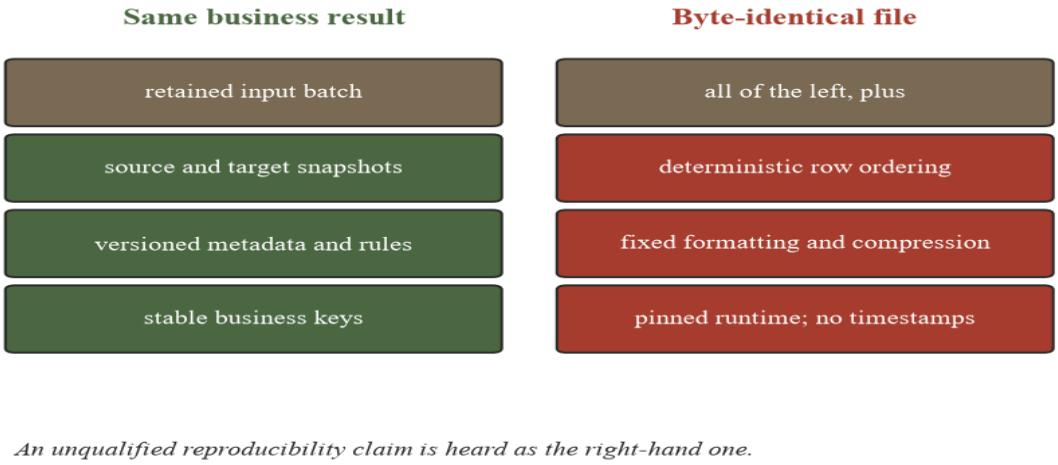

Figure 3. Two different reproducibility guarantees. Stating which is offered matters, because an unqualified claim is heard as the stronger one.

Table 3. What each guarantee requires.

Guarantee	Requires	Typical obstacle
Same reconciliation result	Retained input batch, source and target snapshots, versioned metadata and rules, business keys	Reference data that changed between runs
Byte-identical output file	All of the above, plus deterministic row ordering, fixed formatting and compression, pinned runtime, and excluded or parameterised timestamps	Processing timestamps embedded in output

For a regulated pipeline the first guarantee is usually the one that matters, because the question an auditor asks is whether the result can be reproduced and independently verified, not whether two files hash identically. Stating which guarantee is being offered is nonetheless important, because an unqualified claim of reproducibility will be read as the stronger one.

The controls that deliver the first guarantee are the retention set: historical input batches, source and target snapshots, versioned validation metadata, processing run identifiers, lineage, and reconciliation results. Together these allow a prior processing state to be reconstructed and independently verified.

6. What a Replay Actually Requires

Section 5 states the retention set abstractly. It is worth being concrete about what has to exist at the moment someone asks for a historical result to be reproduced, because the gap between intending to be reproducible and being reproducible is usually a retention decision made years earlier by someone optimising storage.

Table 4. The replay set, and what its absence costs.

Artifact	Why it is needed	If absent
Original input batch, immutable	The primary input. Reprocessing current data answers a different question	Replay is impossible; the result cannot be reproduced at all
Source and target snapshots as at the run	The comparison the reconciliation performed	The reconciliation can rerun and will find different discrepancies
Reference data version used	The hidden input ranked second in Table 1	Output differs for reasons nobody can locate
Metadata and rule version	What was actually checked	The replay applies today's rules to yesterday's data, which is a different test
Calibration version	Thresholds in force at the time	Results are comparable in form and not in meaning
Business date parameter	Removes dependence on execution time	Time-dependent logic silently reinterprets the run
Run identifier and audit log	Binds the artifacts together as one execution	The pieces exist and cannot be assembled into a specific run
Reconciliation results as produced	The thing being reproduced	No baseline to compare the replay against

Two observations follow from the table.

Every row is a retention decision rather than an engineering one. The pipeline that produced the result is generally still capable of producing it again. What fails is that one of these artifacts was not kept, or was kept without the association that binds it to a specific run. Reproducibility is therefore mostly a records problem wearing an engineering costume.

The last row is the one most often missing. Organisations retain inputs and code and neglect to retain the output they are trying to reproduce, which means a replay produces a result with nothing to check it against. A replay that cannot be compared demonstrates that the pipeline runs, not that it reproduces.

7. Lineage Sufficient for an Auditor

Lineage is captured from the incoming source file through validation and reconciliation to downstream reporting. The test of whether it is sufficient is concrete: if a value in a published report appears wrong, can it be traced back to the record that produced it?

For each processed record the framework retains the source batch, the business key, the validation rule applied, the source value, the target value, the mismatch status, and the processing run identifier. The evidence path is therefore:

source file → source record → validation rule → reconciliation result → target record → downstream payment or reporting output.

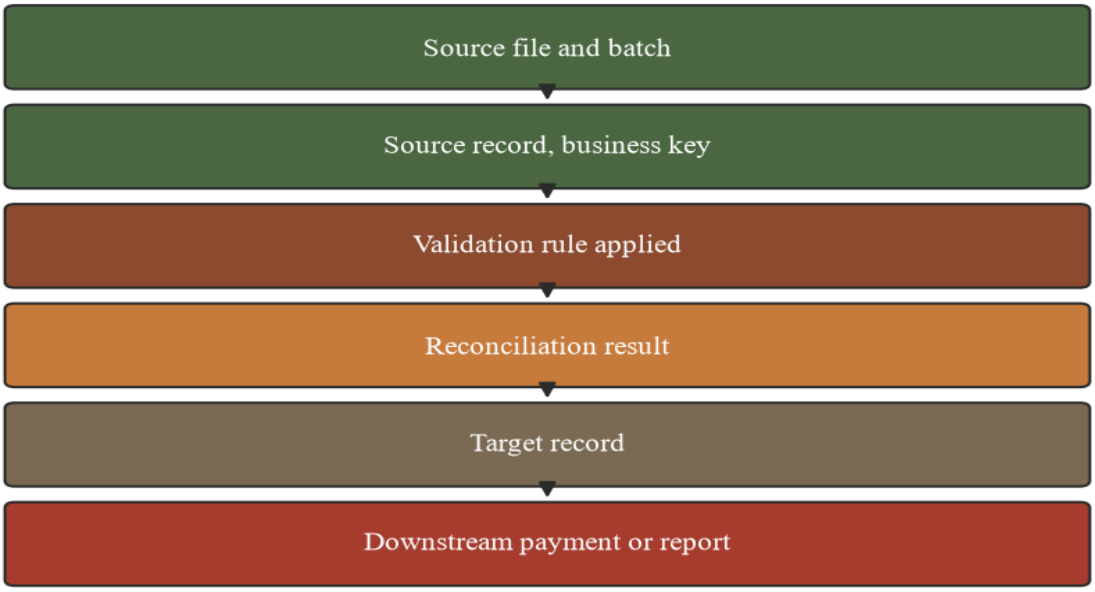

Figure 4. The evidence path. An auditor's question about a published number runs up this chain to the originating source record.

Table 5. Lineage attributes retained per record.

Attribute	**Answers**
Source file and batch identifier	Which delivery did this come from
Business key	Which entity does it concern
Source value	What the originating system said
Validation rule applied	What was checked, under which version
Target value	What the receiving system holds
Mismatch status	Whether they agreed
Processing run identifier	Which execution produced this determination

This level of traceability is what allows an organisation to explain not only what number was published but how it was produced and validated. It also determines whether a discrepancy can be isolated at all: with lineage, the divergence point is queryable; without it, the investigation is a manual search through files.

8. Governance and Change Control

Validation in a regulated environment rests on data governance rather than on testing alone, and the evidence it produces is the point rather than a byproduct.

For each run the framework retains the source batch, business key, validation rule, source value, target value, mismatch status, and processing run identifier. This produces lineage showing where a record originated, how it was evaluated, and where an exception occurred.

From a governance perspective, the significant property is that validation rules and source-to-target mappings are centrally defined and consistently applied rather than implemented as ad hoc checks. The output is therefore not a pass or fail verdict but an evidence trail demonstrating what data was processed, which rules were applied, where discrepancies were found, and how the final downstream result was derived.

The distinction between a verdict and an evidence trail is worth drawing out, because it determines what happens in an audit. A verdict asserts that the data was correct. An evidence trail allows someone else to check. Only the second survives an examiner who is not willing to take the assertion on trust, and in a regulated setting that is the only kind of examiner worth designing for.

9. Exception Handling as a Control

A reconciliation framework that finds discrepancies and does not resolve them has moved the problem rather than solved it. Three properties determine whether exception handling functions as a control or as a queue nobody works.

Exceptions must be classified, not merely counted. A cycle producing several thousand discrepancies is not actionable as a number. It becomes actionable when the discrepancies are grouped by the rule that failed, because a rule failing across thousands of records usually indicates one upstream cause rather than thousands of independent defects. Classification by failing rule converts a volume problem into a small number of investigations.

Exceptions must be attributable to a run. Every exception carries the processing run identifier, which means a discrepancy discovered later can be placed in time and associated with the configuration and calibration in force when it was produced. Without that association an exception is a fact about the data with no context, and establishing whether it was already known requires re-deriving history.

Exception disposition must itself be recorded. An exception that is investigated and found immaterial has been dispositioned, and that disposition is part of the evidence trail. An exception that is silently dropped between cycles leaves no trace of the judgement that dropped it. In a regulated setting the second case is the one that fails an audit, because the question is rarely whether a discrepancy existed and usually whether the organisation knew and what it decided.

The failure mode this guards against. The predictable degradation of a reconciliation programme is that exception volume exceeds investigation capacity, thresholds are raised until the volume is manageable, and the control continues to report healthy while detecting progressively less. The defence is not a better threshold but making the threshold itself a recorded, versioned decision, so that a raised threshold is visible as a change rather than absorbed as an operational adjustment.

This is the same principle that governs the reduction parameters in configuration and the validation rules in metadata: the judgement is not eliminated, it is made inspectable. A materiality threshold held in code or in someone's practice is a policy nobody approved.

10. A Real Reconciliation Defect

Provider and claim-related records were successfully ingested from the source system but did not reconcile correctly against the target claims platform. The metadata-driven validation framework identified more than five thousand mismatched records, including field-level differences that could have propagated into payment processing and reporting.

The defect was traced using lineage and reconciliation metadata along the path described in Section 6: source file and batch, then business key, then source value, then validation rule, then target value, then mismatch result. This isolated exactly where the data diverged rather than requiring a manual search through entire files, which is the practical value of the lineage design and the reason it is treated as a control rather than as documentation.

Four controls were strengthened afterwards: metadata-driven field validation, exception capture, run-level traceability, and source-to-target reconciliation as explicit named controls rather than as implicit properties of the pipeline. The effect was to move defect detection from downstream troubleshooting to upstream data-quality governance.

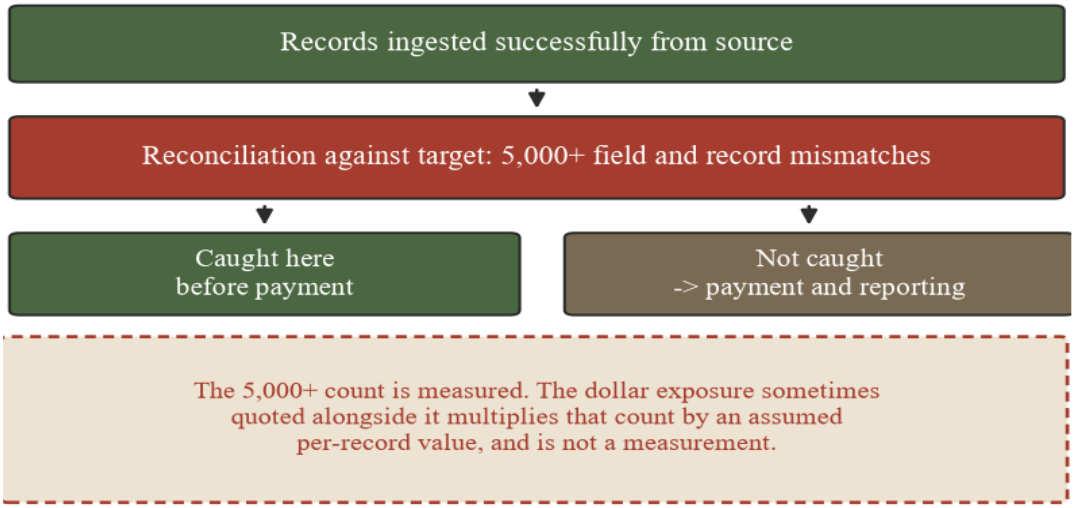

Figure 5. The defect and the claim it supports. The record count is measured; the exposure figure derived from it is not.

On the financial exposure sometimes quoted alongside this finding. A figure of roughly five million dollars in potential payment exposure has been associated with this defect. That figure is arithmetic on an assumed value: it multiplies the discrepancy count by an illustrative average payment of approximately one thousand dollars per affected record. The record count is measured. The per-record payment value is an assumption, and therefore so is the product. It is stated here because the order of magnitude is informative about why field-level reconciliation is worth its cost, and it should not be cited as a measured loss, an incurred cost, or a confirmed exposure. Actual exposure would require the real payment values for the affected records, which were not compiled.

11. What the Framework Measures Per Cycle

At the operational level the framework measures, for each processing cycle, source record counts, target record counts, matched records, unmatched records, field-level exceptions, duplicate records, validation status, batch and run identifiers, and reconciliation totals. This quantifies how much data passed validation and how much required investigation.

These are counts rather than rates, and the distinction matters for what can be claimed. A count of exceptions in a cycle is a fact about that cycle. Converting it into a defect rate, a reproducibility percentage, or a trend requires a consistent denominator and a retained history, and Section 12 records that those were not maintained in a form that would support such figures.

12. Why This Generalises Beyond Healthcare Payers

The implementation described sits in a healthcare payer environment, and the obligations that motivate it are not specific to healthcare. It is worth being explicit about which parts transfer and which are contingent, because the contingent parts are easy to mistake for the general ones.

What transfers is the obligation structure. Any domain where an output may have to be reproduced and independently verified long after production faces the same requirement, and the list is longer than it first appears:

financial reporting, prudential regulation, clinical trial data, environmental disclosure, and safety certification all impose it. What these share is not a subject matter but a temporal property, namely that the correctness of a historical result may be examined after the system that produced it has changed.

What transfers is the ranking. The ordering in Table 1 reflects properties of distributed data processing rather than of healthcare. Late-arriving and corrected data ranks first in any domain where sources deliver asynchronously and issue corrections, which is most of them. Mutable reference data ranks second wherever business rules depend on tables somebody else maintains. Neither observation depends on what the data describes.

What transfers is the metadata-driven approach, with one qualification. Configuration-driven validation is economic wherever the same reconciliation pattern recurs across many source-target pairs. In a domain with three feeds it is over-engineering, and the payoff arrives only once the number of pairs is large enough that per-feed implementation becomes the bottleneck.

What is contingent is the sensitivity handling. The disclosure discipline described here is shaped by protected health information, and a domain without an equivalent category would not need the same constraints on what may appear in an exception record or a lineage trace. That constraint is real and it is domain-specific, and it interacts with the design in one non-obvious way: a lineage record rich enough to satisfy an auditor may be rich enough to constitute a disclosure, which means retention and access control on the evidence trail are themselves regulated activities rather than infrastructure decisions.

What is contingent is the reconciliation granularity. Field-level comparison across every column is affordable at the record volumes described. A domain operating at several orders of magnitude more volume would need to sample or to prioritise columns by materiality, and the paper offers no guidance on doing so without losing the property that makes field-level reconciliation valuable, which is that it catches the record that is present and wrong.

The honest summary is that the obligations and the failure modes generalise, the economics of the specific approach depend on scale, and the handling of sensitive content does not transfer at all.

13. Evidence and Its Limits

Table 6. Evidence status by claim class.

Claim class	**Status**
More than five thousand source-to-target mismatches identified in one reconciliation cycle	**Measured.** Output of the framework in production
Defect traced through lineage to the divergence point	Established. The trace was performed
Four controls strengthened in response	Established
Per-cycle operational counts in Section 11	Measured per cycle. Not retained as a consistent series
Metadata-driven framework reusable across data domains	Design property, exercised in the implementation
Non-determinism ranking in Table 1	Practitioner judgement informed by production experience. Not an incidence count
Reproducibility of the business result	Design position with the controls stated. No verified reproducibility percentage exists
Approximately five million dollars in payment exposure	**Not measured.** Arithmetic on an assumed per-record value. See Section 9
Count of production pipelines under management	Not documented
Audited count of findings opened and closed	Not documented

Three of these deserve emphasis.

The discrepancy count is the paper's one solid measured result, and it is a strong one, because it is exactly the quantity the framework exists to produce and it was produced before the records reached downstream processing rather than after.

The exposure figure is not a result and is marked as such in two places deliberately. It is the kind of number that travels well and gets repeated, and the distance between "five thousand records were caught" and "five million dollars was saved" is one unverified assumption.

The absence of a reproducibility percentage is worth stating plainly rather than leaving as an omission. The framework is designed for reproducibility and its determinism was relied upon in practice, but no systematic replay campaign was run to establish the rate at which historical runs actually reproduce. That is the measurement this paper most obviously lacks.

14. What an Implementing Team Should Do First

The architecture assumes several things are already in place, and a team building it faces an ordering problem the design does not state.

Establish the input inventory before building validation. Section 3's governing rule is that every value able to influence the output must be an explicit versioned input. Applying it requires first knowing what those values are, and the enumeration is harder than it sounds, because the entries that matter most are the ones nobody currently classifies as inputs. Reference tables, calibration data, the business date, and the runtime set are the usual omissions.

Snapshot before reconciling. Reconciliation compares a source against a target at a point in time. If neither is snapshotted, the comparison drifts while it runs and a discrepancy may be an artifact of timing rather than a defect. Snapshotting is a retention cost and it is prior to the validation being meaningful.

Decide the retention policy with the auditor's question in mind. Section 6 lists the replay set. Each entry is a retention decision, and those decisions are usually made on storage-cost grounds by people who will not be asked to reproduce a result three years later. Making the retention policy explicit, and stating what it makes impossible, is more useful than a longer retention period chosen arbitrarily.

Start with one source-target pair and generalise deliberately. The metadata-driven approach earns its cost across many pairs. Building it for the first pair costs more than a bespoke check would, and the payoff arrives at the point where the pattern recurs. Teams that build the general framework for a single feed have over-engineered; teams that build bespoke checks for twenty feeds have under-engineered, and the crossover is worth estimating rather than assuming.

15. LIMITATIONS

One measured quantity. The discrepancy count is real. Almost everything else in the paper is design position or practitioner judgement, and the paper should be read accordingly.

The framework validates agreement, not correctness. Source-to-target reconciliation establishes that two systems hold the same values. It does not establish that either holds the right value. A defect present identically in source and target passes every check described here, and the paper's controls are silent on that class entirely. This is the correct scope for a reconciliation framework and it is a real boundary on what the evidence trail proves.

Determinism is assessed at pipeline level, not end to end. The controls in Section 3 cover the pipeline the author built. A published number typically passes through systems upstream and downstream of it, each with its own determinism properties, and reproducing a result end to end requires all of them to hold. The paper establishes one link of that chain.

No comparative evaluation. The metadata-driven approach is not compared against per-feed hand-coded validation on cost, coverage, or defect detection. The reuse argument is structural rather than measured.

Reproducibility is designed for, not demonstrated. As Section 12 records, no replay campaign was conducted. The controls in Section 5 are the right controls; whether they deliver reproduction at a high rate across historical runs was not established.

The non-determinism ranking is judgement. Table 1 orders sources by production trouble based on experience, not on a catalogued incident distribution. A different environment, particularly one with heavier concurrency, would plausibly rank processing order higher.

Exception triage is not addressed. Section 4.2 argues that captured exceptions must be addressable. What is not covered is how an organisation prioritises among thousands of them, what constitutes a materiality threshold, or who owns the decision that a given discrepancy class need not be investigated. That is where a reconciliation programme most often degrades in practice, and this paper does not treat it.

Single-environment basis. The work comes from one healthcare payer environment. Whether the ranking and the framework generalise to other regulated domains is argued from the generality of the obligation rather than demonstrated across settings.

The exposure figure will outlive its caveat. This is a limitation of publication rather than of the work. Section 9 marks the five million dollar figure as arithmetic on an assumption, and figures of that kind are routinely repeated without their qualifications. The honest position is that including it at all carries that risk.

16. Future Work

Run a replay campaign. Selecting a sample of historical processing cycles, re-executing them against retained inputs and configuration, and measuring the rate at which the business result reproduces would convert the paper's central design claim into a measured one. It is the most valuable missing piece and it requires no new engineering, only the discipline to run it.

Measure the reuse claim. Recording the effort to onboard each additional source-to-target pair would establish whether configuration-driven validation actually reduces the marginal cost of coverage, and by how much.

Establish exception baselines. Converting the per-cycle counts of Section 11 into a retained series would allow a cycle's exception profile to be compared against its own history, which turns reconciliation output into a drift signal rather than only a gate.

Compile the real exposure. Attaching actual payment values to a caught discrepancy set would replace the assumed per-record figure with a measured one, and would let the framework's value be stated in terms nobody has to caveat.

A note on sequencing. The replay campaign should come first and does not depend on the others. It uses artifacts already retained, it requires no new engineering, and its result determines how much confidence the rest of the paper's claims deserve. If historical runs do not reproduce at a high rate, the reuse and baseline work is premature, because the framework's determinism is the property everything else rests on.

The exposure compilation is last for a different reason. It requires access to payment values for the affected records, which is an approval question rather than an engineering one, and pursuing it before the reproducibility result is available risks producing a compelling number attached to a claim that has not been established.

17. CONCLUSION

Determinism in a regulated pipeline is not a property of the transformation code. It is a property of the completeness with which the pipeline's inputs are known, recorded, and retained, and the most common way it is lost is that something which determines the output was never regarded as an input at all.

The paper's central rule follows directly: every value capable of influencing the output must be an explicit, versioned input. Late-arriving data and mutable reference tables sit at the top of the trouble ranking precisely because they violate that rule while the code behaves impeccably, which sends investigations to the wrong place.

Two design positions carry the rest. Validation expressed as metadata rather than code is reusable across data domains and is simultaneously an auditable artifact, which turns a governance requirement into a property of the architecture. And reproducibility should be stated as the specific guarantee being offered, because reproducing a business result and recreating a byte-identical file are different promises with different costs, and an unqualified claim will be heard as the stronger one.

The framework caught more than five thousand source-to-target discrepancies in a single reconciliation cycle before they reached payment processing. That count is the paper's measured result. The financial exposure sometimes quoted alongside it is arithmetic on an assumed per-record value and is not a measurement, and the difference between those two statements is exactly the kind of distinction that a paper about auditable pipelines has an obligation to get right about itself.

REFERENCES

1. Health Level Seven International, HL7 FHIR Release: Standard for Health Care Data Exchange, HL7 International, Ann Arbor, MI, USA.
2. United States Congress, Health Insurance Portability and Accountability Act of 1996, Public Law 104-191.
3. United States Department of Health and Human Services, Standards for Privacy of Individually Identifiable Health Information, 45 CFR Parts 160 and 164.
4. P. Helland, "Life Beyond Distributed Transactions," ACM Queue, vol. 14, no. 5, pp. 69-98, 2016. doi: 10.1145/3012426.3025012.
5. National Committee for Quality Assurance, HEDIS Measurement Year Technical Specifications for Health Plans, NCQA, Washington, DC, USA.