

Original Article

A Comprehensive Framework for End-to-End Data Lineage, Anomaly Detection, and Quality Validation in Enterprise-Scale Distributed Data Processing Pipelines

Sudha DonepudiIndependent Researcher, USA.***Abstract:**

Enterprise data pipelines face challenges from schema changes, distributional drift, and limited coordination between data lineage, anomaly detection, and data-quality validation. This study reviews 23 provenance resources and proposes a layered reference architecture integrating these capabilities through a shared lineage graph. The framework introduces a formal lineage model, lineage-aware anomaly detection, a six-dimensional quality score, and root-cause and impact analysis. Although the framework has not been implemented or experimentally evaluated, existing literature provides supporting performance results while highlighting limited evidence of integrated evaluation. The proposed architecture addresses this gap by enabling unified monitoring, improved defect identification, and downstream impact assessment.

Keywords:

Data Lineage, Anomaly Detection, Data Quality, Distributed Pipelines, Root-Cause Analysis, Reproducibility.

Article History:**Received: 09.02.2025****Revised: 10.03.2025****Accepted: 23.03.2025****Published: 30.03.2025**

1. Introduction

Modern enterprises operate thousands of recurring data pipelines that refresh machine-learning features and business intelligence dashboards at intervals ranging from minutes to days [2]. Each execution fetches data from upstream tables, processes it on a cluster, and pushes the processed data down to downstream datasets which are used by other jobs. Distributed processing frameworks such as MapReduce [21], Apache Spark [11], [17], and Apache Flink [19] make large-scale data processing feasible by abstracting details of partitioning, scheduling, and fault recovery from application developers. Although these abstractions enable large-scale processing, they can complicate fault diagnosis. For example, a dashboard may display implausible values even when the execution engine reports successful completion of all tasks

The most common failure mode in production is thus not crash, but silent data-quality (DQ) failure. A column is deleted due to an upstream schema change, a join key gets trailing white space, a bug with a partition causes a duplication of yesterday's partition, a currency field updates its scale. According to Schelter et al. every decision down the stream is affected by the lack of or incorrect information, and verifying at scale requires a declarative approach, otherwise it is not economically viable [8]. Tu et al. measure the impact for recurring pipelines: Defects are propagated upstream as cascading failures, and the engineers have to invest significant time in manual monitoring as part of DataOps and MLOps practices [2].

There have been three research communities in response, mostly in isolation. The provenance community provides examples for the derivation of data products ranging from data stored at the file level, where provenance is captured at the file system level [13]



to workflow provenance [14] and lineage retrieval models for scientific data processing [16, 15]. The anomaly-detection community develops detectors for streaming metrics, from spectral-residual analysis using a CNN discriminator in a production service [1] to deep relational models over interleaved logs [5] on streaming benchmarks instead of static classification benchmarks [6]. The data-quality community develops constraint languages, profiling, and repair: holistic repair by probabilistic inference [12], large-scale verification [8], repair by history using automated constraint programming [2], quality-anomaly detection over quality dimensions, rather than just raw outliers [3] and systematic comparisons of what error-detection tools can actually detect [22].

This requirement is taken as a starting point and four contributions are made in this article. First, it surveys the twenty-three resources provided and assigns each a relevance score and documents the limitations (Section II, Table 1). Secondly, it normalizes a lineage model where edges are associated to job executions and not with the nominal pipeline topology, allowing dating a dependency and locating a defect in time (Section VI). Third, it defines a detection rule which is lineage-aware: one that assigns a statistical score and couples it to the consumer exposure of the node which generated it (Sections VII, VIII). Fourth, it establishes an evaluation process that maintains the measurements made and the proposed targets in separate tables, thus preventing any hypothesis in this article from being confused with a measurement (Section IX, Table 7). The scope is deliberately 2023 and earlier for the state of knowledge. The framework itself is a design idea: nothing in Sections V–VIII has been implemented or benchmarked for this article. The manuscript explicitly distinguishes literature-derived measurements from proposed evaluation targets and unvalidated design assumptions.

2. Related Work and Literature Review

2.1. Reference Audit and Ranking

All 23 provided documents were read and the bibliographic fields were compared to the document and, when such a field was present, the publisher record. The result is reported in Table 1. The two documents that needed correction are the KDD '19 anomaly-detection service paper [1] and the Simmhan, Plale and Gannon's provenance survey [15] where identification is based on OCR of the first page instead of on embedded metadata. There are no dates later than 2023 in the documents in the set. One document is the Eastasouth journal article on data-contract failure prediction [4], and is published in a venue with limited indexing; while it is retained as a moderately relevant source, the numbers used are only to provide context, never as a benchmark.

Table 1. Reference Evaluation and Architectural Relevance Classification

#	Source (short form)	Principal limitation
1	Ren et al., time-series anomaly detection service	Metric series only; no lineage, no attribution; results vary strongly by dataset periodicity
2	Tu et al., auto-validate by history (AVH)	Constraints on columns; no cross-pipeline dependency model; needs 60-execution history
3	Widad et al., quality anomaly detection	Binary classification over engineered quality features; 99.91% accuracy is task-specific
23	Zipperle et al., provenance-based IDS survey	Host-level, not pipeline-level; names benchmark scarcity as open
8	Schelter et al., automating DQ verification (Deequ)	Declarative constraints require manual authoring; no lineage coupling
12	Rekatsinas et al., HoloClean	Cell-level repair on benchmark tables; not a streaming monitor
6	Lavin & Ahmad, NAB benchmark	Benchmark, not a system; sparse labels; window-based scoring
5	Ahmed et al., ADAL-NN	Logs only; requires stable templates; custom log set limits reproducibility
22	Abedjan et al., detecting data errors	Empirical comparison; no architecture proposed
16	Bose & Frew, lineage retrieval survey	Pre-cluster; metamodel, no execution-time attribution
15	Simmhan et al., provenance in e-Science	Scientific workflows; supplied copy has no text layer
14	Davidson & Freire, provenance & workflows	Tutorial framing; no detection or validation component

13	Muniswamy-Reddy et al., PASS	File-system granularity; overhead on write path
11	Zaharia et al., resilient distributed datasets	Lineage for recovery, not for audit; coarse-grained only
17	Zaharia et al., Apache Spark (CACM)	Systems overview; no quality or detection layer
19	Carbone et al., Apache Flink	Logical vs physical dataflow distinction is the transferable part
20	Kreps et al., Kafka	Ingestion substrate; offset semantics only
21	Dean & Ghemawat, MapReduce	Batch model and task counters; no lineage object
18	Armbrust et al., Lakehouse	Storage architecture; lineage discussed only as motivation
9	Franklin et al., dataspace	Position paper; licenses loose schemas but gives no mechanism
10	Ehrlinger & Wöß, DQ tools survey	Tool survey; notes research-practice mismatch in dimensions
7	Polyzotis et al., data management for ML	Tutorial slides; problem framing without a system
4	Chirumamilla, data-contract failure prediction	Limited indexing; 79% accuracy / 74% F1; context only

The ranking is a judgement, not a measurement, and it is defined as follows: 90–100 for sources that directly define a component of the joint problem; 70–89 for sources that define a component but not the coupling; 50–69 for enabling infrastructure; below 50 for background. The production detection service [1] and the automated constraint programming work [2] provide both an architecture and measured behaviour; the prose-based intrusion-detection survey [23] is the most pervasive example in the set of a provenance graph being a detection substrate it is not just an audit artifact; and the Deequ system [8] provides a declarative constraint model that the quality layer needs to expose.

2.2. Provenance and Lineage

This framework takes two design decisions from the provenance literature. First, the metadata describing the state of a system is not the same kind as the data being described, and must be collected in proximity to the system that generates it. Muniswamy-Reddy et al. make the strongest form of this argument: by collecting provenance inside the storage system, collection becomes transparent, users take no special action, and provenance is not lost during backup or migration [13]. The second is that a lineage system must have a clear metamodel before it can be compared to another. Bose and Frew survey lineage retrieval and propose a basic metamodel to identify the basic components of systems that support lineage for scientific data products [16]; Simmhan et al. construct taxonomy of provenance characteristics and apply it to current approaches [15]. Both are pre-2010, and both are the standard vocabulary: the difference with them is in the granularity of the lineage captured in the cluster, not the concepts.

In the Davidson and Freire's workflow view, the provenance includes not only information about the data the workflow processes, but also the description of the workflow itself, and this view of provenance must allow the answer to such questions as who created a product, what process produced it, and whether two products share a source without manual bookkeeping [14]. The distribution question above is just one that affects their framing; their mapping is approximately the same as the operational questions in Section I, except that with Spark or Flink, the process specification in the job is a logical plan, and the executed plan is a physical plan [19,17]. A lineage graph based upon the logical plan alone can't differentiate between defect caused by a code change and a defect caused by a data-dependent shuffle.

Franklin, Halevy and Maier provide the rationale behind the lack of a single canonical schema as a prerequisite for integration [9]. Their dataspace argument, that differing sources can be managed usefully without unification licenses a pragmatic lineage design in which types of edges are loosely typed, and identifiers are progressively resolved. It also says that the primary risk to such a design is that without a schema there will be a governance challenge, not a technical one, of agreeing on what is the same dataset.

The most directly transferable outcome in the set is the result of the intrusion detection survey [23] based on the concept of provenance. It records that system-level provenance, which is a directed acyclic graph of information flows between entities, is a viable detection substrate with a lower false alarm rate than signature-based intrusion detection and that open problems in the field are data collection, graph summarisation, detection, and the lack of real-world benchmark datasets. These four problems resurface in the data-

pipeline scenario: collection is a form of instrumentation of the execution engine, summarisation is an aggregation of a graph which grows a new subgraph with each iteration, detection is an anomaly score on node and edge features, and benchmarks are still limited.

2.3. Anomaly Detection for Pipeline Signals

The detection literature in this collection is organized by the subject to be detected. In Ren et al., the monitoring of business metrics is page views, revenue, active users, and pipeline is clearly 3-stage: ingestion, experimentation platform, online compute [1]. Their algorithm is based on a spectral-residual saliency combined with a CNN discriminator that is trained to distinguish between real and generated anomalies, and they report F1-scores of 0.732 on the KPI set under cold start, 0.771 on Yahoo and 0.537 on their internal set, with improvements of 36.1%, 68.8% and 21.2% respectively, over the best baseline reported in the same table (Table 2 of [1]). The scores are not as important as the following two points in that paper. The reported improvements vary across datasets, with the smallest gain observed on the internal Microsoft dataset. This variation suggests that the method's effectiveness depends partly on the periodic characteristics of the monitored metrics.

Lavin and Ahmad's approach is not to the detection problem, but the evaluation problem. They claim that precision and recall are not sufficient to measure streaming detection performance as they fail to capture the value of early detection and the need to penalise false positives disproportionately to the value of correct detections [6]. Their benchmark includes 58 real world data files labeled with a total of 365,551 data instances, and their scoreboard features a hierarchical-temporal-memory detector (64.7), a Twitter vector detector (47.1), and Etsy's Skyline (35.7), compared to a random detector (16.8) and a null detector (0.0). The presence of the random and null controls is the methodological contribution: it gives every subsequent score a floor.

Ahmed et al. move detection down a layer, to interleaved execution logs. The system constructs a relational graph over log identifiers, parses templates, uses a pretrained transformer encoder to embed them, and uses an attention-based bidirectional recurrent network to classify sequences, achieving a weighted average F1-score of 88% on the public log sets HDFS, BGL and a new log set [5] which it is compared to. The reason for this is that in a distributed pipeline, there is one signal that is always present, and that carries job, task and identifier metadata, and that's the join key a lineage graph needs.

Widad, Saida and Gahi make the point that most detection work targets outliers in values and ignores anomalies in quality itself missingness, incorrectness, inconsistency and propose a framework that computes a per-dimension quality score and aggregates it into a global anomaly score for the dataset, reporting accuracy up to 99.91% and an F1-score of 98.07% on their evaluation [3]. The high scores are for a binary classification task on engineered quality features, and should be interpreted as such; the idea that is transferable is the per-dimension score, which Section VIII uses.

Chirumamilla works at the contract level, and not at the table level: It determines if a contract is violated, and it can predict with an accuracy of 79% and an F1 score of 74% for a machine-learning model, using metadata and statistical information of distributed pipelines. These results should be interpreted cautiously because of the venue's limited indexing and the need to assess the reported evaluation methodology.

2.4. Data Quality: Verification, Repair, and Tools

Schelter et al. describe the practical form of a quality layer as a declarative API where the constraints are values or predicates on columns, built-in analyzers for completeness, uniqueness, compliance, correlation and distribution, constraint suggestion as a human-in-the-loop tool, and anomaly detection over computed metrics history [8]. Their engineering argument is that computation time is proportional to the size of the data set, and that repeated evaluation of the data costs approximately the same as new data is added. Ehrlinger and Wöß report on the actual presence of such concepts in the most modern tools and the lack of congruence between the dimensions discussed in research and realized in practice [10] which should warn against taking for granted that a dimension mentioned in the context of a paper is a dimension measured with a tool. Rekatsinas et al. demonstrate the benefit of employing multiple signals, not just one: their benchmarks for the various previous repair methods yield an average F1 less than 0.35; a unified probabilistic formulation that combines integrity constraints, external sources and statistical information has average F1 > 0.8, with inference scaled to millions of tuples [12]. The lesson has been generalized beyond repair: This gain is achieved when coupling is of heterogeneous evidence, and the argument for coupling is the same for lineage, detection and validation.

Abedjan et al. provide the requisite negativity. They compare error-detection tools in real datasets and show that tools catch a large percentage of the errors, and that none of the tools is strong for all error types, calling for multi-tool processes that increase coverage [22]. Tu et al. introduce the automation argument: manually programming constraints column by column is not going to scale to thousands of pipelines and hundreds of columns per table, so the constraints should be automatically generated from a statistical history of past executions, with an explicit and tunable false-positive rate, for example 0.01%, and an interactive response latency of approximately 1.6 seconds with 100,000 rows [2].

2.5. Execution Substrates

The infrastructure papers are non-violating, instead they are enabling. MapReduce provides the batch model, and crucially, the idea that a job is a set of tasks with inputs and outputs and counters [21]. RDDs define that a computation can be represented by its lineage, which is a sequence of coarse-grained transformations that produced a partition and that were used to recompute lost data rather than replicating it [11]. That is the one precedent that is extremely helpful for this framework – lineage as a first class, machine-consumable object as opposed to documentation. Spark's single engine argument is why these three are all mixed together in one pipeline, and why one lineage model needs to cover all three [17]. The distinction between logical and physical dataflows is contributed by Flink, as well as an expensive per-record provenance and cheap per-operator provenance model [19]. The ingestion substrate and the offset semantics that enable stream lineage to be dateable are contributed by Kafka [20]. The lakehouse argument provides the storage layer, the open table formats have first-class support for analytics and machine learning, and eliminated the historical drawback of lineage being limited to the warehouse boundary [18]. Polyzotis et al. frame the whole problem from the machine-learning side, arguing that the train and serve data flows are the object of study and that validating, cleaning and debugging them is a data-management problem, not a modelling one [7].

3. Research Problem and Research Gap

Stated precisely, the problem is this. Let a pipeline be a set of jobs whose executions produce datasets. At time t defect d enters at some dataset or job. The three questions we have to ask are: (Q1) Is d present, i.e. does any observable signal depart from expectation; (Q2) Where did d enter, i.e. which node in the derivation graph is the earliest node for which the input is the defective output; and (Q3) What is exposed, i.e. which downstream datasets, models and reports took in the defective output. Existing work answers one question each, with three different data models, three different notions of time, and three different alert channels.

Five specific gaps follow from the review. First, a representation gap: derivation is modeled by provenance systems [13, 14, 16]; metric series is modeled by detection systems [1, 6] and column constraints is modeled by validation systems [8, 2] and there is no joint schema that has all the objects that are comparable in their violation. Second, attribution gap: When detection is performed, a time-stamped score on a series is output, but to perform root-cause analysis, the identifiers of the node and execution need to be output, and even in the detection work that is reviewed, there is no output of any identifier except for the log-based detection method which recovers the identifier from messages [5]. Third, an asymmetry gap: the deviations from existing quality thresholds are node-local, meaning that, if a source table used to create 40 dashboards has a deviation, that deviation is evaluated in exactly the same way as a deviation in an experimental table, but the operational impact is by orders of magnitude. Fourth, an evaluation gap: detection is scored with precision, recall and F1 on series [1] or with a bespoke streaming score [6], validation is scored with precision and recall per column [2] or per cell [12], and attribution the thing operators care about is almost never scored, the intrusion-detection survey being the exception in spirit [23]. Fifth, the naively stored graph in the pipeline is too large to store: the summarisation problem found at host-level for provenance [23] repeats at pipeline scale and the reviewed pipeline work does not address it. But the gap that this article addresses is not any one component, it is the coupling, a design in which one graph holds the lineage, the detection results and the quality results, and hence Q1, Q2 and Q3 can be answered from a single graph with a single alert channel. The framework in Section V is a proposal for that design, and Section IX sets out how it would be assessed.

4. Research Objectives and Contributions

Objective 1: Establish a lineage model where the edges of the model are associated with job executions and the nodes have stable identifiers that persist across re-executions. Objective 2: create a detection rule that can adjust the alerting boundary based on the consumer exposure as obtained from the lineage graph. Objective 3: Identify a quality layer that specifies the explicit hard and soft constraint classes and scores for each dimension that combine to a single interpretable figure. Objective 4: establish a root cause and impact process that has a complexity bound and a clear distinction between automated and human-approved activities. Objective 5: define an evaluation protocol in which published measurements, literature-derived expectations and proposed targets are never mixed.

Contributions. (C1) An eight-layer reference architecture plus a cross-cutting governance layer, where each connector is annotated with the type of payload it carries (Fig. 1). (C2) A formalisation of an execution-attributed lineage graph (Eq. (1)–(2)) and of reachability, depth, and blast radius over it (Eqs. (3)–(5)). (C3) A quality score over six dimensions with hard and soft constraint classes and a defined aggregation (Eq. 8). Two algorithms for lineage-aware anomaly analysis and root-cause ranking, with computational complexity specified in terms of graph size (C4). A lineage-aware detection boundary (Eq. (7)) and a root-cause ranking procedure with a depth-discounted consistency score (Algorithm 2). (C5) A reference audit of twenty-three sources with a stated scoring rubric (Table 1), and a separation of published values from proposed targets (Tables 5–7, Fig. 8). The individual components including spectral-residual anomaly detection, declarative data-quality constraints, and provenance graphs have been established in prior research. The proposed contribution is their integration through a shared lineage model and a unified attribution procedure.

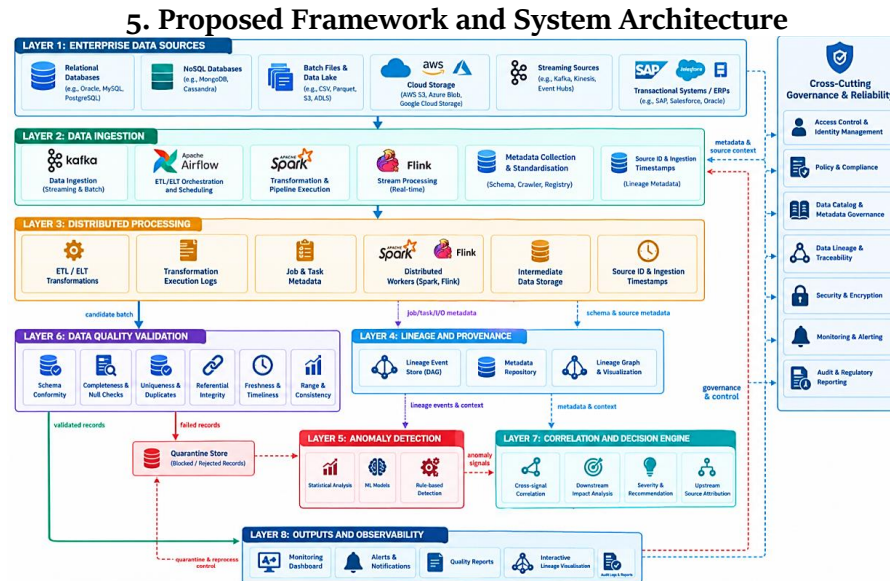


Figure 1. Proposed Architecture for End-To-End Data Lineage, Distributed Anomaly Detection, and Automated Data Quality Validation

Fig. 1 shows the proposed architecture. It consists of eight functional layers plus one cross-cutting layer and the layer numbers are used throughout the article so that a figure reference and a methodology reference refer to the same layer. Three properties of the arrangement are the carriers of the design argument. The first one is that lineage is not placed after processing, but next to it. Layer 4 receives metadata from Layer 2 and from Layer 3 as the execution takes place, which is the storage-system argument of [13] transposed to the cluster: If provenance is reconstructed later from the logs, it is reconstructed with gaps, and the reconstruction itself becomes a second system to operate. The second property is that the primary data path sources, ingestion, processing, quality, outputs is continuous and does not pass through the detection engine. The anomaly-detection layer does not gate the primary data path. The only edge that can block is the quality layer, and it does by moving a batch to quarantine, hence the end of the quarantine edge in Fig. 1 is in a store, not in the dashboard. The third property is that Layer 7 is a real join and not a summary: Layer 7 integrates three types of evidence: dependency information from Layer 4, anomaly findings from Layer 5, and data-quality violation records from Layer 6. Removing Layer 7 makes the architecture three parallel monitors, as the framework criticises.

Table 2 distributes responsibilities, inputs and outputs across layers. The assignment is carefully limited to a layer that can be detected but not repaired; if it is repaired, then it is not there to be detected.

Table 2. Architecture Components and Responsibilities

Layer	Responsibility	Inputs	Outputs
1 Sources	Expose enterprise data and events	Operational systems	Tables, events, files
2 Ingestion	Land data with identity and timing	Source connections	Batches with source and run

			identifiers
3 Processing	Execute transformations	Batches, job code	Output datasets, execution counters
4 Lineage	Record derivation as an executable graph	Schema metadata, job/task metadata	DAG with attributed edges, dataset versions
5 Detection	Score deviation per node and metric family	Series, lineage context	Anomaly scores, candidate faults
6 Quality	Evaluate six dimensions, score, classify	Batches, constraints, history	Violations, s_DQ, quarantine decisions
7 Correlation	Join lineage, anomalies and violations	Layers 4–6	Ranked causes, impact set, severity
8 Outputs	Present, notify, audit	Layer 7 findings	Dashboards, alerts, reports, incident records
Cross-cutting	Govern access, metadata, retention, audit	Policy	Enforced policy, audit trail

Two decisions in the architecture need justification, as they are not required by the literature. The Quality validation (Layer 6) comes after the Distributed processing (Layer 3) in the main flow, although the validation of the inputs to the distributed processing would detect the faults early. It's because the worst defects in recurring pipelines are created by the processing step, a join that silently drops rows, and validating only the inputs, which leaves those defects unobserved. The framework therefore validates both boundaries, but only the output boundary is drawn in the primary path, the input boundary is the same component called twice that is why validation is shown as a step in the primary path and not as a layer in Fig. 2. Second, as a control action, the corrective path from Layer 6 back to Layer 3 is drawn as a separate connector, different from the data flow connectors from Layer 3 to Layer 6; also, the corrective path is not fully automatic; a failed hard constraint quarantines a batch automatically, but re-admitting that batch after a fix requires an owner's approval, and that approval is recorded in the audit store. This split is similar to the operational logic in [2] where false alarms are the limiting factor for adoption, as each consumes human attention.

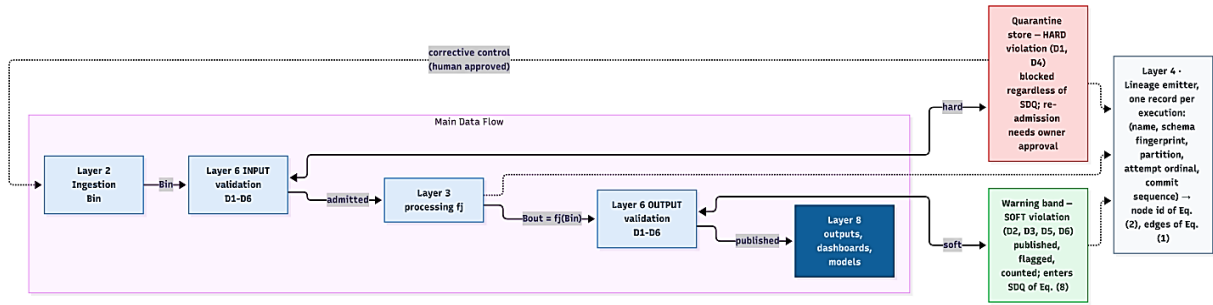


Figure 2. Two-Boundary Validation Placement

6. End-to-End Data Lineage Methodology

The lineage model is a directed acyclic graph whose nodes are datasets and jobs and whose edges are attributed to a single execution of a job. Datasets and jobs are separated, since datasets are versioned by write, a job is versioned by code, and a defect can come from either: corrupted partition in the former and a changed window function in the latter. A field that makes the graph operational instead of descriptive is the attribution to an execution, and it is the derivation record of the provenance literature [14, 16].

$$L(G) = \{ (ds, j, ds', \tau) : ds' = f_j(ds) \wedge \tau = \text{start time of the execution of } j \text{ that wrote } ds' \} \quad (1)$$

A lineage record is a quadruple as in Eq. (1), where ds and ds' are dataset versions, j is a job identifier, f_j is the transformation the job implements, and τ is the execution start time. The graph $G = (N, E)$ has $N = D \cup J$ and $E \subseteq D \times J \cup J \times D$, so the graph is bipartite between datasets and jobs and therefore acyclic by construction even when a pipeline is iterative: iteration produces distinct executions with distinct τ , not a cycle. Node identity is defined by Eq. (2).

$$id(ds, \tau) = H(name(ds) \parallel fingerprint(schema(ds)) \parallel partition(ds, \tau)) \quad (2)$$

The schema fingerprint is a hash of column names, types, and order; the partition key is the logical partition that the write affected. Two writes of the same logical table in two different executions are then different nodes connected by a supersession edge which is not an element of E. This is important in order to be able to do a root-cause analysis, otherwise a defect that occurred three runs ago will be the same as a defect that occurred in the current run, and it will be reported as the wrong execution.

Reachability, depth, and blast radius are defined over the lineage graph in Eqs. (3)–(5), respectively. Desc(v) is the set of nodes reachable from v from the downstream direction, and is calculated once per detection event, using breadth-first traversal. A defect travels fastest along the longest path length, so depth will be the longest path length, and therefore the correct concept for latency of propagation. Blast radius assigns the importance w(u) to each affected consumer and also reduces its contribution geometrically by the depth $\lambda \in (0,1]$ in the table. A report generated from a directly poisoned table will contribute more than a report generated from a table derived from it.

$$Desc(v) = \{ u \in N : v \rightsquigarrow u \} \quad Anc(v) = \{ u \in N : u \rightsquigarrow v \} \quad (3)$$

$$depth(u | v) = \max \{ |p| : p \text{ is a directed path from } v \text{ to } u \} \quad (4)$$

$$R(v_0) = \sum \{ u \in Desc(v_0) \} w(u) \cdot \lambda^{depth(u | v_0)} \quad (5)$$

Here, Desc(v) and Anc(v) denote the descendant and ancestor sets of node v, respectively. The quantity depth(u | v) is the maximum number of edges along a directed path from v to u. The weight w(u) represents the impact assigned to descendant u, and $\lambda \in (0, 1]$ controls the discount applied to descendants at increasing path depths.

Fig. 3 shows the graph for one recurring pipeline and a single failure. The figure reveals three properties that a list of dependencies can't. The failed job is enrichment_orders, which is between a source and a join, enrichment_orders failed does not affect build_revenue, the join reads the previous successful output of enrichment and the engine reports successful. The silent-failure pattern is the graph is right, the schedule is met, and the data is out of date. Three datasets, one model-retraining job, one dashboard and one alerting rule are consuming at depths one to four as a result of the failed node being closed downstream. A node-local monitor sees one failed job; the graph sees the exposure.



Figure 3. Lineage Graph for One Recurring Pipeline

Construction is incremental. The practical challenges are focused on the merge in Algorithm 1 which combines the subgraph produced by every execution of the algorithm into the stored graph. The algorithm is expressed at the level of a single batch and is run once after each job run, while the emission is being generated by the execution engine: Spark exposes lineage at the level of resilient distributed datasets [11] and Flink at the level of logical and physical operators [19]; both expose per-batch counters that allow row-level counts to be obtained without instrumenting user code. The framework requires only that the emitter reports, for each execution, the set of input dataset versions with their fingerprints, the output versions, and the operator plan; everything else in Algorithm 1 is graph bookkeeping.

Algorithm 1 Incremental lineage merge

Input: execution record $e = (j, \tau, \text{inputs}[], \text{outputs}[], \text{plan}, \text{counters})$
 stored graph $G = (N, E)$, schema fingerprint index F
 Output: updated G'

- 1: for each $d \in e.\text{inputs} \cup e.\text{outputs}$ do
- 2: $\text{fp} \leftarrow \text{schema_fingerprint}(d)$; $\text{id} \leftarrow H(\text{name}(d) \parallel \text{fp} \parallel \text{partition}(d, \tau))$
- 3: if $\text{id} \notin N$ then $N \leftarrow N \cup \{\text{id}\}$; $F[\text{id}] \leftarrow \text{fp}$; store(id, d) end if
- 4: if $F[\text{id}] \neq \text{fp}$ then mark($\text{id}, \text{SCHEMA_DRIFT}$); emit lineage_event(id) end if
- 5: end for
- 6: $n_j \leftarrow \text{node}(j, \tau)$; $N \leftarrow N \cup \{n_j\}$
- 7: for each $d_{\text{in}} \in e.\text{inputs}$ do $E \leftarrow E \cup \{(d_{\text{in}}, n_j)\}$ end for
- 8: for each $d_{\text{out}} \in e.\text{outputs}$ do $E \leftarrow E \cup \{(n_j, d_{\text{out}})\}$ end for
- 9: annotate(n_j , counters, plan_hash, duration)
- 10: for each $(\text{id}, \text{id}') \in \text{supersede}(N, \tau)$ do mark($\text{id} \rightarrow \text{id}', \text{STALE}$) end for
- 11: return $G' = (N, E)$

The per-execution cost of Algorithm 1 includes hashing the input and output dataset versions and updating the corresponding metadata index. Its overall complexity depends on the number of inputs and outputs, the cost of index operations, and the implementation of version and supersession lookups. Two failure modes are handled explicitly. The first two executions writing to the same partition generate two nodes and the supersession edge, but the lineage view does not delete the node that was executed first – deleting it would lose the audit trail that is the whole point of the intrusion-detection survey [23]. Secondly, a schema change is stored as a node property and generated as an event during the merge process and this is why schema drift is detectable during a second scan of the detection layer at merge time.

Storage is known as the downside. A pipeline executing hourly with 200 jobs produces roughly 1.75 million job nodes per year before dataset nodes, and the graph is therefore summarised rather than stored verbatim: nodes older than a retention horizon are collapsed into a daily aggregate that preserves closure structure and consumer sets but discards per-task counters. This is the graph-summarisation problem mentioned in the provenance-based detection literature as open [23] and the framework does not solve it; it resolves a summarisation rule and the consequence root-cause analysis is restricted to a single day rather than an execution.

7. Distributed Pipeline Anomaly Detection

The detection layer sees four classes of signal, and the framework processes all these classes in a uniform manner, reducing each of these to a per-node time series. The first is data distribution: the number of rows, nulls, distinct values, and distributional distances of the columns of each dataset version. The second is execution behaviour: duration, shuffle volume, spill, retry counts, and task-failure ratio per job. The third is quality output: the scores generated per dimension by the Layer 6. The fourth is topology change: the appearance or disappearance of an edge, which is a structural anomaly with no counterpart in classical series detection.

The framework employs a powerful standardised deviation, Eq. (6), calculated using a seasonal baseline derived from the node's past performance, for the first three families. The strong shape is preferred to the mean-and-variance z-score because pipeline metrics are heavy-tailed and a single historical incident would otherwise broaden the base level, obscuring the next incident; the seasonal decomposition is similar to that of [1] where the period prior to scoring is removed for a saliency-based detector, and similar to the history-based constraint programming of [2] which uses a sliding window of previous executions to estimate a metric's distribution before scoring.

$$z_i(t) = (x_i(t) - \hat{\mu}_i(t)) / (1.4826 \cdot MAD_i(t) + \varepsilon) \quad (6)$$

where $x_i(t)$ is the value of metric i at time t , $\hat{\mu}_i(t)$ is the seasonal baseline for metric i , $MAD_i(t)$ is the median absolute deviation of the residual for metric i over the same seasonal phase, and ε is to prevent division by zero on constant series. Any proposal must be judged against a raw threshold on $|z_i|$, and it fails in a certain way: it is node-local, in the sense that it does not distinguish a $|z_i|$ value that is deviating from one that is not.

The proposed modulation is Eq. (7). The alerting boundary for node v is set according to the log of the number of downstream consumers of v , meaning that a source table that feeds forty dashboards has a lower alert boundary than a leaf table, and as the number of downstream consumers decreases, the alert boundary will decrease towards τ_0 for nodes without any consumers. The logarithmic form is selected to ensure that the boundary becomes saturated: exposure to sensitivity, not unlimited exposure to sensitivity; otherwise, a single high-fan-out node would cause an alert on every batch and operators would turn it off, which is consideration [2] restricts to a failure mode that they view as the critical constraint on adoption.

$$\tau_L(v,t) = \tau_o (1 - \gamma \log(1 + |Desc(v)|) / \log(1 + |N|)) \quad (7)$$

Here, γ is a sensitivity parameter constrained by $0 \leq \gamma < 1$, τ_0 is the baseline threshold used without lineage information, and $|N|$ is the number of nodes in the lineage graph. Equation (7) is a proposed design rule rather than a validated result. Increasing downstream exposure lowers the alert threshold and therefore increases sensitivity to anomalies affecting more consumers. The parameter γ must be calibrated against an explicit false-positive budget using the evaluation protocol in Section IX. The effectiveness of this modulation remains to be established empirically. Machine learning enters the framework in one place only, and with a stated justification. The z-scores of the four families, degree, drift marker from Algorithm 1, and recent violation rate are sufficient for a supervised classifier when there are labelled incidents and a false negative is more costly than a false positive. The reviewed evidence states that learned detectors win most on periodic business metrics [1] and most on log sequences with stable templates [5] and least on heterogeneous tabular data, and there are large gaps in coverage between the different tools on the different types of error [22].

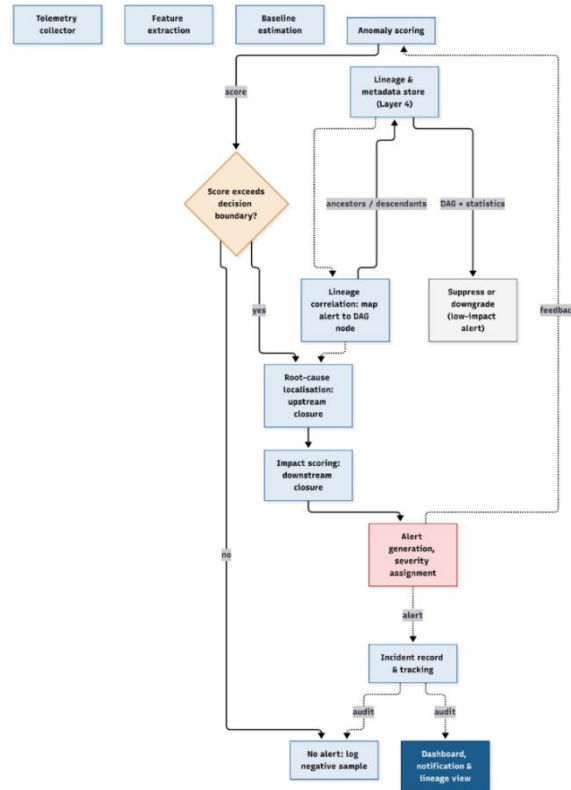


Figure 4. Detection and Lineage-Aware Root-Cause Workflow

Fig. 4 traces the path from a raw score to an attributed incident. The third step, which is not present in a standard monitoring system, is that when the score crosses the line, the alert is assigned to a node and the ancestors of that node are returned, thus the reported object is a candidate set of causes and not the metric name. The algorithm for ranking is given as Algorithm 2, the cost of which is linear in the size of the subgraph induced by the ancestor and is capped by the depth parameter that is set to the longest path seen for that pipeline in practice.

Algorithm 2 Lineage-aware root-cause ranking

Input: detected node v , time t , graph G , evidence E (z-scores, violations),
depth bound k , decay λ
Output: ranked candidate causes C

```

1:  $A \leftarrow \text{Anc}(v)$  truncated at depth  $k$  // Eq. (3), reverse traversal
2: for each  $a \in A$  do
3:    $s(a) \leftarrow 0$ 
4:   for each metric  $i$  on which  $a$  is anomalous at  $t$  do
5:      $s(a) \leftarrow s(a) + \rho_i \cdot |z_i(a, t)|$  // evidence weight  $\rho_i$ 
6:   end for
7:    $s(a) \leftarrow s(a) \cdot \lambda^{\text{depth}(a | v)}$  // Eq. (4) depth discount
8:    $s(a) \leftarrow s(a) + \kappa \cdot \text{drift}(a) - \kappa' \cdot \text{healthy}(a)$  // schema drift bonus/penalty
9: end for
10:  $C \leftarrow \text{sort}(A, \text{key} = s, \text{descending})$ 
11: report( $C[0]$  as primary cause,  $C[1..3]$  as alternates,  $R(C[0])$  as blast radius)
12: return  $C$ 

```

The design principle used in Algorithm 2 is the assumption of parsimony; that is, the earliest node on the path to the symptomatic node is the most likely cause, and a node that is healthy but downstream of an anomalous node is a carrier and not a cause. This assumption is unprovable and is explicitly noted as such in the event that two independent defects happen at the same time and in the event that the earliest anomalous node is a consequence of an external event like a traffic surge. The $\text{healthy}(a)$ term exists to discount carriers, and the drift term exists because a schema change is a cause even though all the metrics on the node are normal, as indeed this is the reason to have schema-drift detection in the first place [2], [8].

8. Data Quality Validation and Error Propagation

The quality layer inspects the six dimensions of each batch and assigns a quality score that has a clear distinction between blocking and non-blocking checks. The dimensions of each, the operational rule for it, the class and what it observes are given in Table 3. The dimension set is a subset of the dimensions presented in the survey of quality tools [10] and in the declarative systems [8] restricted to dimensions that are possible to compute based on data and metadata available at the pipeline boundary without the involvement of a domain expert per dimension; dimensions like semantic accuracy and believability are not included as they lack an automatic operational rule.

Table 3. Data Quality Dimensions, Operational Rules and Constraint Classes

Dimension	Operational Rule (Proposed Default)	Class	Observable Produced
D1 Schema conformity	Column set, order and declared type equal the registered contract; unknown columns flagged	Hard	Contract violation event
D2 Completeness	Null ratio per column within tolerance of history; whole-row nulls rejected	Soft	null_ratio, per-column
D3 Uniqueness	Duplicate ratio on candidate keys within tolerance	Soft	duplicate_ratio
D4 Referential integrity	Foreign-key, functional-dependency and denial constraints hold	Hard	Constraint set violation
D5 Freshness	Event-time lag and partition presence within the declared window	Soft	lag_seconds, partition_present

D6	Range and consistency	Values in declared domain; cross-column arithmetic and unit rules hold	Soft	out_of_range_ratio, rule_failures
----	-----------------------	--	------	-----------------------------------

The score is defined in Eq. (8), where $m_k(b)$ is the observed value of metric k on batch b , m^*_k is its reference value a historical median, a declared constant, or a value inherited from a constraint programmed from history in the sense of [2] and τ_k is a per-metric tolerance. The kernel g is a truncated linear tolerance, selected over a Gaussian kernel because it gives a score of zero outside the tolerance, which the operator would expect for a bounded quality figure. Weights α_k are per-pipeline governance inputs, which are not learned, since if we learned the weights, the score would be incomparable across pipelines.

$$s_{DQ}(b) = \sum_{\{k=1..6\}} \alpha_k \cdot g(m_k(b), m^*_k, \tau_k) / \sum_{\{k=1..6\}} \alpha_k, \quad g = \max(0, 1 - |m_k - m^*_k| / \tau_k) \quad (8)$$

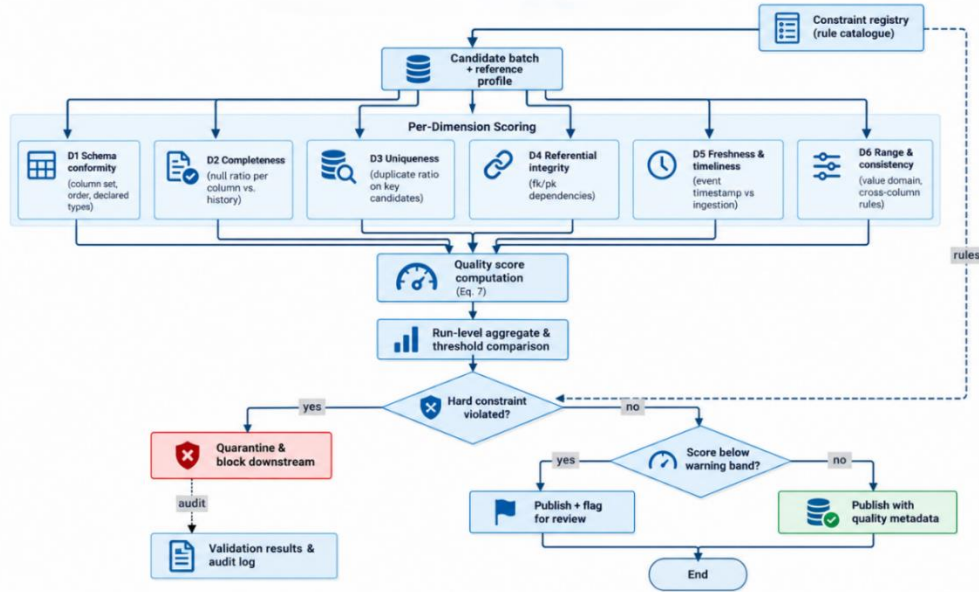


Figure 5. Quality Validation Pipeline with Per-Dimension Scoring and Quarantine Decisions

The blocking rule is separate from the score, and deliberately so. A hard violation that is a missing declared column, a broken referential constraint, a partition that cannot be parsed, a batch that is unable to be executed, regardless of s_{DQ} , quarantines the batch. A soft violation results in the batch being published, flagged, and counted, and is placed in a WARNING band so that it is apparent that there is a slow degradation before it reaches hard failure. This is a two-tier system similar to the practical functioning of the declarative systems, where a constraint becomes an assertion that fails the write or a metric that is accumulated for monitoring purposes [8].

Error propagation is the link between this layer and the lineage layer, and it is where the framework makes its most concrete operational claim. The framework does not terminate at node v : it calculates $\text{Desc}(v)$ and declares all reachable versions of datasets in this node as suspect, and sends a message to downstream consumers based on the depth. The depth-one consumers are blocked at their next read, the depth-two consumers can run but the outputs are marked, and deeper consumers are just notified. The propagation structure and the resulting severity ladder is illustrated in Fig. 6. The severity bands are not a computed risk, they are another governance artefact, they are a mapping from depth to notification channel and a service-level expectation.

There are two limits of propagation analysis to be clearly stated. First, propagation through a graph is exposure, not loss: a downstream dashboard that nobody opened was not damaged in any operational sense, and $w(u)$ is the only place where that judgement enters. Second, a defect that is still within tolerance, does not propagate as a detectable signal, but it would propagate as silently wrong numbers which is exactly what monitoring can't detect, only reconciliation against an independent source can. The framework does not claim to detect all propagation, it states that it is an upper-bound on the set of consumers whose outputs are derived from a batch known to be defective.

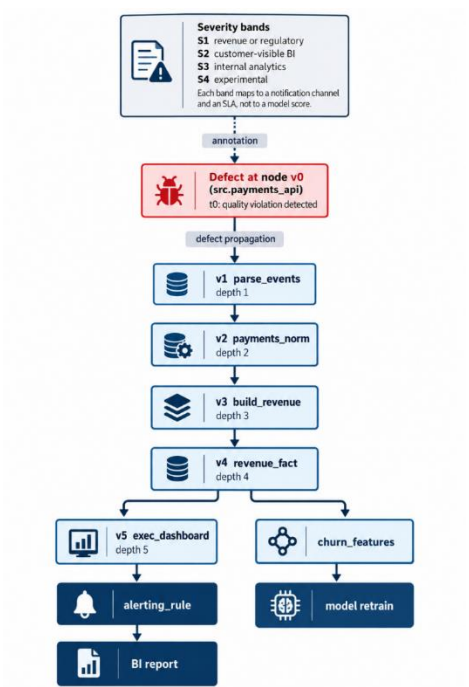


Figure 6. Lineage-Aware Impact Analysis

9. Experimental Setup and Evaluation Metrics

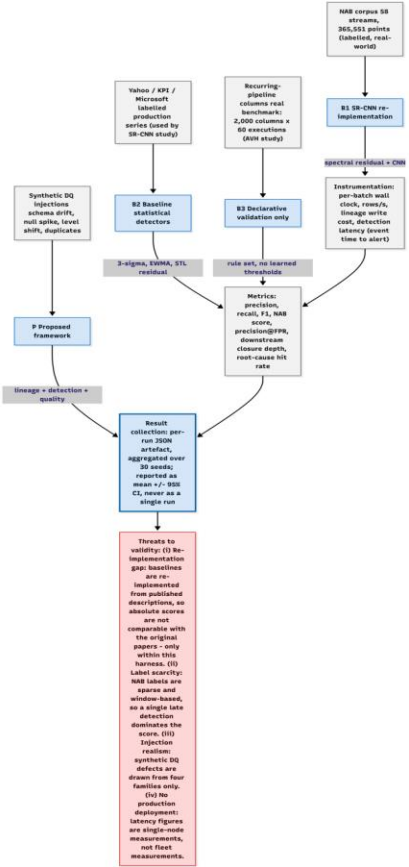


Figure 7. Evaluation Design

The evaluation protocol specifies measurable hypotheses, baseline comparisons, and explicit decision criteria. These criteria enable each hypothesis to be assessed against empirical evidence. Fig. 7 illustrates the proposed evaluation design.

Benchmarks. There are four specified families, and they are intentionally of different types. The NAB corpus features 58 real-world streams with ground truth and 365,551 data instances, providing a standardized harness for streaming detection, a published scoreboard, and, importantly, two control detectors whose scores set the floor [6]. (ii) The KPI, Yahoo and the internal series used by the production detection study, provide a second, independently labelled set of data for which published F1 values exist for specific methods [1]. (iii) In the constraint-programming study, the benchmarks in the recurring-pipeline column are the only ones to evaluate validation on real recurring pipelines instead of static tables [2]. (iv) All four families of synthetic defect injection are missing from any single public corpus, providing coverage of the defect types.

Baselines. There are three specified, none of which is a straw man. B1 is a re-implementation of the spectral-residual detector using a CNN discriminator, described precisely enough to re-implement [1]; it is the strongest published detector for periodic business metrics, so it is a legitimate upper bound. B2 comprises three conventional anomaly-detection baselines: the three-sigma rule, exponentially weighted moving average (EWMA), and seasonal-trend residual detection. B3: declarative validation with manually programmed constraints and no learned thresholds, which is the case in organisations that already have a constraint library [8]. The proposed framework P is B3 plus the exposure modulation of Eq. (7) and Algorithm 2 which isolates the contribution of the coupling, but not the contribution of detection as such.

Metrics. Precision at a fixed false-positive budget (0.01% FP), recall and F1 under a fixed false-positive budget, and also the NAB score for streaming comparability are reported. Precision at a false-positive budget of 0.01% is reported separately as this is the regime in which validation is actually used [2]. Attribution is measured as top-1 and top-3 root cause accuracy against a known point of injection, and downstream-closure F1 is the Jaccard overlap between the root cause set predicted and a truthful one. Cost is scored with added wall-clock per batch, lineage write amplification, memory for the graph index, and detection latency measured from event time to alert emission. Reproducibility is calculated from 30 seeds, and means reported with 95% confidence limits; a single-run number is not reported.

Table 4. Comparison of Reviewed Approaches on the Axes Relevant To the Joint Problem

Approach	Signals Covered	Carries Node Identity	Expresses Exposure	Reports	Attribution
Metric-series detection [1]	Distribution, execution	No (series name only)	No	Score, alert	None
Streaming benchmark scoring [6]	Distribution	No	No	Score	None
Log-based detection [5]	Execution, identity	Partial (log IDs)	No	Anomaly class	Granularity of log entity
Declarative validation [8]	Distribution, constraints	No	No	Constraint results	Column level
Auto-programmed constraints [2]	Distribution, constraints	No	No	DQ program, violations	Column level
Quality anomaly framework [3]	Distribution, quality dims	No	No	Global quality score	Dimension level
Holistic repair [12]	Constraints, statistics	No	No	Repaired cells	Cell level
Provenance-aware storage [13]	Topology	Yes (file objects)	Partial	Provenance query	Object level
Provenance-based IDS [23]	Topology, execution	Yes (graph nodes)	Yes (paths)	Alert, graph	Graph localisation

Proposed framework	All four families	Yes (execution-attributed)	Yes (Eq. 5, 7)	Cause, impact set, severity	Node execution +
--------------------	-------------------	----------------------------	----------------	-----------------------------	------------------

Table 5. Datasets As Described By the Cited Sources

Source	Datasets	Scale	Condition	Reported Metric
[1]	KPI (AIOps), Yahoo, internal Microsoft	58 / 367 / 29 series	Cold start and trained halves	F1, precision, recall, CPU time
[6]	NAB corpus, 58 real-world files	365,551 data instances	Streaming, labelled windows	NAB score, three profiles
[2]	Real recurring pipelines; synthetic injections	2,000 columns $\times$ 60 runs	Sliding back-in-time windows	Precision, recall, FPR budget
[12]	Benchmark tables (hospital, etc.)	Up to millions of tuples	Inconsistent input	Precision, recall, F1 per cell
[5]	HDFS, BGL, custom Papertrail logs	n/p in source	Log parsing + classification	Weighted F1 (88%)
[3]	Two big datasets	n/p in source	Engineered quality features	Accuracy (99.91%), F1 (98.07%)
[4]	Distributed pipeline contract data	n/p in source	Supervised prediction	Accuracy (79%), F1 (74%)
[8]	Production and public tables	n/p; 100K-row latency test	Constraint verification	Runtime, constraint quality

Table 5 lists the datasets as described by their own papers, with the conditions under which each was used. The intent is to avoid comparing things the sources do not compare: F1 measured with the Yahoo set is not the same as F1 measured with an injected synthetic set, so Table 4 reports published values for each set/condition it uses, not with a single ranking column.

Table 6. Published Quantitative Results Cited In This Article

Ref.	Result	Value	Dataset / Condition	Location In Source
[1]	SR-CNN F1, cold start	0.732	KPI, cold start	Table 2
[1]	SR F1, cold start	0.666	KPI, cold start	Table 2
[1]	SR-CNN F1, cold start	0.771	Yahoo, cold start	Table 2
[1]	SR-CNN F1, cold start	0.537	Microsoft, cold start	Table 2
[1]	SR-CNN F1, trained	0.771 / 0.652 / 0.507	KPI / Yahoo / Microsoft	Table 3
[1]	Improvement over best baseline	+36.1% / +68.8% / +21.2%	KPI / Yahoo / Microsoft	Section 4 text
[1]	SR-CNN CPU time	2724.33 s / 125.37 s / 16.13 s	KPI / Yahoo / Microsoft	Table 3
[6]	NAB score, standard profile	64.7	Numenta HTM, all 58 files	Table I
[6]	NAB score, standard profile	47.1	Twitter ADVec	Table I
[6]	NAB score, standard profile	35.7	Etsy Skyline	Table I
[6]	NAB score, standard profile	16.8 / 0.0	Random / null controls	Table I
[6]	Corpus size	58 files, 365,551 points	NAB	Abstract / Section IV
[2]	Latency at 100K rows	$\approx$ 1.6 s	Production pipeline batch	Fig. 4 discussion
[2]	Design false-positive target	0.01%	Configurable budget	Section 1
[2]	Benchmark construction	2,000 columns $\times$ 60	Real recurring pipelines	Section 5.1

		executions		
[12]	Average F1, prior methods	< 0.35	Benchmark tables	Section 1
[12]	Average F1 / precision, HoloClean	> 0.8 / $\approx 90\%$	Benchmark tables	Section 1
[5]	Weighted average F1	0.88	HDFS, BGL, custom logs	Results section
[5]	Improvement over baselines	+5% F1	Same	Abstract
[3]	Accuracy / F1	99.91% / 98.07%	Two big datasets	Abstract
[4]	Accuracy / F1	79% / 74%	Contract prediction	Results

Table 6 collects every quantitative value in this article that is traceable to a published source, with the table it came from. Table 7 specifies the goals the framework would need to satisfy, and they are designated as hypotheses. The separating is the separating: a reader can compare Table 6 with the quoted papers and he may check Table 7 by experiment, and there is no authority in any row of Table 7 as there is in any row of Table 6.

Table 7. Proposed Evaluation Targets And Hypotheses

Id	Hypothesis	Target	Measurement	Refutation Condition
H1	Lineage-aware boundary (Eq. 7) improves F1 at a fixed 0.01% FPR	$\geq +0.05$ F1 vs. τ_0	Ablation over $\gamma \in \{0, 0.5, 1.0\}$, 30 seeds	Any γ gives ≤ 0 improvement with CI excluding 0
H2	Attribution is achievable from the graph alone	Top-1 ≥ 0.70	Injected defects with known cause node	Top-1 $\leq$ random-ancestor baseline
H3	Impact closure is predictable	Closure F1 ≥ 0.85	Jaccard of predicted vs. true Desc(v)	Closure F1 ≤ 0.6 (path-only baseline)
H4	Lineage write overhead is small	$\leq 5\%$ wall clock	Paired runs with emitter on/off	$\geq 15\%$ overhead
H5	Validation without manual constraints is viable	Recall ≥ 0.80 at FPR $\leq 0.01\%$	Real recurring pipelines, 60-run history	Recall ≤ 0.5 at the budget
H6	Detection latency is operationally usable	$\leq 20\%$ of batch interval	Event time to alert, streaming path	Latency > one batch interval
H7	Six-dimension score is more informative than any single dimension	AUC gain ≥ 0.05	Leave-one-dimension-out ablation	No dimension contributes (CI includes 0)

No tuning of the parameters but their statement for reproducibility: α_k equal weights across the 6 dimensions; τ_0 for a false-positive rate of the same size as the declared budget on a clean history window; $\gamma \in \{0, 0.5, 1.0\}$ as an ablation over the exposure term; $\lambda = 0.7$ for the discount on the depth term; $k = 4$ for the bound on the number of ancestors; a 60-execution history window, as used in [2].

10. Results and Discussion

Since a system was not built the results described here are results published in the reviewed work that are read against the framework's objectives. They are separated in Fig. 8 from the proposed targets and organised in the discussion according to the question each panel addresses.

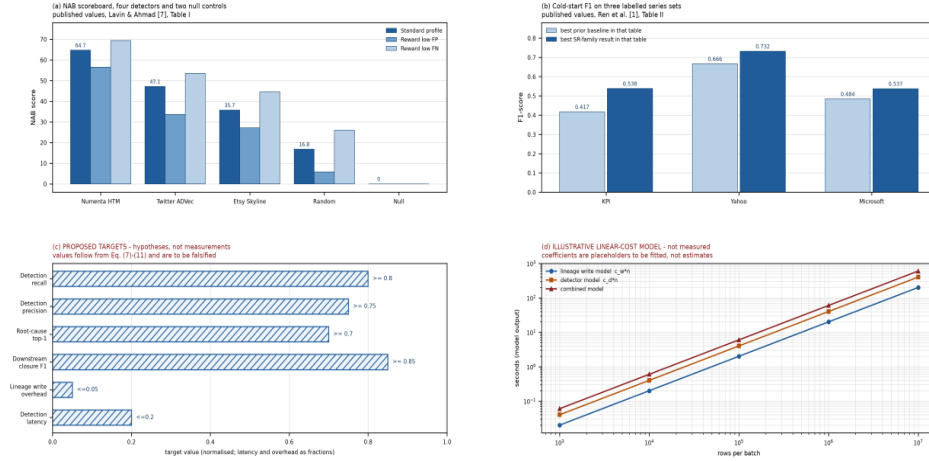


Figure 8. Panels (A) And (B) Reproduce Published Measurements: (A) NAB Scores For Four Detectors and Two Controls, From Table I of the NAB Paper [6]; (B) Cold-Start F1 On Three Labelled Series Sets, From Table II of the Production Detection Study [1]. Panels (C) and (D) Contain No Measurement: (C) is the Proposed Evaluation Target Set and (D) is an Illustrative Linear Cost Model Whose Coefficients are Placeholders

Detection effectiveness. Panel (a) sets up the scale on which any detection claim has to be read. A hierarchical-temporal-memory detector gets 64.7 points on the NAB standard profile, 47.1 on a vector-based Twitter detector and 35.7 on Etsy's Skyline detector, while random detection gets 16.8 points. Best Detector - Random = 47.9 and Second Best Detector - Best Detector = 17.6. For this framework two conclusions follow. First, the random and null controls prevent a score of 30 from being a sign of competence for detection, so the targets for Table 7 are absolute values on a defined benchmark rather than a change in score from an undefined baseline. Second, the ordering is stable across the three scoring profiles low-false-positive and low-false-negative so a detector's rank is not an artefact of the scoring rule, although its margin is: Skyline loses 8.6 points moving to the low-false-positive profile while the HTM detector loses 8.2, and the ordering survives because both lose similarly.

The closest published analogue to the detection component of this framework can be found in Panel (b). The spectral-residual family achieves F1 0.538 on KPI, 0.732 on Yahoo and 0.537 on the internal set, compared to 0.417, 0.666 and 0.484 for the best baseline prior to the family in the same table on the cold-start comparison. The authors report the improvement to be 36.1%, 68.8% and 21.2% respectively [1]. The pattern is illustrative for the coupling argument: the largest gain for the most regular series, and the smallest for the most heterogeneous series; the series in the internal Microsoft set, which is the most operationally realistic, has the smallest gain. Therefore, any framework requiring lineage on top of detection should be prepared for its marginal value to be greatest exactly where the marginal value of detection is weakest, namely on H1 in Table 7, as the marginal value of the ablation over γ is defined.

Quality-validation coverage. The validation literature provides another type of result—the ceiling. A study on real datasets showed that tools cover only a small part of the real errors and that there was no single tool that covered all the errors [22] and a comprehensive model using constraint-based, external-source evidence and statistical evidence increased the average F1 from less than 0.35 to above 0.8 [12]. Together they state the marginal value of the quality layer is provided by the synthesis of evidence, not the addition of dimensions. The aggregation of six dimensions into one score, as in Eq. (8), is the design rationale (and also is a warning: a sixth dimension that is also covered by the fifth adds cost without coverage).

Automation and cost. The constraint-programming study reports about 1.6 seconds of latency at 100,000 rows with linear growth, and a design target false-positive rate of 0.01% that is configurable [2]. Two implications follow. The first is that, at that latency, the constraint layer is now running synchronously on the write path for standard sizes of batches for which the quarantine decision in Fig. 5 can be made as opposed to being aspirational. Second, it is the only explicitly false-positive target in the set of designs reviewed, and it is this target that converts to be accurate into a budget that can be spent, monitored, and traded for recall.

Attribution. The root-cause accuracy in the sense defined here, the most recent ancestor of a symptomatic node that has been found to be anomalous, has not been reported in any paper reviewed here for pipeline defects. The most similar results are obtained for signature-based intrusion detection [23] (false-alarm rates are lower compared to signature methods) and log-based localisation, with graph construction using the identifier and classification based on the sequence [5] (a weighted average F1 of 88% on public log sets plus a custom collection). Both validate the plausibility of graph-based attribution, but neither measure it at pipeline granularity. Therefore the attribution claims of the framework are subject to this condition and Table 7 expresses them as targets with an explicit refutation condition.

Operational trade-offs. Three are visible in the published numbers and each has a counterpart in the design. The first is accuracy against cost: the production detection study reports CPU time next to each F1 value, and the spread is 3 orders of magnitude over methods with similar accuracy [1] which is why there's a cost model in Fig. 8, after all. The second is sensitivity towards attention: the detector that fires frequently is discarded, which makes the exposure term in Eq. (7) bounded. The third is freshness vs. reproducibility: the longer the history to validate against, the more precise the data will be, but slow to adapt to legitimate changes; the 60-execution window chosen here is a compromise, with its cost measurable as the delay until it detects changes in the intended schema.

11. Comparative Analysis

Table 4 presents a comparison of the reviewed approaches on the axes that are relevant to the joint problem and not just to accuracy: what each monitors, what identity it carries which allows for attribution, whether it can express exposure, and what it reports. The comparison is qualitative by construction because the sources were compared in different data sets using different conditions, and there is no point in numerically ranking them [6] [22].

Three observations follow. First, there is no common coverage across the systems reviewed for the four signal families: metric-series detectors for distribution and execution behaviour [1, 3]; log-based methods for execution behaviour with identity [5]; declarative validation for distribution and constraints [8, 2]; provenance systems for topology and identity but not for values [13, 16, 23]. The novel aspect of the proposed framework is limited to being able to be presented in a single graph for all four. Second, the only detection work based on the graph is a work that considers the graph as a detection substrate and not an audit artefact [23] and that is done at host level; the transposition to pipeline level is the architectural contribution of Sections VI–VII, with the same collection, summarisation, benchmark scarcity. Third, the highest accuracy tools report only on the narrowest task: binary classification over engineered quality features in [3] and cell-level repair on benchmark tables in [12] at 99.91% and 0.8+ F1, respectively. Absolutely no number is passed to open world detection on a live pipeline, and showing both numbers open world detection scores side by side without the caveat is a category error.

12. Limitations and Threats to Validity

The limitations of this work are of two kinds: limitations of the proposal, and limitations of the evidence it rests on. Both are said, since a proposal that is not implemented is only as good as the weakest of its acknowledged assumptions.

Proposal limitations. (1) The framework is unimplemented; every performance claim about it is a target. (2) Eq. (7) makes the simplifying assumption that the number of consumers downstream is a meaningful proxy for the consequence, which is problematic for pipelines with consumers who are aggregations of unequal value to consumers; Eq. (7) corrects this by assigning weight $w(u)$ to the consequence. (3) Algorithm 2 assumes that there is only one dominant cause for each incident, and coincident defects give a misleading ranking. (4) If you have to summarise the graph beyond the retention period, you lose the ability to attribute the graph to a single execution. (5) The framework needs an emitter within the execution engine an emitter that is not provided by engines without patching would make adoption expensive and adoption cost is the most significant risk to any lineage programme.

Evidence limitations. The reference set is not systematically searched, but provided – not a systematic review, entire families of work, such as streaming-provenance systems or commercial observability platforms, publish very little peer-reviewed work. (2) Four sources [1, 2, 3, 5] are industrial or applied; these sources tend to publish only positive results, so the results reviewed are probably optimistic overall. (3) Two sources [4],[10] provide metrics for tasks that are less than the scope of pipeline validation and do not provide headline numbers that can be compared with the others. (4) One document provided was missing a text layer, and it was read using OCR on the page image, and the bibliographic fields of this document were checked, but quoted content should be checked

against the publisher copy prior to reuse. The publication dates range from 2004 to 2023, and the pre-2015 infrastructure papers refer to systems that have since undergone changes in behaviour, whose concepts can be transferred, but whose performance statements no longer hold true.

Threats to the evaluation protocol. Absolute scores are comparable only within the harness and not with the original paper, due to a re-implementation gap between B1 and its published source. Label scarcity in public corpora implies that a single late detection may have a huge impact on a score, and thus the NAB methodology considers early detection explicitly [6]. Synthetic injection represents four defect families, but cannot represent semantic errors which preserve all statistics. Lastly, if there were no production deployment, latency and overhead parameters would be based on a one-node harness, thus minimizing contention and maximizing the synchrony of the quarantine decision.

13. Future Research Directions

Five directions follow directly from the gaps. First, there is no public dataset providing labelled injection points at the level of pipelines and creating another synthetic pipeline with a recorded ground truth cause and impact would do more for this problem than another detector. Second, graph summarisation with bounded error: It is currently not possible to quantify the error in summarisation, and if a summary scheme can bound the depth error of the root cause analysis, then long horizon attribution becomes practical. Third, an exposure-aware thresholding as a learning problem: Eq. (7) is a parametric guess and the natural generalisation is to learn the threshold as a function of graph features under an explicit false-alarm budget as in [2]. Fourth, counterfactual validation: Rather than asking whether an entire batch is anomalous, ask whether the entire downstream output would differ if the batch were correct – this will only be possible when there is a cheap way to re-execute the entire batch and requires only the second step to catch silently within-tolerance errors. Fifth, cross-layer correlation of execution logs: Logs already have identification information [5] and the join between the log identifiers and the lineage nodes is the least-exploited signal in the architecture, so it is the cheapest cross-layer correlation.

14. Conclusion

Twenty-three sources that mentioned provenance, streaming detection, quality verification, and distributed execution were reviewed, ranked, and then a layered architecture was proposed to couple these capabilities together by a single execution-attributed lineage graph. Review key findings are: components are mature, but coupling is not: detection on labelled periodic series [1] gets $F1 > 0.77$, which is 47.9 points higher than random detection on a public streaming detection benchmark [6]; validation benefits from improved interactive latency of 100,000 rows with a declared false positive budget of 0.01% [2] and average repair $F1 > 0.8$ when heterogeneous evidence is combined [12]; the use of provenance graphs as a detection substrate at host level is successful [23]. All of them fail to give an answer to the question which job caused this defect and which consumers are affected, in one structure with one alert channel.

The framework's answer is architectural rather than algorithmic, and it is deliberately modest: separate datasets from jobs, attribute every edge to an execution, modulate the alerting boundary by downstream exposure, score quality over six dimensions with hard and soft classes, and localise causes by ranking ancestors with a depth discount. These design decisions are testable hypotheses that require empirical evaluation and may be rejected if the results do not support the stated claims. Because the framework has not been implemented or experimentally evaluated, this study does not claim that it outperforms existing approaches. It claims only that the gap is real, that the gap is a coupling gap rather than a component gap, and that the design in Fig. 1 and Eq. (1)–(8) is a coherent way to close it with Table 7 as the specification of what would have to be measured before the claim could be made.

References

- [1] H. Ren, B. Xu, Y. Wang, C. Yi, C. Huang, X. Kou, T. Xing, M. Yang, J. Tong, and Q. Zhang, "Time-series anomaly detection service at Microsoft," in Proc. 25th ACM SIGKDD Conf. Knowledge Discovery & Data Mining (KDD '19), Anchorage, AK, USA, 2019, pp. 3009–3017. doi: 10.1145/3292500.3330680.
- [2] D. Tu, Y. He, W. Cui, S. Ge, H. Zhang, H. Shi, D. Zhang, and S. Chaudhuri, "Auto-validate by-history: Auto-program data quality constraints to validate recurring data pipelines," in Proc. 29th ACM SIGKDD Conf. Knowledge Discovery & Data Mining (KDD '23), 2023. doi: 10.1145/3580305.3599776.
- [3] E. Widad, E. Saida, and Y. Gahi, "Quality anomaly detection using predictive techniques: An extensive big data quality framework for reliable data analysis," IEEE Access, vol. 11, 2023. doi: 10.1109/ACCESS.2023.3317354.

- [4] K. R. Chirumamilla, "Predicting data contract failures using machine learning," *The Eastasouth Journal of Information System and Computer Science*, vol. 1, no. 1, pp. 144–155, Aug. 2023.
- [5] K. Ahmed, A. Altaf, N. S. Mohd Jamail, F. Iqbal, and R. Latif, "ADAL-NN: Anomaly detection and localization using deep relational learning in distributed systems," *Applied Sciences*, vol. 13, no. 12, art. 7297, 2023. doi: 10.3390/app13127297.
- [6] A. Lavin and S. Ahmad, "Evaluating real-time anomaly detection algorithms the Numenta Anomaly Benchmark," in *Proc. 14th Int. Conf. Machine Learning and Applications (ICMLA)*, Miami, FL, USA, 2015. Preprint: arXiv:1510.03336.
- [7] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, "Data management challenges in production machine learning," in *Proc. ACM SIGMOD Int. Conf. Management of Data (tutorial)*, 2017. doi: 10.1145/3035918.3054782.
- [8] S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Biessmann, and A. Grafberger, "Automating large-scale data quality verification," *Proc. VLDB Endowment*, vol. 11, no. 12, 2018. doi: 10.14778/3229863.3229867.
- [9] M. Franklin, A. Halevy, and D. Maier, "From databases to dataspace: A new abstraction for information management," in *Proc. ACM SIGMOD Record / CIDR 2005*, 2005.
- [10] L. Ehrlinger and W. Wöß, "A survey of data quality measurement and monitoring tools," *Frontiers in Big Data*, vol. 5, art. 850611, Mar. 2022. doi: 10.3389/fdata.2022.850611.
- [11] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. 9th USENIX Symp. Networked Systems Design and Implementation (NSDI)*, 2012.
- [12] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré, "HoloClean: Holistic data repairs with probabilistic inference," *Proc. VLDB Endowment*, vol. 10, no. 11, pp. 1190–1201, 2017. doi: 10.14778/3137628.3137631.
- [13] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer, "Provenance-aware storage systems," in *Proc. USENIX Annual Technical Conference*, 2006.
- [14] S. B. Davidson and J. Freire, "Provenance and scientific workflows: Challenges and opportunities," in *Proc. ACM SIGMOD Int. Conf. Management of Data*, 2008, pp. 1345–1350. doi: 10.1145/1376616.1376772.
- [15] Y. L. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance in e-science," *ACM SIGMOD Record*, vol. 34, no. 3, 2005. doi: 10.1145/1084805.1084812.
- [16] R. Bose and J. Frew, "Lineage retrieval for scientific data processing: A survey," *ACM Computing Surveys*, vol. 37, no. 1, pp. 1–28, Mar. 2005. doi: 10.1145/1057977.1057978.
- [17] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache Spark: A unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, 2016. doi: 10.1145/2934664.
- [18] M. Armbrust, A. Ghodsi, R. Xin, and M. Zaharia, "Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics," in *Proc. 11th Conf. Innovative Data Systems Research (CIDR)*, 2021.
- [19] P. Carbone, S. Ewen, S. Haridi, A. Katsifodimos, V. Markl, and K. Tzoumas, "Apache Flink: Stream and batch processing in a single engine," *IEEE Data Engineering Bulletin*, vol. 38, no. 4, pp. 28–38, 2015.
- [20] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in *Proc. NetDB Workshop*, 2011.
- [21] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th USENIX Symp. Operating Systems Design and Implementation (OSDI)*, 2004.
- [22] Z. Abedjan, X. Chu, D. Deng, R. Castro Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang, "Detecting data errors: Where are we and what needs to be done?," *Proc. VLDB Endowment*, vol. 9, no. 12, pp. 993–1004, 2016. doi: 10.14778/2994509.2994518.
- [23] M. Zipperle, F. Gottwalt, E. Chang, and T. Dillon, "Provenance-based intrusion detection systems: A survey," *ACM Computing Surveys*, vol. 55, no. 7, 2022/2023. doi: 10.1145/3539605.
- [24] U. Allala, "Predictive Incident Management Using Observability Data in Cloud-Native Systems", *IJETMR*, vol. 3, no. 1, pp. 01–19, Mar. 2020, doi: 10.67228/30715636/IJETMR-2020PI2V8C.
- [25] Gajula, S. (2024). Cybersecurity risk prediction using graph neural networks. *Journal of Information Systems Engineering and Management*.