International Journal of Advanced Artificial Intelligence Research (IJAAIR)
eISSN: 3087-4270
Volume03 Issue10 | Page No: - 13-24

Received: 24 Aug 2026, Revised: 11 Sep 2026, Accepted: 20 Sep 2026, Published: 27 Oct 2026

DOI: https://doi.org/10.55640/ijaair-v03i10-03

Policy-As-Code Governance For Bronze-Layer Ingestion In Regulated Databricks Medallion Lakehouse Architectures

Shikha Sharma
Independent Researcher, Arlington, VA, USA
Email: **shikhasharma78916@gmail.com**

ABSTRACT: Modern data platforms widely use the medallion architecture to organize pipelines on lakehouse systems such as Databricks. Governance controls, including schema validation, sensitive-data handling, retention, lineage tracking, quality checks, and access management, are typically applied in downstream layers or through manual processes. This leaves the bronze layer, where raw data first enters the platform, inconsistently governed. In regulated sectors, a lapse at this layer, such as an unhandled schema change or unmasked sensitive field, can spread across the pipeline and create compliance, financial, and operational risk, because downstream analytics and machine learning depend on the integrity of this raw foundation. This paper reviews and synthesizes existing governance, data quality, and policy-as-code approaches, then proposes a unifying framework that expresses governance requirements as declarative, version-controlled policies enforced automatically at bronze ingestion. The framework combines a policy repository, an enforcement engine with allow, warn, quarantine, and reject outcomes, and an append-only audit store, using Databricks-native tools and general policy engines. The synthesis shows that six governance concerns, namely schema, sensitive data, retention, lineage, quality, and access, can be coordinated by a single policy layer in which each concern is handled by the mechanism suited to it: an ingestion-time gate for schema, sensitive data, and quality; a lifecycle policy for retention; metadata capture for lineage; and access control applied at query and access time. The contribution is a unified policy layer that coordinates these six concerns with continuous auditing, rather than forcing all of them through one enforcement point. Novelty is claimed relative to the approaches examined in this review.

KEYWORDS—Data governance, policy-as-code, Databricks medallion architecture, bronze-layer ingestion, data quality, data lineage, auditability, regulated data pipelines.

1. Introduction

Data engineering on modern lakehouse platforms increasingly relies on the medallion architecture to organize data pipelines into distinct bronze, silver, and gold layers over transactional table formats such as Delta Lake [8], consistent with the lakehouse paradigm [2], [9]. This architectural pattern, widely adopted on systems such as Databricks, supports modular data transformations and traceability across regulated sectors [6], [7]. Within this context, the integration of data governance, data quality, and policy-as-code practices, which originated in cloud and infrastructure operations, is increasingly important [4]. Effective data management in these environments requires robust controls from the initial ingestion point.

However, governance controls, including schema validation, sensitive-data handling, data retention, lineage tracking, quality checks, and access management, are commonly applied in downstream layers or managed through manual processes [3]. This approach leaves the bronze layer, where raw data first enters the platform, inconsistently governed. The bronze layer serves as the foundational input for all subsequent layers, so any lapse in governance at this stage can propagate throughout the entire pipeline. In regulated settings such as insurance and clinical data, an unhandled schema change or an

unmasked sensitive field at ingestion can lead to significant compliance, financial, and operational risks [5], [7].

Existing work often addresses data quality, privacy, lineage, and access as separate concerns, employing disparate tools and applying controls at various stages, frequently after data has already been ingested [3]. While policy-as-code has seen adoption for infrastructure and access control, and more recently for the governance of AI systems and AI-assisted engineering [4], [18], [19], it has not been consolidated into a unified governance layer enforced directly at data ingestion. As lakehouse adoption continues to grow and regulatory expectations tighten, the absence of a cohesive, automated, and auditable governance approach at the bronze layer represents a notable gap.

This paper makes the following contributions:

- A review and synthesis of existing governance, data quality, and policy-as-code approaches, showing how their controls map onto the bronze ingestion layer of a medallion architecture.
- A unifying framework that expresses six governance concerns, namely schema, sensitive data, retention, lineage, quality, and access, as declarative, version-controlled policies coordinated by a single policy layer, each enforced by the mechanism suited to it, with allow, warn, quarantine, and reject outcomes for the gating concerns and append-only audit logging.
- A set of integration patterns that map the framework onto Databricks-native mechanisms (Delta Lake, Delta Live Tables, Unity Catalog) and general policy engines, alongside a discussion of benefits, trade-offs, and limitations.

The remainder of this paper details the proposed framework, its implementation considerations, and a discussion of its implications for data governance in regulated lakehouse environments.

This work is guided by three questions: (RQ1) which governance dimensions can reasonably be enforced at the point of bronze ingestion; (RQ2) how can those dimensions be expressed as declarative, version-controlled policy-as-code; and (RQ3) which Databricks-native mechanisms can implement the resulting framework. The paper addresses these questions through an architectural synthesis rather than an empirical study, and its contribution should be read as a reference framework that consolidates existing governance techniques at a new enforcement point, not as an entirely new governance technology.

2. Literature Review

Medallion architecture and lakehouse ingestion. The medallion pattern organizes pipelines into bronze, silver, and gold layers, with the bronze layer acting as the raw landing zone under schema-on-read assumptions [8], [9]. The pattern and the broader lakehouse model have been examined in comparative reviews and surveys [10], [12], applied to sensitive domains such as biomedical research [11], and analyzed across competing storage systems [13]. Brahmandam's empirical evaluation on Databricks characterizes throughput, latency, and cost across batch and real-time ingestion, establishing the pattern's practical viability but treating governance as outside its scope [2]. Kolla et al. address the bronze-to-silver stage directly, replacing per-table notebooks with a metadata-driven framework that applies declarative quality rules and separates valid from quarantined records [3]. The two differ in where control sits: Brahmandam measures ingestion performance and leaves data control implicit, while Kolla et al. locate quality enforcement at the transformation step. Both, however, act after raw data has already landed in bronze, leaving the ingestion point itself ungoverned.

Data governance and quality in lakehouse platforms. Moncivais presents Unity Catalog as a unified governance framework that enforces cataloging, access, and lineage policies in-engine, at query and access time [1]. This resolves governance consistency at query and access time, but not at ingestion, so raw data can enter bronze before any policy is applied. This contrasts with Kolla et al., where enforcement happens during transformation rather than at query time [3]. The tension across these two is instructive: one governs at consumption, the other at curation, and neither governs at entry. In both cases, issues such as an unhandled schema change or an exposed sensitive field can propagate before they are detected.

Policy-as-code and declarative enforcement. Policy-as-code originated in cloud and infrastructure operations as a declarative, version-controlled way to manage configuration and access, valued for testability and auditability. General-purpose engines such as Open Policy

Agent evaluate declarative rules to make automated decisions across the stack [14], [15], and the same practice is increasingly applied to enterprise data governance and operations [16], [17]. Madan's ArGen extends the paradigm to generative AI, aligning model behavior with machine-readable rules through an Open Policy Agent-style governance layer [4]. What ArGen governs, however, is the behavior and output of a model, not data at rest as it enters a platform. The paradigm has therefore been applied to infrastructure, access, and model conduct, but not to the governance of raw data at ingestion, which is a different object of control with different enforcement points.

Governance in regulated enterprise data environments. Qiu's cross-system reconciliation work targets regulated enterprises where fragmented, non-interoperable systems produce unreconciled records and weak audit-readiness, proposing a layered architecture spanning ingestion, staging, core models, and semantic serving [5]. Fannouch et al. take a comparable stance for healthcare, using a governed medallion lakehouse with explicit promotion contracts between layers [7]. Both deliver strong governance, but they distribute controls across multiple layers or external frameworks rather than concentrating them at the entry point. Their comprehensiveness comes at the cost of a single, enforceable point of control for raw data.

Research gap. Read together, these strands share a boundary. Governance is enforced at query time [1], at transformation [3], or across multiple architectural layers [5], [7], and policy-as-code has been applied to infrastructure and model behavior [4] but not to raw data as it lands. Quality, privacy, lineage, and access remain separate concerns handled by different mechanisms at different stages. What none of these approaches provides is a single, declarative, version-controlled policy layer that consolidates these concerns and enforces them automatically at the moment of bronze ingestion, where a control has the widest downstream effect. That consolidation is what this paper contributes.

3. Problem Statement

We formalize governance enforcement at the bronze ingestion layer of a medallion architecture as follows.

Inputs. A stream or batch of incoming records D arriving at the bronze layer from a source feed s, together with its associated metadata M (source identity, declared schema, arrival time, and column-level classification). A set of governance policies P covering six concerns: schema, sensitive data, retention, lineage, quality, and access.

Outputs. For each record or batch, a governance decision d that is one of allow, warn, quarantine, or reject, the resulting write to either the bronze table or a quarantine location, and an append-only audit entry recording the policy evaluated, the outcome, and the affected data.

Constraints. Policies must be declarative and version-controlled, evaluated automatically without manual intervention. Enforcement must occur at ingestion, before data is admitted to the bronze table, not at a later transformation or query stage. Failing records must be handled without halting the pipeline or breaching its service-level objectives, that is, graceful quarantine rather than a full stop. Every decision must be auditable, producing continuous compliance evidence.

Objective. Maximize governance coverage and consistency at the ingestion point, that is, the share of the six concerns enforced automatically at bronze across all feeds, while minimizing both undetected governance violations that propagate downstream and disruption to ingestion throughput.

Assumptions. The platform is a Databricks lakehouse using Delta tables, so native mechanisms (Delta constraints, Delta Live Tables expectations, Unity Catalog) are available as enforcement points. Source metadata, including a declared or inferable schema and sensitive-data classification, is available or derivable at ingestion time. Governance requirements can be expressed as machine-readable rules. A general policy engine such as Open Policy Agent can be integrated alongside native mechanisms.

Under this formulation, the gap identified in the literature is the absence of a single policy layer that satisfies these constraints at once, evaluating P over (D, M) at the moment of ingestion. The framework proposed in the next section is one realization of a system that meets this specification.

4. Methodology

Research Design

This study employs a conceptual, design-oriented methodology that combines a structured literature

synthesis with a proposed reference framework. It is not an experimental study and does not involve empirical data collection or statistical analysis. The design proceeds in two phases. In the first, a thematic synthesis of existing governance, data quality, and policy-as-code approaches establishes a set of requirements for governance at the bronze layer. In the second, a framework is designed to satisfy those requirements and is illustrated with representative ingestion scenarios mapped onto native Databricks mechanisms. Fig. 1 summarizes this end-to-end process, in which each stage feeds the next from literature synthesis through to qualitative evaluation.

The seed papers were identified by searching Google Scholar, IEEE Xplore, and arXiv in 2026 for the intersection of the medallion architecture, lakehouse governance, data quality, and policy-as-code, using combinations of the terms policy-as-code, data governance, medallion architecture, bronze layer, lakehouse, Delta Lake, and data quality. Papers were retained when they addressed governance or quality at a defined stage of a Databricks or lakehouse pipeline, and this screening retained 16 sources, which are the works reviewed in Section II. The six governance concerns, namely schema, sensitive data, retention, lineage, quality, and access, were derived inductively from the recurring controls named across these sources; concern identification was treated as saturated once additional papers introduced no new control category. These six were selected because each is enforceable, in principle, at the moment data lands in bronze; broader organizational concerns such as data stewardship or business ownership were excluded as out of scope for an ingestion-time layer.

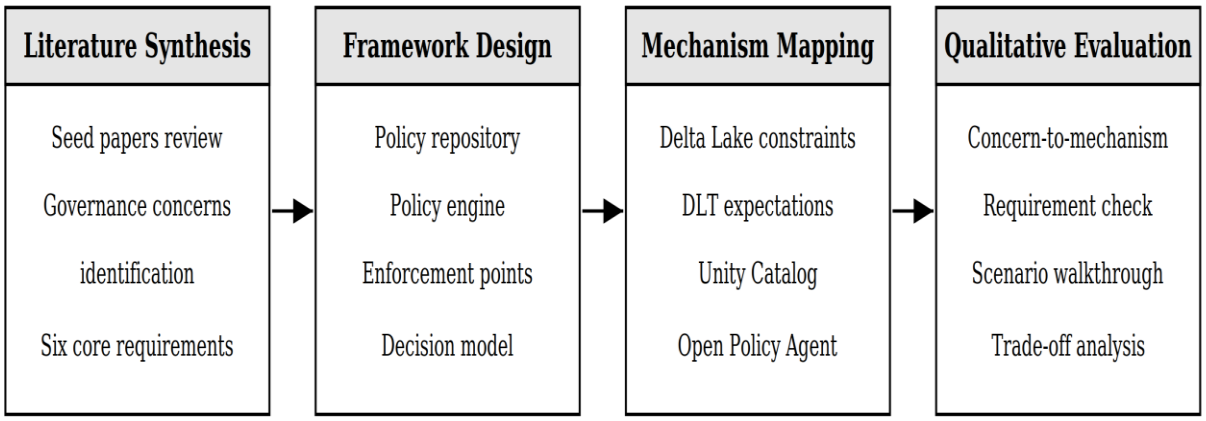


Fig. 1. Methodology overview: literature synthesis, framework design, mechanism mapping, and qualitative evaluation.

Data and Illustrative Scenarios

The sample for this study is the body of reviewed literature, comprising the selected seed papers and related work on medallion architectures, lakehouse governance, and policy-as-code. Using literature and illustrative scenarios rather than production data is appropriate for a design-stage study and avoids the privacy exposure of real or regulated data. To show how the framework operates, three representative bronze ingestion scenarios are used: a source feed with a required identifier field, a feed containing columns that must be classified as sensitive, and a feed with records that violate a defined quality rule. As a concrete illustration, a source feed ingested into the bronze layer requires a non-null record identifier, flags any column classified as sensitive for masking, and routes records that fail these checks to a quarantine table so the main load continues to meet its schedule. Table 1 lists the framework's components and the role, input, and output of each.

Table 1. Framework components, their roles, and their inputs and outputs.

Component	Technology	Role	Input	Output
Policy repository	Git-based version control	Store declarative governance rules	Policy definitions (YAML/Rego)	Versioned policy artifacts
Policy engine	Open Policy Agent	Evaluate structural and metadata rules	Incoming record + policy	Decision (allow/warn/quarantine/reject)
Enforcement points	Delta Live Tables expectations	Apply quality and schema rules at ingestion	Streaming data + expectations	Validated records + violations
Append-only audit store	Delta Lake append-only table	Record all governance decisions	Decision metadata + timestamp	Append-only audit trail
Ingestion pipeline	Apache Spark + Auto Loader	Ingest source feeds into bronze	Raw source data	Structured bronze records
Cataloging and lineage	Unity Catalog	Manage metadata, lineage, and access	Data assets + operations	Unified catalog + lineage graph

Framework Development Process

The framework was developed through the following procedure:

Identify governance concerns. From the literature synthesis, six concerns relevant at bronze ingestion were identified: schema enforcement, sensitive-data handling, data retention, lineage tracking, data quality validation, and access control [3].

Derive requirements. From these concerns, six requirements for a governance layer were derived: declarative, version-controlled, automatically enforced, auditable, non-blocking to ingestion, and portable.

Design the components. Four components were specified: a policy repository holding version-controlled rules, a policy engine that evaluates them, ingestion-time enforcement points embedded in the pipeline, and an append-only audit store that records every decision.

Define the decision model. Each incoming record is resolved to one of four outcomes: allow, warn, quarantine, or reject. This gating model applies to the concerns that govern admission of a record, namely schema, sensitive data, and data quality. Records that fail a non-critical rule are routed to a separate quarantine table or path while the main stream continues, whereas a rule configured as a hard failure, such as a strict expectation or a Delta table constraint, aborts the write for the affected batch rather than quarantining it.

Handle non-gating concerns separately. Retention and lineage are deliberately kept outside the four-outcome model: retention is enforced as a lifecycle policy on stored data, and lineage is captured as metadata on every ingestion event. Making this separation explicit

keeps the decision model accurate rather than forcing every concern into the same outcomes.

Map concerns to mechanisms. Each concern is mapped to a concrete enforcement mechanism in the target platform, described in the next subsection.

Fig. 2 shows the resulting architecture, from source ingestion through the governance layer to the bronze layer and the decision outcomes.

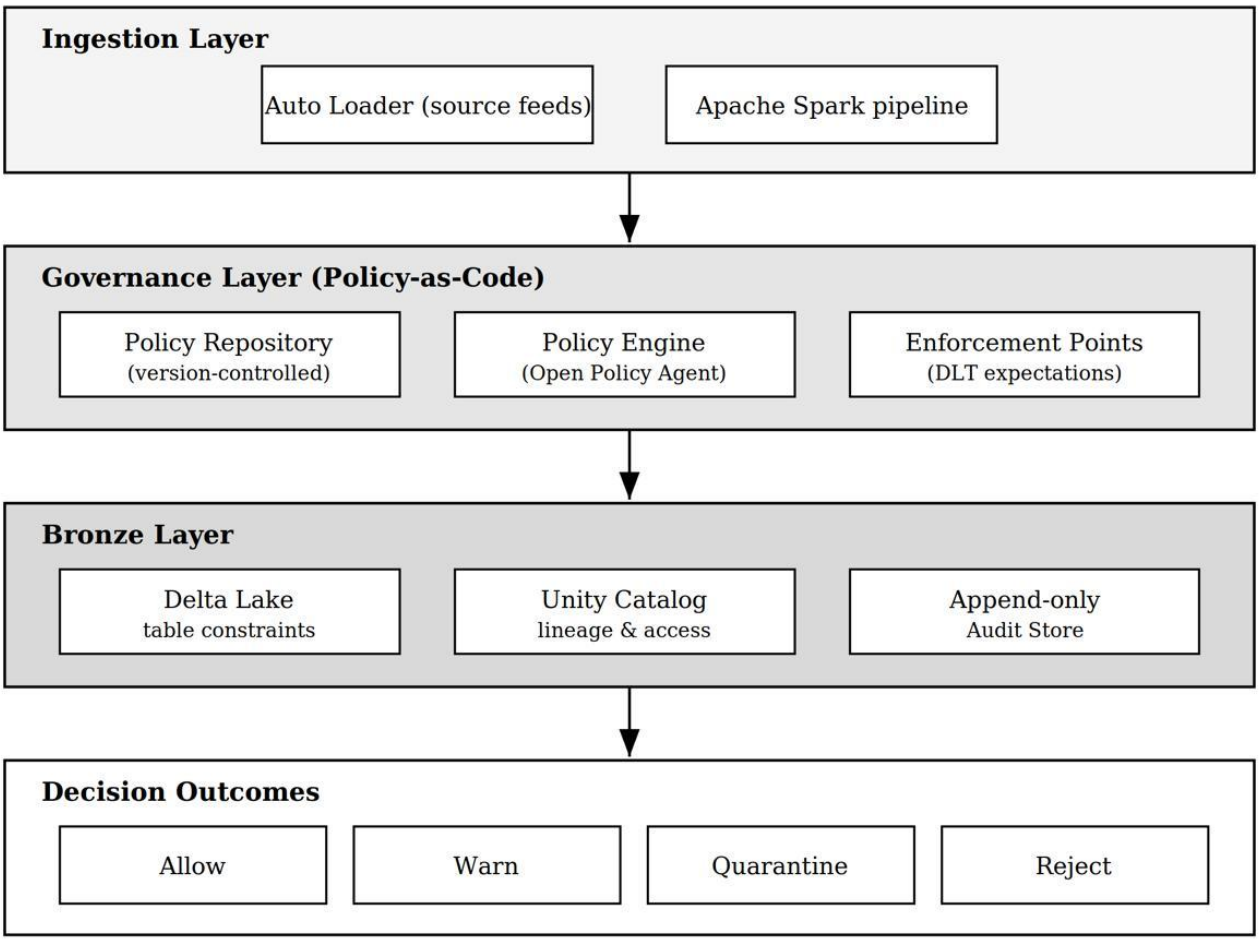


Fig. 2. Framework architecture: ingestion layer, governance layer, bronze layer, and decision outcomes.

Tools and Platform

The framework is described within the Databricks lakehouse platform. The native mechanisms it maps onto are Delta Lake table constraints for schema enforcement and integrity, Delta Live Tables expectations for declarative data quality rules [3], and Unity Catalog for cataloging, automated lineage capture, sensitive-column classification, and fine-grained access control [1]. Unity Catalog applies column masking at query time, so where a value must be hidden before it is written to bronze, masking or tokenization is performed as a separate pre-write step in the ingestion job rather than by the catalog. Open Policy Agent is referenced as a general-purpose engine for structural and metadata rules that sit outside native table mechanisms [4]. Ingestion into the bronze layer is described using Databricks Auto Loader and Apache Spark [2]. As a design study, no dedicated hardware or benchmarking cluster is used; these tools define the target environment onto which the framework is mapped. Table 2 maps each governance concern to its enforcement mechanism, its Databricks component, and whether it acts as a record-level gate, a lifecycle policy, or metadata capture. Retention is enforced as a lifecycle policy rather than a single command: VACUUM only removes files already past the table's retention window, so an explicit expiry or deletion process must set and apply that window for records that reach their retention limit.

Table 2. Mapping of each governance concern to its enforcement mechanism, Databricks component, and enforcement type.

Governance concern	Enforcement mechanism	Databricks component	Enforcement type
Schema enforcement	Table constraints / schema checks	Delta Lake	Record-level gate (reject or quarantine)
Sensitive-data handling	Classification, tagging, column masking	Unity Catalog	Record-level gate (warn or quarantine)
Data quality validation	Expectations	Delta Live Tables	Record-level gate (warn or quarantine)
Data retention	Time travel and vacuum policies	Delta Lake	Lifecycle policy (not a per-record gate)
Lineage tracking	Automated lineage capture	Unity Catalog	Metadata capture (recorded, non-blocking)
Access control	Fine-grained permissions	Unity Catalog	Access-time gate (deny unauthorized)

Qualitative Evaluation

Because this is a design study, the framework is evaluated qualitatively rather than statistically. The evaluation has three parts. First, each of the six governance concerns is mapped to its enforcement mechanism to confirm coverage. Second, the design is checked against the six requirements, verifying that each concern is expressed declaratively, enforced at ingestion, recorded for audit, and handled without blocking the pipeline where appropriate. Third, the illustrative scenarios are walked through to show the outcomes the framework produces. The benefits, architectural trade-offs, and limitations are then discussed. Empirical measurement of ingestion overhead and of downstream defect reduction is identified as future work.

Worked Example

To ground the framework, take a feed called claims_raw, a daily batch that lands in the bronze layer through Auto Loader. Its governance policy lives in the policy repository as a single version-controlled file, shown in Listing 1. That one file carries five of the six concerns outright: the identifier fields a record must have (schema), the two columns that must be masked before landing (sensitive data), and a sanity check on the claim amount (quality), along with the retention window and the lineage fields to capture. Access is the exception, handled by catalog permissions on the table the feed writes to rather than by the ingestion policy itself.

Listing 1. *Declarative bronze-ingestion policy for the claims_raw feed (version-controlled).*

```
# policy: claims_raw.bronze.yaml   (version-controlled)
feed:        claims_raw
enforcement: bronze_ingestion
rules:
  - id: required_fields          # concern: schema
    assert: not_null(claim_id) and not_null(policy_id)
    on_fail: quarantine
  - id: sensitive_columns        # concern: sensitive data
    columns: [ssn, member_dob]
    action: mask
    on_fail: quarantine
  - id: amount_valid             # concern: quality
    assert: is_numeric(claim_amount) and claim_amount >= 0
    on_fail: warn
retention_days: 400              # concern: retention
lineage: capture(source_system, load_id, ingest_ts)
audit:   append(decision, rule_id, record_key, ts)
```

When a batch arrives, the engine runs each record past these rules and settles on one of the four outcomes. A clean record, with its identifiers present, a claim amount that is not negative, and ssn and member_dob already masked, goes straight into the bronze table. Drop the claim_id and the record fails the required-fields rule, so it is peeled off into a quarantine table instead of bronze while the rest of the batch keeps moving and the load still finishes on time. A negative claim_amount is treated more gently. It trips only the amount_valid rule, which is set to warn, so the record still lands but is tagged for someone to review. Whatever the outcome, the engine writes one line to the audit store naming the rule that ran, the decision it made, the record key, and the time, so there is always a running account of how each row was handled rather than one rebuilt after an incident. Quarantining instead of failing the

whole batch is a deliberate, practical choice: the feed keeps its SLA, and governance still happens at the door.

5. Analysis and Discussion

This study presents a design-oriented analysis focused on the architectural consolidation of governance controls, not on empirical performance measurement. The primary design finding is that all six identified governance concerns, namely schema enforcement, sensitive-data handling, data retention, lineage capture, quality validation, and access management, can be coordinated by a single policy-as-code layer in which each concern is handled by the mechanism suited to it rather than all being forced through one enforcement point. The framework is designed to satisfy the six derived requirements: policies are intended to be declarative, version-controlled, automatically enforced, auditable, non-blocking to ingestion, and portable across data assets. These are design properties of the proposed framework, not empirically demonstrated results. In short, six governance concerns are coordinated through one policy layer, in contrast to the reviewed approaches that spread these controls across multiple stages and tools. Fig. 3 summarizes the evaluation workflow, and Table 3 lists the intended design property behind each requirement, none of which is an empirically demonstrated result.

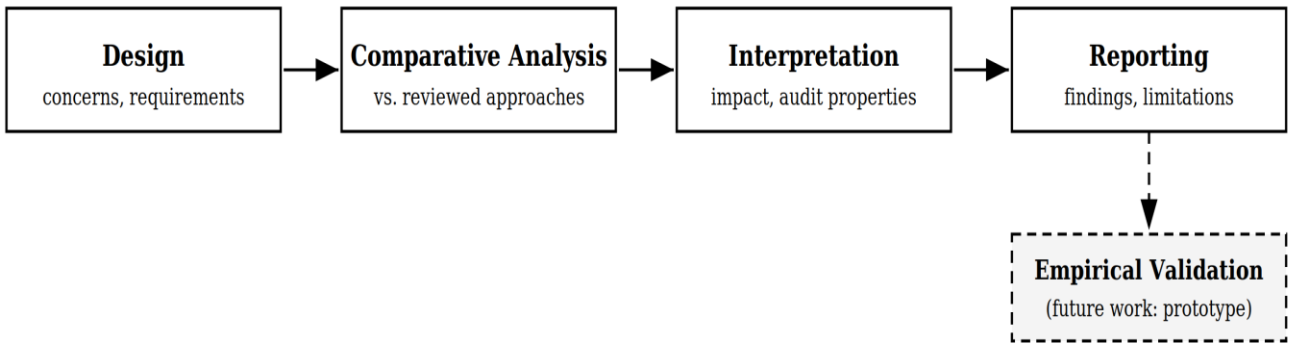


Fig. 3. Evaluation workflow, with empirical validation identified as future work.

Table 3. Verification of each requirement against its implementation mechanism.

Requirement	Implementation mechanism	Concerns addressed	Intended design property
Declarative	YAML/Rego policy specifications	All six	Policies expressed as configuration, not procedural code
Version-controlled	Git-based policy repository	All six	Changes tracked with commit history and review
Automatically enforced	Pre-ingestion validation	Schema, sensitive data, quality	Policies applied before data enters bronze
Auditable	Append-only audit store	All six	Every decision recorded with timestamp and policy version
Non-blocking	Quarantine-and-flag on failure	Quality, lineage	Defective records routed aside; ingestion continues
Portable	Template-based policy inheritance	All six	Policies defined once, applied across sources

Comparison to Existing Approaches

The framework differs from current approaches mainly in its point of enforcement and its combined scope. Unity Catalog provides unified governance but enforces primarily at query time [1]. The metadata-driven framework of Kolla et al. applies quality rules during the bronze-to-silver transformation rather than at entry into bronze [3]. Architectures for regulated enterprises distribute controls across several layers, as in the cross-system reconciliation framework [5] and the governed DataOps architecture for healthcare [7]. Policy-as-code work such as ArGen targets model behavior rather than data at rest [4]. As Table 4 shows, some of these match the proposed framework on coverage, but none coordinates all six concerns through a single declarative policy layer. The contribution is therefore not broader coverage alone but the coordination of these controls through one policy layer, with each concern enforced at the stage suited to it, so that governance stays consistent and auditable across the pipeline.

Table 4. Comparison of governance coverage and enforcement point across the proposed framework and reviewed approaches.

Approach	Schema	Sensitive data	Retention	Lineage	Quality	Access	Enforcement point	Policy model
Proposed framework	Y	Y	Y	Y	Y	Y	Coordinated layer	Declarative, version-controlled
Unity Catalog [1]	Y	Y	—	Y	—	Y	Query time	Centralized catalog
Kolla et al. [3]	Y	—	—	—	Y	—	Bronze-to-silver	Metadata-driven rules
Cross-system framework [5]	Y	Y	Y	Y	Y	Y	Distributed across layers	Multi-layer controls
Governed DataOps [7]	Y	Y	Y	Y	Y	Y	Distributed across layers	Workflow-integrated
ArGen [4]	—	—	—	—	—	—	Model behavior	Policy-as-code (ML)

Note: *Y indicates the concern is covered; — indicates it is not.*

Interpretation of Findings

Putting the checks at bronze ingestion means a bad record is caught where it enters, before it can seep into silver, gold, and whatever analytics or models sit on top of them. Because every decision is written down as it happens, the compliance evidence is already there when someone asks for it, instead of being pieced back together after an

incident. Keeping the rules as version-controlled code has a quieter benefit as well: a governance change goes through the same review and history as any other code change, rather than living in one person's notebook. For a team in a regulated setting, this turns governance from something done by hand, late, and downstream into a property the ingestion layer simply has.

Limitations

Several limitations apply. First, the framework is evaluated conceptually and has not been validated empirically, so its runtime overhead and its effect on downstream defect rates are not yet measured. Second, the synthesis rests on a limited set of seed papers and related work and may not capture every relevant approach. Third, the mechanism mapping is specific to the Databricks lakehouse; portability to other platforms is argued but not demonstrated. Fourth, expressing some governance concerns, particularly nuanced sensitive-data rules, as fully declarative policy may be harder in practice than the design suggests. These points define the future work: a prototype implementation and empirical measurement of ingestion overhead and defect reduction on representative feeds.

6. Conclusion and Future Work

This paper reviewed existing governance, data quality, and policy-as-code approaches and proposed a unifying framework that consolidates six governance concerns, namely schema, sensitive data, retention, lineage, quality, and access, into a single declarative and version-controlled policy layer that coordinates them across a Databricks medallion architecture, with every decision recorded in an append-only audit trail. The central finding is analytical rather than empirical: the six concerns can be coordinated through one policy layer, each enforced by the mechanism suited to it, shifting controls from reactive, fragmented, downstream processes toward a consistent and auditable property of the pipeline. This matters because analytics and machine learning are only as reliable as the raw data beneath them, and in regulated settings an undetected defect at ingestion carries real compliance and operational cost. Two directions follow from this work: a prototype implementation with empirical measurement of ingestion overhead and downstream defect reduction, and an extension of the policy-as-code layer beyond bronze to the silver and gold stages. Governing raw data as code, at the point it enters the platform, offers a practical foundation for trustworthy data in the systems that depend on it.

References

1. I. Moncivais, "Databricks Unity Catalog: A Unified Governance Framework for Enterprise Data and AI Platforms," 2026. DOI: 10.2139/ssrn.6905498.

2. L. M. K. Brahmandam, "An Empirical Evaluation of the Medallion Architecture on Databricks and Apache Spark with Snowflake: Throughput, Latency, and Cost for Batch and Real-Time Ingestion Patterns," Int. J. AI, BigData, Computational and Management Studies, vol. 5, no. 3, pp. 197–206, 2024. DOI: 10.63282/3050-9416.ijaibdcms-v5i3p122.

3. A. K. Kolla, A. Kokate, R. Vellanki, R. Lolam, and P. Bandi, "A Metadata-Driven Framework for Automated Bronze-to-Silver Data Ingestion, Validation and Loading on Databricks," European J. Information Technologies and Computer Science, vol. 6, no. 3, pp. 1–7, 2026. DOI: 10.24018/compute.2026.6.3.70578.

4. K. Madan, "ArGen: Auto-Regulation of Generative AI via GRPO and Policy-as-Code," arXiv:2509.07006, 2025.

5. Z. Qiu, "Data Engineering Patterns for Cross-System Reconciliation in Regulated Enterprises: Architecture, Anomaly Detection, and Governance," arXiv:2604.15108, 2026.

6. H. A. Mohna, T. Barua, M. Mohiuddin, and M. M. Rahman, "AI-Ready Data Engineering Pipelines: A Review of Medallion Architecture and Cloud-Based Integration Models," American J. Scholarly Research and Innovation, vol. 1, no. 1, pp. 319–350, 2022. DOI: 10.63125/51kxtf08.

7. A. Fannouch, J. Gharib, and Y. Gahi, "A Governed Lakehouse DataOps Architecture: Design and Evaluation in Healthcare," HighTech and Innovation Journal, 2026. DOI: 10.28991/hij-2026-07-02-021.

8. M. Armbrust, T. Das, S. Paranjpye, R. Xin, S. Zhu, A. Ghodsi, et al., "Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores,"

Proc. VLDB Endowment, vol. 13, no. 12, pp. 3411-3424, 2020. DOI: 10.14778/3415478.3415560.

9. 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. Conf. Innovative Data Systems Research (CIDR), 2021. [Online]. Available: https://www.cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf.

10. A. A. Harby and F. Zulkernine, "From Data Warehouse to Lakehouse: A Comparative Review," in Proc. IEEE Int. Conf. Big Data (Big Data), 2022, pp. 389-395. DOI: 10.1109/BigData55660.2022.10020719.

11. E. Begoli, I. Goethert, and K. Knight, "A Lakehouse Architecture for the Management and Analysis of Heterogeneous Data for Biomedical Research and Mega-biobanks," in Proc. IEEE Int. Conf. Big Data (Big Data), 2021, pp. 4643-4651.

12. A. A. Harby and F. Zulkernine, "Data Lakehouse: A Survey and Experimental Study," Information Systems, vol. 127, art. 102460, 2025. DOI: 10.1016/j.is.2024.102460.

13. P. Jain, P. Kraft, C. Power, T. Das, I. Stoica, and M. Zaharia, "Analyzing and Comparing Lakehouse Storage Systems," in Proc. Conf. Innovative Data Systems Research (CIDR), 2023.

14. Open Policy Agent, "Open Policy Agent: Policy-based control for cloud native environments," Cloud Native Computing Foundation. [Online]. Available: https://www.openpolicyagent.org/. Accessed: 2026.

15. Amazon Web Services, "Governing Infrastructure as Code Using Pattern-Based Policy as Code," AWS Security Blog, 2026. [Online]. Available: https://aws.amazon.com/blogs/security/governing-infrastructure-as-code-using-pattern-based-policy-as-code/.

16. I. Moncivais, "Data Architecture and Governance in the Agentic Era: Paradigm Shifts from 2026 Databricks DAIS," 2026. DOI: 10.2139/ssrn.6979203.

17. Acceldata, "What Does Policy-as-Code Look Like for Enterprise Data Governance," 2026. [Online]. Available: https://www.acceldata.io/blog/what-does-policy-as-code-look-like-for-enterprise-data-governance.

18. R. Kang, "Governed AI-Assisted Engineering: Graduated Human Oversight for Agentic Code Generation in Regulated Domains," arXiv:2606.22484, 2026.

19. H. Morgan, "AI Governance Control Stack for Operational Stability: Achieving Hardened Governance in AI Systems," arXiv:2604.03262, 2026.