International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(22s) 2026 pp. 1210-1220
https://doi.org/10.70917/ijcisim-2026-5512
Article

Designing a Cloud-Native Lakehouse Architecture for Real-Time and Batch Enterprise Data Intelligence

Pavan Kumar Kodati

Independent Researcher, USA
ORCID ID: 0009-0000-5372-1504

Abstract: Enterprises require unified platforms capable of supporting real-time data ingestion and large-scale batch analytics within the same operational environment. Traditional architectures separate data lakes, data warehouses, and streaming systems, producing duplicated storage, fragmented governance, inconsistent data semantics, and escalating infrastructure costs. This article proposes a cloud-native lakehouse architecture that converges real-time and batch data processing within a single scalable framework. The architecture integrates six functional layers: ingestion, storage with transactional table formats, distributed processing, semantic serving, governance, and workload optimization. A medallion data organization model structures the progression of data through three quality zones, raw ingestion, conformed and validated datasets, and curated business-ready outputs, from source to analytical consumption. The proposed architecture is evaluated using metrics including ingestion throughput, end-to-end latency, query performance, storage efficiency, cost per workload, and schema evolution ease. Expected benefits include reduced data duplication, lower operational overhead from unified governance, more consistent data quality across workloads, and improved support for artificial intelligence and machine learning pipelines. The contribution is a reference architecture and implementation blueprint for enterprise-scale data intelligence that unifies streaming and batch processing under a consistent storage, governance, and serving model.

Keywords: batch analytics, cloud-native architecture, data governance, data lakehouse, medallion architecture, real-time ingestion, streaming analytics, transactional table formats

1. Introduction

Data-intensive enterprises no longer operate in a purely batch-oriented world. Customer events, operational telemetry, Internet of Things (IoT) streams, application programming interface (API) interactions, and transactional changes generate continuous data flows that must integrate with historical datasets for analytics, reporting, and machine learning (ML). Many organizations have responded by layering new technologies onto legacy data warehouses, producing disconnected systems for streaming ingestion, batch processing, business intelligence (BI) reporting, and ML feature generation that share no common governance, no common metadata model, and no common operational surface [1].

The fragmentation carries a cost that compounds over time. Data duplication increases storage spend and reconciliation effort. Governance becomes inconsistent; access policies, quality rules, and lineage tracking apply differently across the warehouse, the lake, and the streaming layer. Latency-sensitive workloads compete with analytical queries for shared compute. Development teams maintain multiple pipeline implementations for the same business entity, each with its failure modes. As data volumes and consumption patterns grow, these inefficiencies limit the ability of organizations to deliver timely and trustworthy intelligence [2].

The lakehouse model addresses this fragmentation directly. It combines the cost and scalability characteristics of cloud object storage with the reliability and governance characteristics of a data warehouse through transactional table formats that enforce atomicity, consistency, isolation, and durability (ACID) guarantees over standard object storage. The result is not simply a new storage pattern. It is an operating model for ingestion, curation, serving, governance, and workload optimization across both streaming and batch workloads without requiring organizations to maintain separate platforms for each [3].

The individual technologies that enable the lakehouse have matured substantially. Transactional table formats demonstrate ACID-compliant operations at petabyte scale over cloud object storage [1][4]. Stream processing frameworks support declarative, fault-tolerant real-time transformations within the same compute environment used for batch jobs [5]. Cloud-native database architectures have established patterns for elastic, multi-tenant analytical workloads at production scale [7]. What organizations lack is a coherent reference architecture that integrates these components as a unified system, one that addresses ingestion, storage, processing, serving, governance, and optimization together rather than leaving each layer to be independently designed and operated [12].

This article proposes that reference architecture. The contribution is fourfold: a six-layer architectural model for cloud-native lakehouse deployment; a medallion data organization pattern that enforces progressive data quality from ingestion to consumption; an implementation blueprint covering key engineering decisions at each layer; and an evaluation framework for assessing the architecture against fragmented legacy alternatives. The article proceeds as follows: Section 2 examines the background and related work; Section 3 presents the proposed architecture; Section 4 describes the implementation blueprint and evaluation; Section 5 discusses strategic value and limitations; Section 6 concludes.

2. Background and Related Work

Data warehouses provided enterprises with the first systematic approach to analytical data management, centralized schema control, optimized query execution, and structured governance for structured, slowly evolving data. The model served well for batch reporting workloads but proved ill-suited to raw, semi-structured, and high-velocity data. As data volumes grew and data variety expanded, the rigid schema requirements and high storage cost of the warehouse became architectural constraints rather than engineering choices [6].

Data lakes addressed scale and flexibility by storing data in open formats on low-cost distributed storage, deferring schema enforcement to query time. The flexibility proved real; the reliability did not. Without enforced data contracts, consistent quality rules, or transactional guarantees, data lakes accumulated raw data that was expensive to query reliably and difficult to govern. Concurrent reads and writes without isolation produced inconsistent results. The absence of version history complicated audit and compliance requirements. Practitioners named the outcome the "data swamp," a storage resource that accumulated data faster than it could be made trustworthy [3].

The lakehouse concept closes this gap. Transactional metadata formats layered over cloud object storage deliver ACID guarantees at lake-scale cost. Delta Lake demonstrated the concept at production scale, and subsequent work has extended ACID-compliant row-level operations to petabyte-scale datasets under production conditions [1][4]. A comprehensive survey of lakehouse architectures confirms that this design pattern has become the reference approach for building modern enterprise data platforms, with measurable advantages in governance consistency, query performance, and operational overhead compared to separated lake-plus-warehouse architectures [12].

Stream processing has followed a parallel trajectory. Early distributed batch systems introduced scalable data processing but imposed high latency between data arrival and analytical availability [13]. Later frameworks introduced stateful stream processing with exactly-once guarantees and unified programming models that treat batch and streaming as a single computational abstraction [5][9]. Cloud-native stream management systems now demonstrate production-scale streaming workload management for major internet platforms, establishing that streaming and batch can share infrastructure, governance, and operational tooling without sacrificing performance in either dimension [10].

3. Method: Proposed Lakehouse Architecture

The proposed architecture defines six layers that operate as an integrated system: ingestion, storage, processing, semantic serving, governance, and optimization. The integration is the point; these layers share metadata registries, governance rule engines, lineage capture infrastructure, and operational instrumentation. This shared foundation is what distinguishes the lakehouse from a collection of independently operated platforms that happen to exchange data [2].

The ingestion layer supports two arrival paths. The real-time path handles event streams, change data capture (CDC) feeds, API webhooks, and IoT telemetry through messaging infrastructure. The batch path handles periodic extracts from operational systems, file arrivals, and bulk API pulls. Both paths land data in a raw storage zone with immutable retention policies; source data is preserved without modification, enabling replay and reprocessing as downstream transformation logic evolves. Separating ingestion from processing preserves source fidelity and allows ingestion throughput and transformation capacity to scale independently [5].

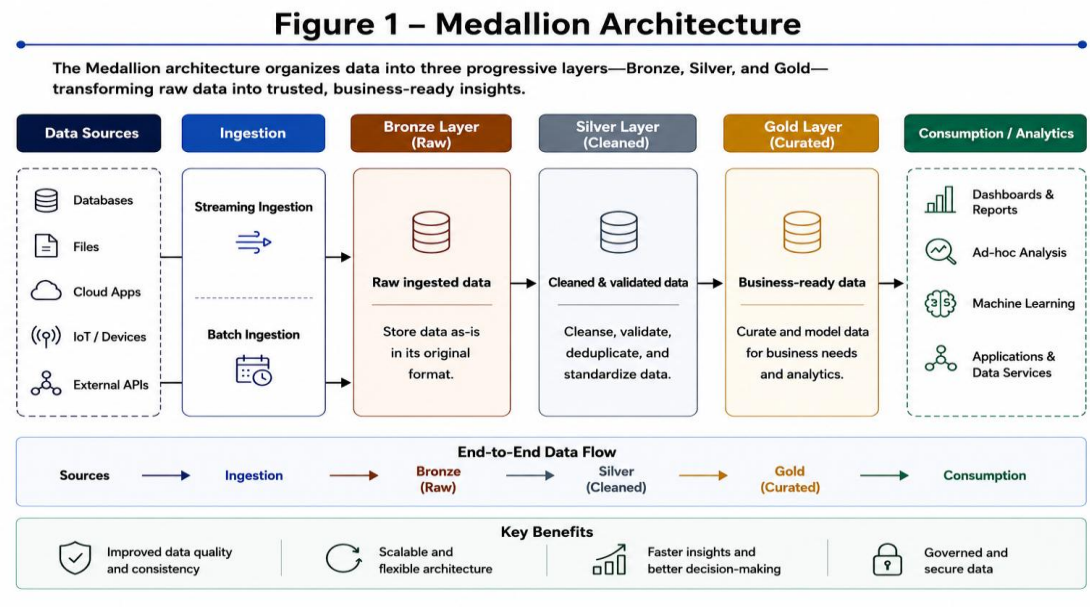

Figure 1. Medallion data organization model showing progressive quality zones from raw ingestion to curated business-ready datasets.

The storage layer uses cloud object storage as the platform foundation, with a transactional table format managing table state, schema evolution, partitioning, and version history across all medallion zones. The medallion model organizes data into three quality zones. Bronze tables preserve source-faithful raw data with minimal transformation; duplicates are permitted, and the schema is permissive. Silver tables apply deduplication, schema conformation to organizational data models, business rule validation, and null boundary enforcement. Gold tables contain curated, domain-aligned datasets built for specific consumption patterns and carrying defined quality contracts and freshness commitments. Promotion from bronze to silver and silver to gold is gated on quality rule validation, ensuring that downstream consumers receive data that meets their declared expectations [3].

The processing layer provides distributed compute for medallion transitions. Stream jobs process real-time data with micro-batch or continuous execution, applying the same transformation logic as batch equivalents where business rules permit. Batch jobs handle large-scale historical reprocessing, incremental merge operations, and full partition refreshes. Sharing metadata registries, governance rule engines, and transformation libraries between stream and batch jobs reduces code duplication and ensures consistent business logic across data arrival patterns, eliminating the class of data quality incidents caused by divergent implementations of the same rule in stream and batch code [9].

The semantic serving layer exposes gold-layer data to downstream consumers through interfaces optimized for each consumption pattern. SQL interfaces serve BI tools with query plans optimized for analytical access patterns on gold tables. ML feature stores consume gold datasets directly, ensuring training data and serving features share the same governed source without a reconciliation step. API consumers access materialized views and aggregate tables. Domain-oriented data products, curated datasets published by specific business domains under defined service-level agreements, enable multi-consumer architectures without requiring centralized curation of every downstream access pattern [8].

The governance layer enforces access control, lineage capture, data quality monitoring, retention policy management, and schema validation across all medallion zones. Policy engines apply fine-grained access controls at the table and column level. Lineage is captured at each zone transition, linking raw source records to curated outputs and downstream dashboards. Quality rules execute continuously, flagging records that fail validation before they reach the next zone. Schema evolution policies define which changes are backward-compatible and which require coordinated deployment across dependent consumers, preventing silent breaking changes from propagating downstream [7].

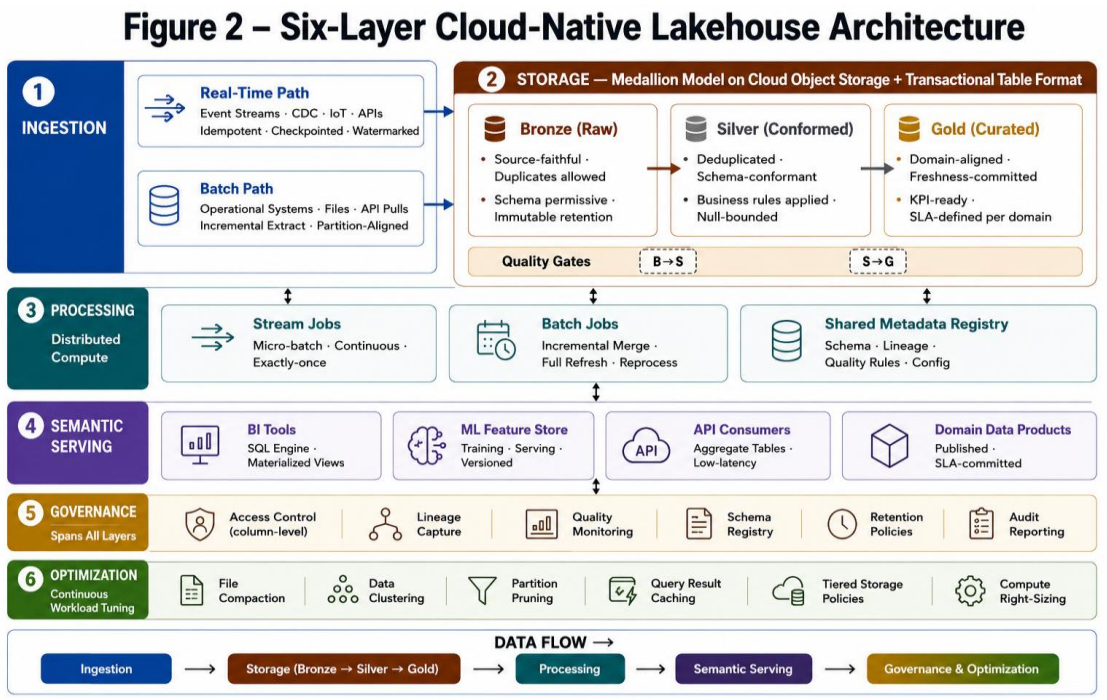

Figure 2. Six-layer cloud-native lakehouse architecture with component interactions and data flow across all layers.

The governance layer enforces access control, lineage capture, data quality monitoring, retention policy management, and schema validation across all medallion zones. Its scope extends across all six architectural layers, making it the connective tissue that determines whether the platform operates as a trustworthy enterprise system or merely as a collection of technical components.

Data catalog integration. A centralized data catalog registers every table, view, materialized dataset, and data product created within the platform. Each asset entry carries business metadata, including a plain-language description, assigned data domain, responsible steward, and applicable data classification tags, alongside technical metadata such as schema, partition structure, storage location, and record counts. The catalog is populated automatically at zone promotion boundaries: when a bronze table is promoted to silver, or a silver dataset to gold, catalog entries are created or updated without requiring manual registration steps. Business users discover available

data assets through the catalog interface rather than through direct storage access, separating the discovery experience from the infrastructure layer [7][12].

Lineage at table, column, and job level. Lineage is captured at three levels of granularity. Table-level lineage records which upstream source tables contributed to each downstream table at each medallion zone transition. Column-level lineage traces the derivation of individual output columns back to their source attributes, including any transformation expressions applied during promotion. Job-level lineage links each executed pipeline run to the input datasets it read, the output datasets it wrote, the transformation logic it applied, and the execution timestamp. Together, these three levels allow data stewards to answer three practical questions: where did this data come from, which downstream consumers are affected if an upstream source changes, and which pipeline run produced a specific version of a dataset. Lineage records are stored in a queryable graph structure and surfaced through the catalog for both interactive investigation and automated impact analysis [7].

PII classification. The governance layer integrates automated scanning that identifies columns containing personally identifiable information at catalog registration time. Classification assigns each identified column a sensitivity category, such as direct identifier, quasi-identifier, or sensitive attribute, based on column name patterns, value distribution sampling, and configurable detection rules. Classification results are stored as catalog metadata and propagate downstream: when a silver or gold table is derived from a classified source column, the classification tag is inherited unless the transformation demonstrably removes identifying characteristics. Classified columns are subject to additional access controls, masking policies, and audit logging requirements applied automatically by the policy engine without requiring pipeline developers to implement protection logic in transformation code [12].

Row-level and column-level security. The policy engine applies two categories of data-level access control independently of the query interface used to access a table. Column-level security restricts visibility of specific columns based on the requesting user's role or group membership; classified PII columns default to masked output for roles without explicit visibility grants, returning a redacted value rather than the underlying data. Row-level security applies filter predicates at query execution time, restricting which rows a given user can retrieve based on attributes such as organizational unit, geography, or data classification. Both control types are defined centrally in the policy engine and evaluated at the serving layer, ensuring that the same restrictions apply whether a user queries through a BI tool, a notebook environment, an API endpoint, or a direct SQL interface [7][12].

Data retention policies. Retention rules are defined per table and per medallion zone and enforced by the governance layer's policy engine rather than by individual pipeline teams. Bronze tables are subject to source-faithful immutable retention for a compliance-defined period, after which records are either purged or moved to archive storage tiers. Silver and gold tables carry retention windows aligned to their freshness commitments and organizational data minimization requirements. Transactional table format time-travel history is bounded by a separately configured retention window to prevent unbounded accumulation of historical snapshot data. Right-to-erasure requests, required under data privacy regulations, are handled through the retention system: the governance layer tracks which physical records correspond to a subject identifier and executes targeted row-level deletes through the transactional table format's ACID delete path, propagating deletions through silver and gold layers in a controlled sequence [4][12].

Data quality rule ownership. Quality rules applied at bronze-to-silver and silver-to-gold zone transitions are registered in the governance layer with an assigned owner: the business domain team or data steward responsible for defining the expectation and responding to violations. Rule ownership is explicit metadata in the catalog, not an informal convention. When a quality monitor detects that a rule is failing at a sustained rate, automated alerting routes to the registered owner rather than to the platform engineering team. Rule owners are also responsible for reviewing and approving changes to rule logic, preventing silent modification of quality expectations across consumer-facing datasets. This ownership model distributes governance accountability to the domain teams with the relevant business knowledge rather than centralizing it in a platform team with limited business context [3][12].

Audit reporting. The governance layer maintains an immutable audit log covering access events, policy changes, schema modifications, quality rule executions and outcomes, zone promotions, and data subject request

completions. Audit records include the actor identity, action type, affected asset, timestamp, and outcome status. Audit data is queryable through a dedicated reporting interface separate from the analytical serving layer, preventing analytical workloads from contending with compliance reporting queries. Pre-built audit reports cover the most common compliance scenarios: data access by user and asset over a defined period, schema changes and their approval status, quality rule violation history by table and rule owner, and evidence of data subject request completion. For regulated industries, these reports provide the structured evidentiary record required by audit assessors without requiring manual reconstruction from system logs across multiple platforms [7].

Access approval workflow. Access to silver and gold tables requires explicit approval rather than self-service provisioning. Data consumers submit access requests through the catalog interface, specifying the requested asset, the intended use, and the required access duration. Requests are routed to the registered data steward for the relevant domain, who reviews and approves or declines within a defined service-level window. Approved grants are time-bounded and subject to periodic recertification; access that is not recertified within the renewal window is automatically revoked. Emergency access requests bypass the standard approval queue through an expedited path that requires post-hoc justification and generates an elevated-priority audit record. The workflow is integrated with the policy engine so that approved grants are enforced immediately without requiring manual configuration steps, and revocations take effect at the next query evaluation without pipeline restarts [12].

Table 1. Architecture layer summary showing the primary function, key components, input, and output for each layer.

Layer	Primary Function	Key Components	Input	Output
Ingestion	Land raw data with source fidelity and immutability	Streaming platform, CDC connectors, batch loaders	Streams, files, APIs, CDC feeds	Bronze zone raw tables
Storage	Organize data with ACID guarantees and schema governance	Transactional table format, object storage, medallion zones	Raw and processed records	Bronze, silver, gold table layers
Processing	Transform and enrich data across medallion transitions	Distributed compute, stream and batch engines, shared metadata	Bronze and silver tables	Silver and gold tables
Semantic Serving	Expose curated data to diverse consumer patterns	SQL interface, feature store, materialized views, data products	Gold tables	BI reports, ML features, API responses
Governance	Enforce access, quality, lineage, and retention uniformly across all layers	Data catalog, policy engine, lineage tracker, quality monitors, schema registry, PII classifier, audit log, access workflow	All layers	Access logs, quality reports, lineage graph, PII classifications, audit reports, approved access grants
Optimization	Tune storage layout and compute to observed workloads	Compaction scheduler, cache manager, partition advisor	Workload telemetry	Optimized storage layout, right-sized compute

4. Results: Implementation Blueprint and Evaluation

A practical implementation begins with the storage foundation. Cloud object storage, such as Amazon S3, Azure Data Lake Storage, or Google Cloud Storage, provides the base layer. A transactional table format is deployed over this storage, managing transaction logs, snapshot isolation, schema evolution, and version history. The choice of table format determines the feature set available for time-travel queries, row-level operations, and concurrent write handling, all of which affect the complexity of bronze-to-silver and silver-to-gold promotion pipelines. Production deployments have demonstrated petabyte-scale row-level operations under this model without architectural modification [4].

Streaming ingestion is configured with idempotent write semantics to prevent duplication on retry. Checkpointing ensures that stream processing state survives job restarts without data loss or reprocessing beyond the committed offset. Watermarking defines the acceptable event-time lateness window, allowing the streaming engine to emit results before all data for a time has arrived, while bounding the staleness of those results for downstream consumers. Batch ingestion uses incremental patterns, CDC, watermark-based extraction, or partition-aligned full refreshes to minimize the volume of data reprocessed at each execution cycle [5][9].

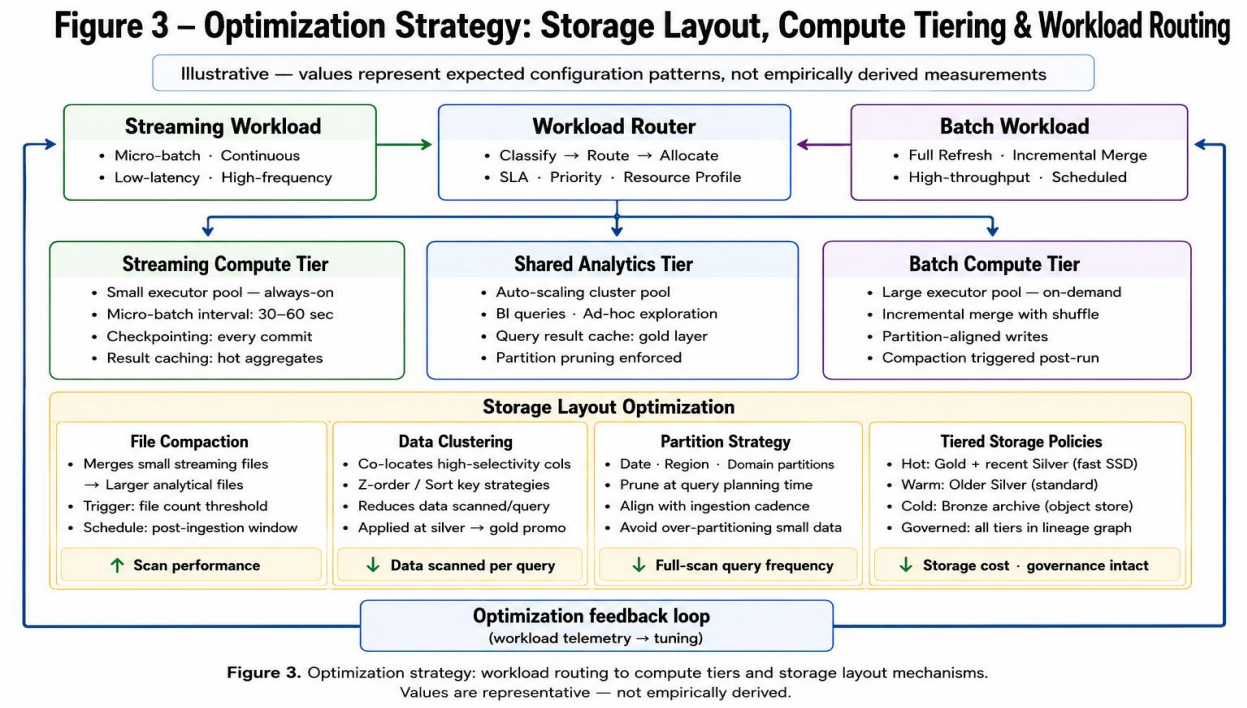

Figure 3. Optimization strategy: workload routing to compute tiers and storage layout mechanisms.
Values are representative — not empirically derived.

Figure 3. Optimization strategy illustrating storage layout, compute tiering, and workload routing for streaming and batch pipelines. Values shown are representative of expected configuration patterns and are not empirically derived.

Performance optimization at the storage layer centers on three mechanisms. File compaction merges the small files produced by streaming micro-batch writes into larger files that analytical query engines read more efficiently. Without compaction, streaming workloads accumulate millions of small files, which degrade scan performance over weeks of operation. Data clustering co-locates records with similar values in high-selectivity filter columns, reducing data scanned per query for common access patterns. Incremental merge operations update silver and gold tables by processing only changed records from upstream zones, rather than recomputing entire partitions on each promotion cycle [11].

The architecture is assessed across six evaluation dimensions. Ingestion throughput measures sustained event processing rate for the streaming path and record processing rate for batch pipelines. End-to-end latency measures elapsed time from event generation to availability in gold-layer tables. Query performance captures 50th and 95th

percentile latency for representative analytical workloads. Storage efficiency is the ratio of logical data volume to physical storage consumed after compression and deduplication. Cost per workload compares total infrastructure spending across equivalent analytical outputs on the unified lakehouse versus the fragmented alternative. Schema evolution ease measures the fraction of schema changes that can be applied without requiring downstream pipeline modification [12].

Table 2. Evaluation metrics with measurement method, fragmented architecture baseline, and unified lakehouse expected outcome.

Metric	Measurement Method	Fragmented Architecture Baseline	Unified Lakehouse Expected Outcome
Ingestion throughput	Events/sec (streaming); records/sec (batch) at sustained load	Separate streaming and batch capacity with independent scaling limits	Unified capacity pool with dynamic allocation between streaming and batch
End-to-end latency	Source event timestamp to gold table availability	High batch latency; streaming data not joined with historical at query time	Low latency for streaming; batch history co-located for unified queries
Query performance (P95)	P95 latency for representative analytical workload mix	Varies across warehouse, lake, and streaming query interfaces	Consistent from unified serving layer with shared storage optimization
Storage efficiency	Logical-to-physical ratio after compression and deduplication	Duplication across lake, warehouse, and streaming sinks	Reduced duplication; single copy with tiered retention policies
Cost per workload	Total infrastructure spend per equivalent analytical output	Higher from duplicate storage and separate compute provisioning	Lower from shared storage foundation and unified compute tier
Schema evolution ease	Fraction of schema changes requiring no downstream modification	High modification rate: changes propagate across multiple independent systems	Higher compatibility through transactional format schema evolution support

Expected outcomes from a comparative evaluation against a fragmented legacy architecture include measurable reductions in data duplication, lower operational overhead from unified governance enforcement, faster onboarding of new data sources into the medallion model, and more consistent data quality across analytical consumers. By combining batch history and streaming context, we remove the reconciliation step that slows down ML feature generation in fragmented architectures. This allows for consistent feature pipelines for model training and real-time serving, without version skew between historical and streaming data [12][3].

Table 3. Medallion layer quality contracts defining expectations, service-level agreements, access patterns, and consumer types for each zone.

Zone	Data Quality Expectations	SLA	Primary Access Patterns	Consumer Types
Bronze	Source-faithful; no quality enforcement; duplicates permitted; schema permissive	Ingest within defined latency window; retain for compliance period	Append-only writes; full-scan or partitioned reads for reprocessing	Reprocessing pipelines, compliance archive, platform engineering

Silver	Deduplicated; schema-conformant; business rules validated; null rates bounded	Available within defined SLA after bronze landing	Incremental reads for gold promotion; investigative ad-hoc queries	Gold promotion jobs, data quality teams, investigative analytics
Gold	Business-ready; domain-aligned; fully governed; freshness-committed	Defined freshness SLA per domain product; breach triggers alerting	Optimized analytical reads; feature store access; API serving	BI tools, ML models, data products, API consumers, executives

5. Discussion

The strategic value of the lakehouse extends beyond the infrastructure metrics. Organizations that consolidate onto a unified data platform create conditions for governance consistency that fragmented architectures cannot achieve, such as a single access control model, a single lineage graph, and a single set of quality rules applied uniformly across all consumers rather than differently across warehouse, lake, and streaming systems. This consistency reduces compliance risk in regulated industries and reduces the audit evidence burden that currently requires reconstruction from multiple systems with different logging formats [7].

The AI and ML implications are equally significant. Because batch history and streaming context exist in the same governed platform, feature generation for model training and model serving can draw from the same sources without reconciliation pipelines that introduce latency, version skew, or quality inconsistencies between training and serving distributions. The medallion gold layer provides a natural feature store boundary, curated, validated, and freshness-committed datasets that ML pipelines can consume with confidence in their quality and consistency [3].

Three limitations require direct acknowledgment. First, the lakehouse reorganizes operational complexity without eliminating it. Compaction scheduling, schema migration management, partition strategy decisions, and medallion promotion logic all require ongoing engineering attention. Organizations that underinvest in platform operations after initial deployment will see query performance and data quality degrade as the platform scales. Second, migration from a fragmented architecture carries non-trivial cost and risk. Parallel operation during migration, running both legacy systems and the new platform simultaneously until consumer cutover is complete, extends the transition period and temporarily increases infrastructure costs. Third, not all workloads justify the investment. Simple batch-only data estates with stable schemas, low query volumes, and limited consumer diversity may not recover the migration cost within a reasonable planning horizon [14][2].

Table 4: Limitation analysis showing when each limitation applies, its severity, and the recommended mitigation approach.

Limitation	**When It Applies**	**Severity**	**Mitigation Approach**
Ongoing operational complexity	All deployments, compaction, schema management, partition strategy require active attention	Medium, manageable with dedicated platform engineering	Dedicated platform engineering function; automation of compaction and optimization routines
Migration cost and transition risk	Organizations migrating from fragmented legacy architectures	High during transition, parallel operation increases cost	Phased migration by domain; consumer cutover milestones with defined rollback criteria
Uneven workload benefits	Simple batch-only estates with stable schemas and low consumer diversity	Low,lakehouse overhead may exceed benefit for small estates	Assess workload complexity and growth trajectory before committing to migration

Governance maturity prerequisite	Organizations with immature cataloging and metadata management practices	Medium,lineage and quality enforcement require metadata infrastructure	Establish metadata standards and governance tooling before lakehouse deployment begins

6. Conclusion

A cloud-native lakehouse provides a compelling architecture for enterprises that require both real-time responsiveness and large-scale analytical depth and that need governance, lineage, and data quality applied consistently across both. By unifying streaming and batch processing under a consistent storage, governance, and serving model, organizations reduce the architectural fragmentation that drives up operational cost, governance inconsistency, and delivery latency in traditional multi-system data estates.

The six-layer architecture proposed in this article addresses the full scope of enterprise data intelligence requirements. The medallion model enforces progressive data quality from source ingestion through conformation to curated domain outputs, creating natural boundaries between raw operational data, validated analytical data, and business-ready data products. The governance and optimization layers ensure that this structure remains consistent and performant as the platform scales without requiring manual intervention for every workload pattern that emerges as adoption grows.

Future directions include autonomous workload optimization, where storage layout and compute allocation decisions are made by learning systems rather than manually configured rules; semantic query acceleration, where natural language query interfaces are grounded in the curated gold-layer semantic model; and integration with AI-driven pipeline automation frameworks that generate and govern the transformation logic operating within the lakehouse platform.

REFERENCES

1. M. Armbrust et al., "Delta lake: High-performance ACID table storage over cloud object stores," Proceedings of the VLDB Endowment, vol. 13, no. 12, pp. 3411-3424, 2020. https://doi.org/10.14778/3415478.3415560
2. A. Nambiar and D. Mundra, "An overview of data warehouse and data lake in modern enterprise data management," Big Data and Cognitive Computing, vol. 6, no. 4, p. 132, 2022. https://www.mdpi.com/2504-2289/6/4/132
3. P. Wieder and H. Nolte, "Toward data lakes as central building blocks for data management and analysis," Frontiers in Big Data, 2022. https://www.frontiersin.org/journals/big-data/articles/10.3389/fdata.2022.945720/full
4. A. Okolnychyi et al., "Petabyte-scale row-level operations in data lakehouses," Proceedings of the VLDB Endowment, vol. 17, no. 12, pp. 4159-4172, 2024. https://doi.org/10.14778/3685800.3685834
5. M. Armbrust et al., "Structured streaming: A declarative API for real-time applications in Apache Spark," in Proc. ACM SIGMOD, 2018, pp. 601-613. https://doi.org/10.1145/3183713.3190664
6. B. Dageville et al., "The Snowflake elastic data warehouse," in Proc. ACM SIGMOD, 2016, pp. 215-226. https://doi.org/10.1145/2882903.2903741
7. H. Dong et al., "Cloud-native databases: A survey," IEEE Transactions on Knowledge and Data Engineering, vol. 36, no. 11, pp. 5572-5591, 2024. https://doi.org/10.1109/TKDE.2024.3397508
8. R. Sethi et al., "Presto: SQL on everything," in Proc. IEEE 35th ICDE, 2019, pp. 1802-1813. https://doi.org/10.1109/icde.2019.00196
9. P. Carbone et al., "State management in Apache Flink(R): Consistent stateful distributed stream processing," Proceedings of the VLDB Endowment, vol. 10, no. 12, pp. 1718-1729, 2017. https://doi.org/10.14778/3137765.3137777
10. Y. Mao et al., "StreamOps: Cloud-native runtime management for streaming services in ByteDance," Proceedings of the VLDB Endowment, vol. 16, no. 12, pp. 3501-3513, 2023. https://doi.org/10.14778/3611540.3611543
11. A. Lamb et al., "The Vertica analytic database: C-Store 7 years later," Proceedings of the VLDB Endowment, vol. 5, no. 12, pp. 1790-1801, 2012. https://doi.org/10.14778/2367502.2367518
12. A. A. Harby and F. Zulkernine, "Data lakehouse: A survey and experimental study," Information Systems, vol. 127, p. 102460, 2025. https://doi.org/10.1016/j.is.2024.102460
13. J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1, pp. 107-113, 2008. https://doi.org/10.1145/1327452.1327492
14. N. Li et al., "Towards near real-time data warehousing," in Proc. IEEE 24th AINA, 2010, pp. 1165-1172. https://doi.org/10.1109/aina.2010.54

15. A. Abouzeid et al., "HadoopDB: An architectural hybrid of MapReduce and DBMS technologies for analytical workloads," Proceedings of the VLDB Endowment, vol. 2, no. 1, pp. 922-933, 2009. https://doi.org/10.14778/1687627.1687731