SvedbergOpen
DISSEMINATION OF KNOWLEDGE
International Journal of Artificial Intelligence and Machine Learning
Publisher's Home Page: https://www.svedbergopen.com/
Research Paper Open Access

Intelligent Lakehouse Architectures For Real-Time Enterprise Decision Systems

Ananda Kumar Dey

Quadrant Technologies LLC, USA ORCID: 0009-0003-2322-3688

Abstract
The lakehouse has emerged as the leading architectural pattern for enterprise data platforms over the past five years. It combines the elastic scale and openness of the data lake with the transactional reliability and SQL performance of the data warehouse. This article examines intelligent lakehouse architectures across eight themes: the evolution from data warehouses through data lakes to lakehouses; unified analytics organized around medallion data zones and open table formats; real-time streaming intelligence built on durable event logs; delta processing and incremental computation that convert expensive full rebuilds into cheap incremental merges; AI-native data infrastructure including feature stores, vector stores, and model registries; performance optimization through partitioning, clustering, and predicate pushdown; cost-efficient analytics through compute-storage decoupling; and industry-specific applications across six sectors. Two mathematical interludes derive quantitative impact claims from canonical research. Results show that incremental delta processing achieves speedups of 100 to 10,000 times over full rebuilds for typical change rates and that the combination of column projection, partition pruning, and clustering can reduce query scan volume by four orders of magnitude. Industry applications in financial services, retail, healthcare, insurance, manufacturing, and the public sector demonstrate that the lakehouse is not merely a technical pattern but a foundation for the next generation of enterprise decision systems.

Keywords: Lakehouse, Delta Lake, Apache Iceberg, Apache Hudi, Apache Parquet, Apache Kafka, Apache Flink, Medallion Architecture, Real-Time Analytics, Streaming Intelligence, Change-Data-Capture, Delta Processing, AI-Native Data Platform, Feature Store, Vector Store, Enterprise Data Platform Architecture, Cloud Modernization.

1. Introduction

Enterprise decision systems have always depended on data platforms, but the relationship between decisions and platforms has changed fundamentally in the past decade. Where decisions were once made by analysts working from periodic reports against a curated warehouse, modern decisions are made in real time by operational systems, machine-learning models, and increasingly by generative AI applications. These applications far exceed the latency, freshness, and breadth requirements that classical data warehouses were designed to meet. According to Armbrust et al. (2021), when lakehouse architecture was introduced at the 11th Conference on Innovative Data Systems Research (CIDR 2021), the goal was to exploit the elastic storage and open platform that usually show up in data lakes, while also adding the transactional integrity, performance, and governance features that you typically see in data warehouses.

That general approach has been broadly taken up (Armbrust et al., 2020) , and it is now applied in open-source tools such as Delta Lake, Apache Iceberg, and Apache Hudi, as well as in enterprise environments like the Databricks Lakehouse Platform, Microsoft Fabric and Azure Synapse (Chaudhari & Charate, 2025). The architectural premise is straightforward but consequential. Cloud object stores provide effectively unlimited capacity at low cost with high durability, but they were originally designed as key-value blob systems without support for the ACID transactions, atomic schema changes, or efficient updates that data warehouse workloads require. The lakehouse closes this gap by adding an open table format over Parquet columnar files in the object store, with a transaction log that provides ACID semantics, schema evolution, time travel, and efficient incremental processing.

Armbrust et al. (2020) described the Delta Lake protocol and reported that thousands of organizations deploy it to process exabytes of data per day. This confirms that the pattern operates not merely in theory but at the scale of the largest enterprise data estates. Apache Kafka, originally developed at LinkedIn, has become the standard durable event log in the modern data platform (Apache Kafka Project, n.d.). Apache Flink, whose unified stream-and-batch processing model was described by Carbone et al. (2015) in the IEEE Data Engineering Bulletin, provides the stream processor that transforms event flows into curated lakehouse tables.

This article examines the lakehouse as the foundation of intelligent enterprise decision systems across eight thematic sections. Section 2 traces the architectural evolution from warehouses through data lakes to lakehouses. Section 3 examines unified analytics architectures organized around medallion data zones. Section 4 addresses real-time streaming intelligence. Section 5 analyzes delta processing and incremental computation, with a mathematical interlude. Section 6 examines AI-native infrastructure. Section 7 addresses performance optimization, with a second mathematical interlude. Section 8 considers cost-efficient large-scale analytics. Section 9 surveys industry-specific applications with a capability checklist. The article concludes that the lakehouse is a foundation for the next generation of enterprise decision systems.

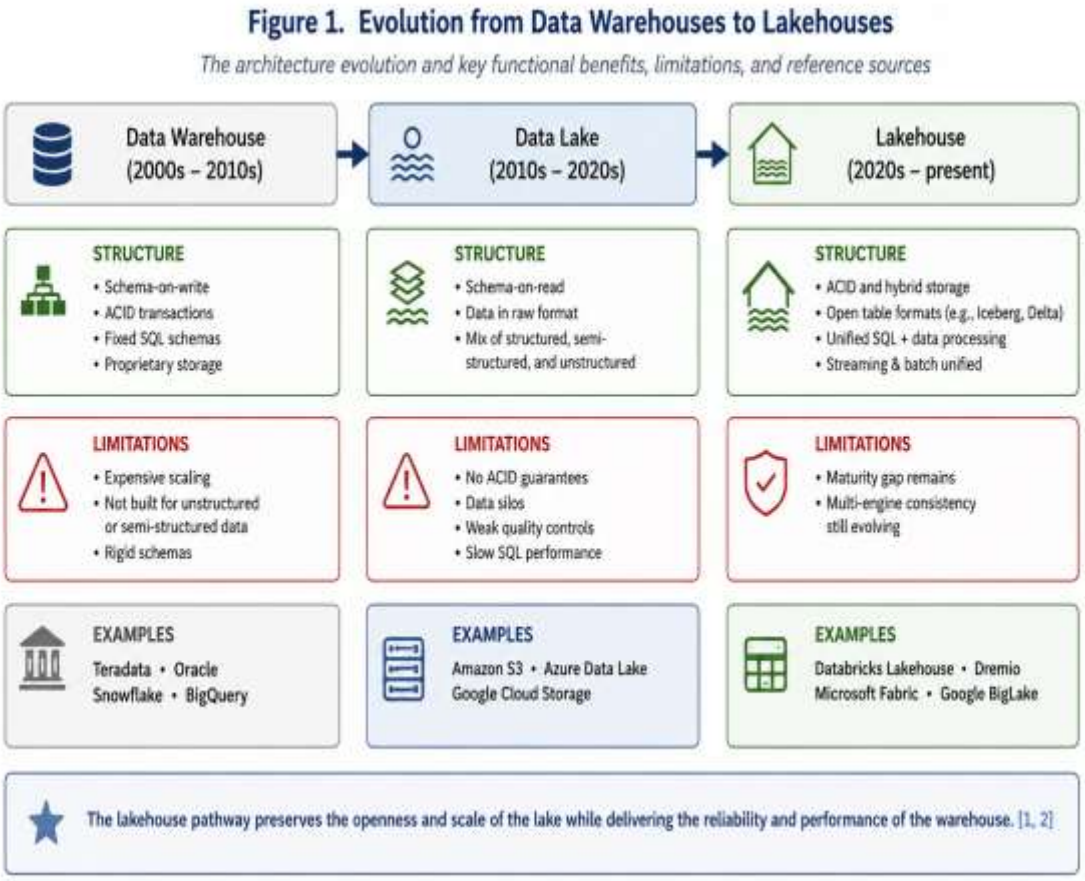
2. Evolution from Data Warehouses to Lakehouses

The architectural history of enterprise data platforms divides into three generations, each defined by the storage technology available at the time and the analytical workloads it was expected to support. This era in data warehousing began in the 1980s with Teradata, Oracle, and IBM DB2, and it has since evolved to include cloud-native data warehouses like Snowflake, BigQuery, and Amazon Redshift. It operates under the assumption that good-quality data must be loaded into a well-governed schema to run queries using SQL. Warehouses delivered fast analytics, ACID transactions, and strong consistency, but their proprietary storage formats produced vendor lock-in and their cost scaled poorly with data volume.

The data lake generation, which began in the 2010s with Hadoop and HDFS and broadened to cloud object stores including Amazon S3 and Azure Data Lake Storage, inverted these trade-offs. Lakes stored data in open formats at low cost, accommodated unstructured and semi-structured data natively, and made the underlying data available to a broad ecosystem of compute engines. The cost was reliability: data lakes lacked the ACID transactions, schema enforcement, and metadata management that warehouses provided, and many enterprise lake deployments degraded into so-called data swamps, large pools of inconsistent and untrustworthy data.
Formalized by Armbrust et al. (2021), the lakehouse generation proposed that the trade-off between the warehouse and the lake is architectural, not fundamental. An appropriately designed table format could deliver warehouse-grade reliability and performance over data lake storage. The Delta Lake protocol described by Armbrust et al. (2020) provides a concrete example: a transaction log written alongside the data files in the object store and a set of protocols to achieve serializability. Delta Lake originated at Databricks, the company that also introduced the lakehouse concept (Armbrust et al., 2021), and it is the table format underlying the Databricks Lakehouse Platform. The Apache Iceberg format, which was created at Netflix, kinda stresses engine interoperability (Mishra, 2022). The Apache Hudi format, developed at Uber Technologies Inc., gives incremental processing and change data capture capability, so it can deal with write-intensive scenarios. Even if their architectures are not the same, these three table formats are actually fairly similar, and there are ongoing efforts to make them play together and be interoperable (Kasireddy, 2023).

Figure 1 shows the evolution from data warehouses to lakehouses. Each generation is characterized by its storage technology, strengths, limitations, and example systems. The lakehouse pattern, formalized at CIDR 2021 (Armbrust et al., 2021) and instantiated in Delta Lake (Armbrust et al., 2020), Apache Iceberg, and Apache Hudi, preserves the openness and scale of the lake while delivering the reliability and performance of the warehouse.

Figure 1: Evolution from Data Warehouses to Lakehouses (Armbrust et al., 2021)



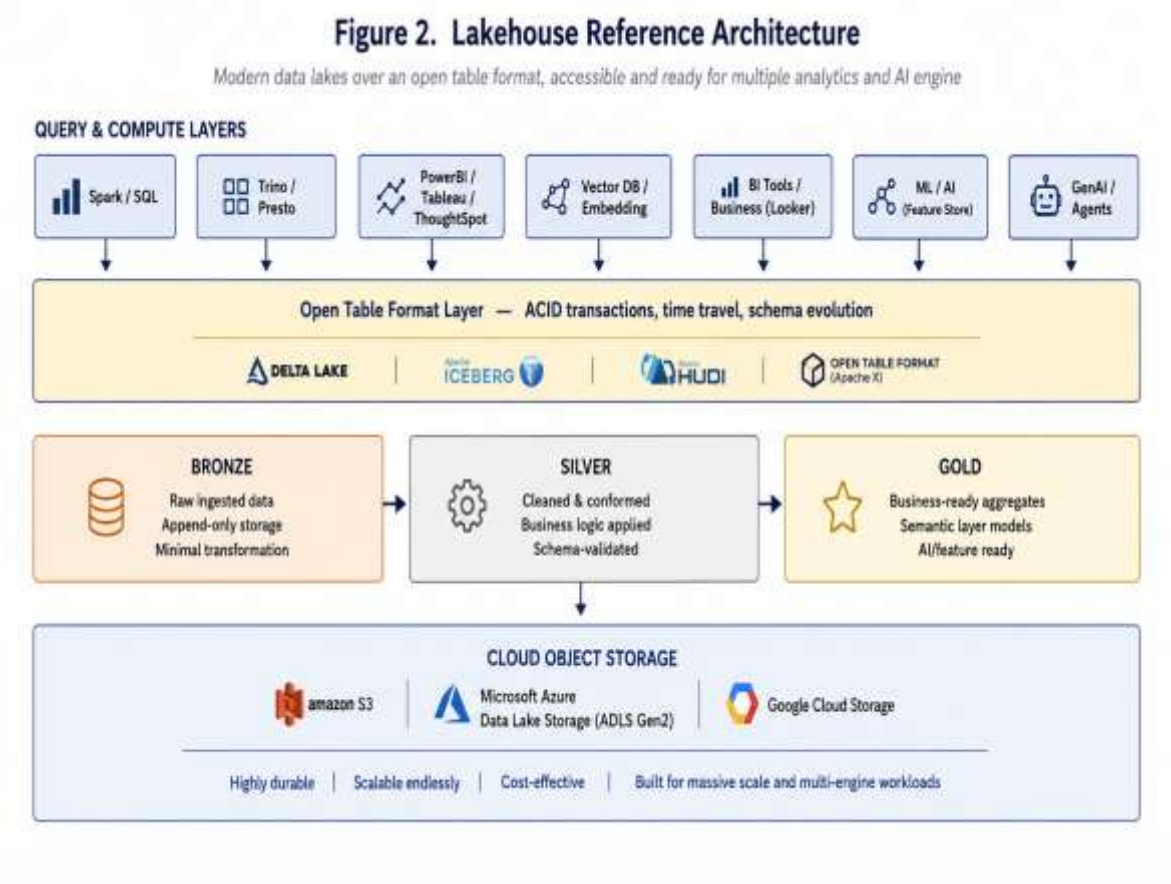
3. Unified Analytics Architectures

A defining property of the lakehouse is that a single physical copy of the data simultaneously serves multiple consumption patterns. In earlier architectures, physical copies often forced SQL analytics, machine learning, streaming, and ad-hoc data science workloads to operate. In the lakehouse architecture, the open table format and cloud object store substrate can be queried in place by all major engines in the ecosystem: Apache Spark for batch and streaming, Trino and Presto for interactive SQL, Microsoft Fabric for enterprise workloads (Chaudhari & Charate, 2025), Apache Flink for streaming SQL (Carbone et al., 2015), and all ML and Python runtimes via DataFrame APIs.

Internally, the medallion architecture organizes a lakehouse into bronze, silver, and gold data zones. The append-only landing zone is the bronze zone where data is ingested from operational systems, providing full history and typically requiring minimal transformation. The silver layer applies deduplication, schema enforcement, and conformance to create cleansed, joined tables that represent the canonical truth of the enterprise's operational data. The gold layer applies business-ready aggregations, dimensional modeling, and AI feature engineering to produce the tables that analytics dashboards, ML models, and AI applications consume directly.

Open table formats provide the connective tissue that holds the unified architecture together. Delta Lake, Iceberg, and Hudi each implement ACID transactions through transaction logs, schema evolution, time travel, and efficient upserts and merges. These functionalities can be done at the table format layer , not so much in any specific engine, so multiple engines can still read the same underlying datasets (Pierce, 2025). Figure 2 depicts lakehouse reference architecture. The cloud object storage substrate is mediated by an open table format layer (Delta Lake, Apache Iceberg, or Apache Hudi) over Apache Parquet columnar storage. The medallion data zones (bronze, silver, gold) organize successive cleansing stages. Multiple analytics and AI engines consume the same data uniformly.

Figure 2: Lakehouse Reference Architecture



4. Real-Time Streaming Intelligence

Modern enterprise decision systems are increasingly intolerant of stale data. Customer-facing personalization, real-time fraud detection, operational dashboards, supply-chain coordination, and AI copilots all depend on data that reflects the current state of the business within seconds. The lakehouse must therefore support continuous ingestion and processing rather than only periodic batch loads. The architecture that has emerged for this requirement consists of three coordinated elements: a durable event log that captures every operational event with strong ordering and replay guarantees; a stream processor that transforms event flows into the table forms that downstream consumers read; and an open table format that supports efficient incremental merges.

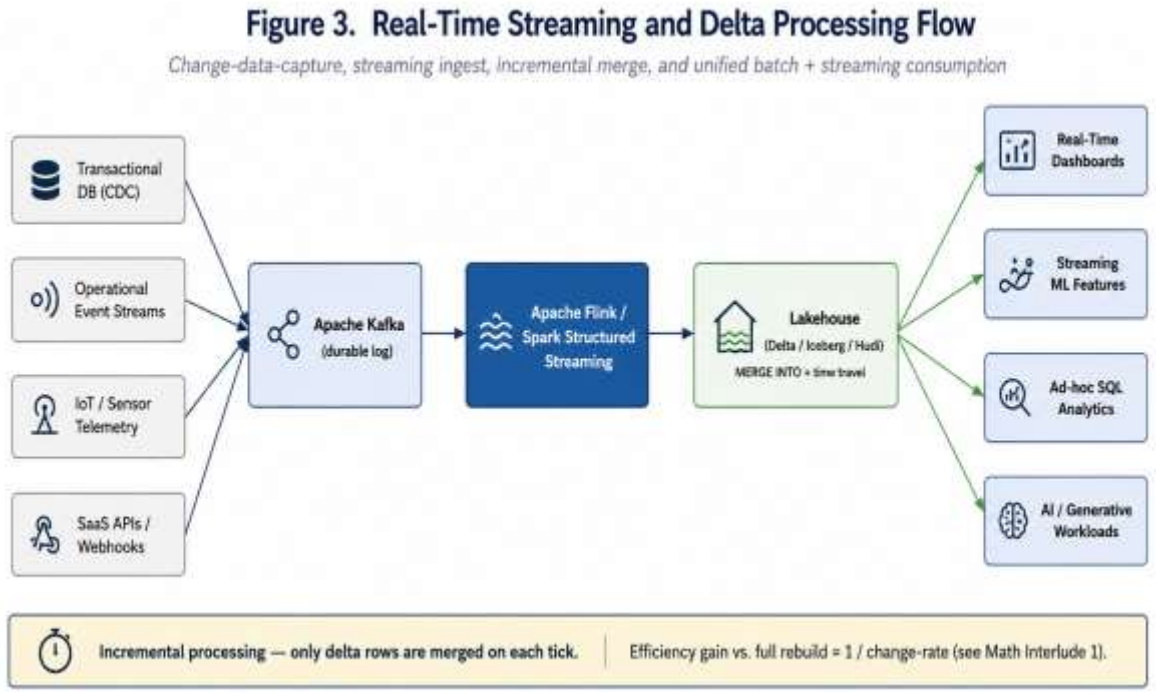
Apache Kafka has become the standard durable event log in this architecture (Apache Kafka Project, n.d.). Kafka's distributed, partitioned, replicated log model provides the scalability and replay semantics that real-time pipelines depend on. Apache Flink and Apache Spark Structured Streaming are the two dominant stream processors operating over Kafka in enterprise lakehouses. Flink's unified stream-batch model, described by Carbone et al. (2015), treats batch as a special case of streaming over a bounded dataset, allowing the same processing logic to handle historical reprocessing and live streaming with minimal modification. Spark Structured Streaming, described by Armbrust et al. (2018), offers tight coupling between streaming ingestion and lakehouse persistence and is the natural choice for organizations whose lakehouse is built on Delta Lake.

The combination of Kafka, Flink, or Structured Streaming, and a lakehouse table format produces an architecture in which the same data simultaneously serves multiple consumption patterns. Real-time dashboards subscribe to streaming aggregates derived directly from the event log. Streaming ML feature pipelines compute time-windowed features and write them to the feature store. Ad hoc SQL analytics queries the lakehouse tables that the streaming pipeline materializes. AI and generative AI workloads retrieve grounding context from the same tables. As an example of a lambda architecture for energy-efficient IoT homes, Serepas et al. (2024) show in a production environment how the overhead of synchronizing joined data sources is eliminated by unified processing.

Figure 3 illustrates the real-time streaming and delta processing flow. Operational sources flow through Apache Kafka (Apache Kafka Project, n.d.) into a stream processor (Apache Flink, Carbone et al., 2015, or Spark Structured

Streaming, Armbrust et al., 2018) that performs MERGE INTO operations against the lakehouse table format. Real-time dashboards, streaming ML features, ad-hoc analytics, and AI workloads consume the same data.

Figure 3: Real-Time Streaming and Delta Processing Flow (Carbone et al., 2015 & Armbrust et al., 2018)



5. Delta Processing and Incremental Computation

The defining innovation of lakehouse table formats is efficient support for incremental processing. Classical data warehouse and data lake operations were largely full-rebuild: a daily batch job read the entirety of an upstream source, applied transformations, and wrote the results. This pattern was operationally simple but computationally wasteful, because the fraction of records that actually change between runs is typically small, often less than one percent in transactional systems.

The delta processing pattern, supported natively by Delta Lake's MERGE INTO operation (Armbrust et al., 2020), Iceberg's row-level operations, and Hudi's primary-key-based upserts, inverts this trade-off. Change-data-capture identifies the rows that have been inserted, updated, or deleted since the last run, and the processing pipeline applies only those changes to the destination table. Debezium, Microsoft Fabric's mirroring, AWS Database Migration Service, and Hudi's incremental queries connect to the transaction logs of operational databases and emit the inserts, updates, and deletes as a continuous stream of events. The stream processor then applies the changes through the table format's transaction log. This is the practical realization of the streaming-batch unification described by Carbone et al. (2015): the same logical pipeline handles both the initial bulk load and the ongoing stream of changes, with consistent semantics across both modes.

Mathematical Interlude 1: Efficiency of Incremental Delta Processing

Let:

D = total rows in the target table

c = fraction of rows changed per processing tick

W_{full} = work by a full-rebuild operation

W_{delta} = work by an incremental delta merge

Order-of-magnitude work models:

$W_{full} = O(D)$ (scan and rewrite every row)

W_delta = O(c * D) (scan and apply only changed rows)

Speedup of incremental over full:
W_full / W_delta = 1 / c

Concrete examples:
c = 0.01 -> speedup ~ 100x
c = 0.001 -> speedup ~ 1,000x
c = 0.0001 -> speedup ~ 10,000x

The 1/c factor explains why CDC-driven incremental processing has become the default pattern for write-heavy lakehouse workloads. For typical transactional systems where the daily change rate is well under 1 percent, delta processing converts hour-scale full rebuilds into minute-scale incremental merges. A daily batch job now reads and applies only the 100,000 rows that changed, instead of rebuilding a 100-million-row table by reading and rewriting the entire table, resulting in a 1,000x reduction in compute cost. The savings compound across the hundreds or thousands of pipelines a large enterprise operates (Cloudera, 2024).

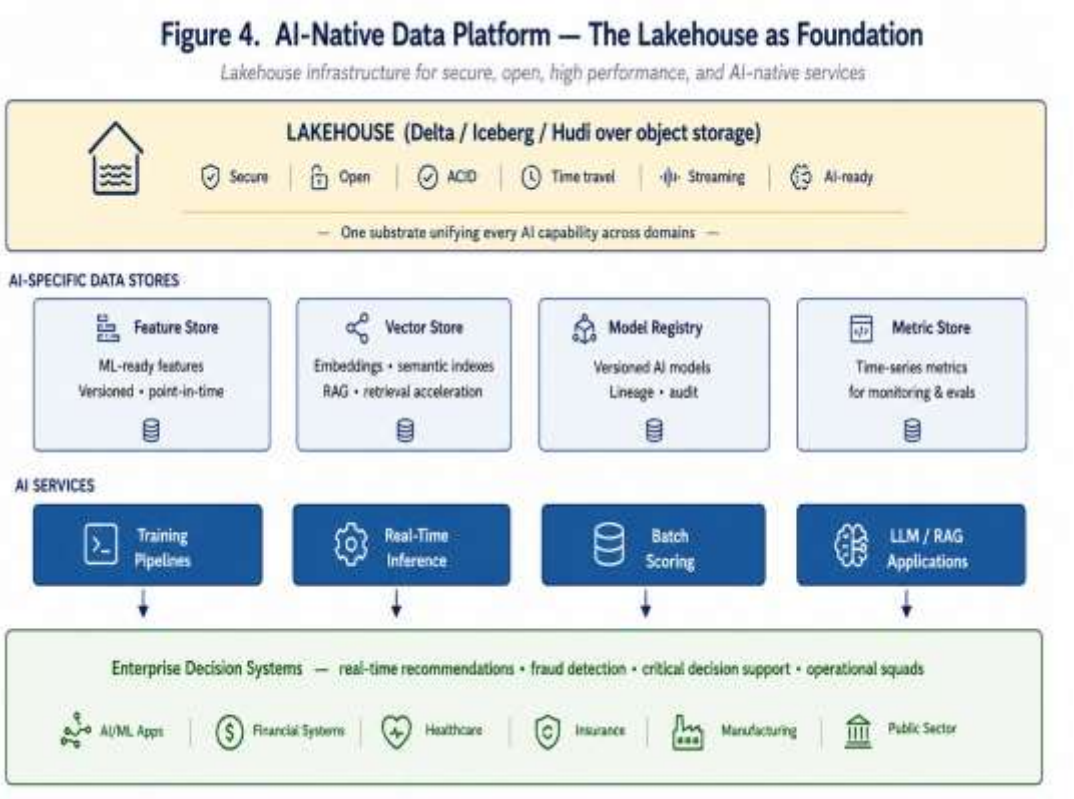
6. AI-Native Data Infrastructure

The lakehouse's emergence as the dominant enterprise data platform pattern coincides with the rise of machine learning and generative AI as central enterprise workloads, and the relationship between the two trends is not coincidental. AI workloads place a distinctive set of demands on the data platform: they consume large volumes of historical data for training, they require fresh data for online inference, they need point-in-time correctness for accurate model evaluation, and they increasingly depend on semantic embeddings and vector retrieval rather than traditional row and column access. Lakehouses are well-suited to support these demands because the same physical storage that holds the analytics tables can also hold training datasets, feature tables, model artifacts, and the metric streams used to monitor models in production.

Three AI-specific data stores typically extend the lakehouse to form an AI-native platform. Feature stores maintain ML-ready features with point-in-time correctness, ensuring that a model trained on features as they appeared at training time receives the same features at inference time without temporal leakage. Vector stores maintain the embedding indexes that retrieval-augmented generation applications depend on, and increasingly these vector indexes are co-located with the textual data they describe. Model registries maintain versioned models with full lineage from training data through evaluation results to deployed inference services, providing the auditability that governance frameworks require. All three stores depend on the lakehouse as the substrate of authoritative data (Jha, 2024).

Microsoft Fabric points out this lakehouse-AI integration through the use of OneLake, a common Delta-format storage layer for the SQL analytics, data engineering, real-time analytics, machine learning and artificial intelligence workloads. This approach enables datasets to be reused between the different Fabric workloads without creating separate copies of the data and ensures consistent usage of governance policies across the whole architecture (Chaudhari & Charate, 2025). The Databricks Lakehouse Platform integrates the same AI-native stores directly on its Delta-format lakehouse (Armbrust et al., 2021), combining a feature store, the MLflow model registry, and vector search with Unity Catalog governance and lineage so that training, inference, and retrieval-augmented generation operate over a single governed copy of the data (Zaharia et al., 2016). A growing body of work around MLOps practices stresses the need for model training and deployment pipelines with the lakehouse storage layer in between to achieve strong, AI-native platforms (Eswararaj et al., 2025). Figure 4 depicts an AI-Native Data Platform, where the lakehouse (Armbrust et al., 2021, 2020) serves as the substrate for four AI-specific data stores (feature store, vector store, model registry, and metric store) and four AI service layers (training pipelines, real-time inference, batch scoring, and LLM/RAG applications). The result is a single coherent platform delivering industry-specific decision systems.

Figure 4: AI-Native Data Platform (Armburst et al., 2020; Armburst et al., 2021)



7. Performance Optimization in Distributed Query Systems

A small number of fundamental techniques shape distributed query performance in lakehouses, and their effects compound multiplicatively. The first is columnar storage. Apache Parquet (Apache Parquet Project, n.d.), the standard columnar format used by every major lakehouse table layer, stores data in column groups with embedded statistics and dictionary encoding. This layout allows query engines to read only the columns referenced by the query and to skip column groups whose statistics indicate they cannot satisfy the query's predicates, a property known as predicate pushdown. The second is partitioning: tables are physically partitioned by columns commonly used in filter predicates so that queries with filters on those columns scan only the relevant partitions. The third is clustering or Z-ordering: within partitions, data is further organized so that rows with similar values in the clustering columns are co-located.

Delta Lake provides Z-ordering as a built-in maintenance operation. Iceberg provides hidden partitioning that decouples the physical partition layout from the column expressions used in queries, allowing the partition strategy to evolve without rewriting the queries. Hudi provides record-level indexing to support efficient point lookups by primary keys. All three engines push predicates down to the Parquet file level, project only the requested columns, and produce query plans that exploit table-level statistics (Kasireddy, 2023). The practical implication for enterprise architects is that the choice of partitioning and clustering strategy at table-design time has enormous downstream consequences for query latency and cost, and the analytical workload patterns of an organization should explicitly inform this choice.

Mathematical Interlude 2: Query Cost under Partition Pruning and Clustering

Let:

D = total data size in bytes

s_part = fraction of partitions accessed (partition selectivity)

s_file = fraction of files within accessed partitions that pass file-level predicate pushdown (depends on clustering)

s_col = fraction of columns read (column projection)

Bytes scanned by the query engine:
Bytes_scanned = D x s_part x s_file x s_col

Concrete example -- date-partitioned fact table:
D = 10 TB
s_part = 1/365 ~ 0.0027 (query touches 1 day of data)
s_file = 0.10 (10% of files pass clustering filter)
s_col = 0.20 (5 of 25 columns referenced)

Bytes_scanned ~ 10 TB x 0.0027 x 0.10 x 0.20
~ 540 MB (~0.005% of total table)

Each factor in the product compounds multiplicatively. Columnar formats, predicate pushdown, partition pruning, and clustering together convert a 10-terabyte scan into a 540-megabyte scan, a reduction of roughly four orders of magnitude that translates directly into proportional reductions in latency and cost on pay-per-scan cloud platforms. Interactive SQL latencies of one to ten seconds against multi-terabyte tables are now routinely achievable in production enterprise lakehouses.

8. Cost-Efficient Large-Scale Analytics

The economic logic of the lakehouse is what has made its rapid enterprise adoption possible. Three structural advantages combine to produce a lower total cost of ownership than the predecessor architectures it replaces. The first is the decoupling of compute and storage. Classical data warehouses tied compute capacity to storage capacity in a single monolithic system, which meant that organizations paid for compute they did not use whenever they paid for storage they did. Cloud object storage decouples these resources: storage is paid per gigabyte per month at low fixed rates, and compute is paid per query, per second, or per cluster hour according to actual usage.

The second advantage is the use of open file formats. Parquet (Apache Parquet Project, n.d.), Delta Lake (Armbrust et al., 2020), Iceberg, and Hudi are open specifications, enabling data portability across engines and cloud providers. Without the concern of vendor lock-in due to proprietary warehouse formats, organizations can use the right engine for the right workload and can move between engines and cloud providers without having to rewrite data. A survey by Cloudera in 2024 found that 90 percent of IT leaders believe that unifying the data lifecycle on a single platform is critical to analytics and AI, and that open-format portability is a prerequisite of that unification.

Third, cost efficiency is achieved by tiered storage and elastic compute runtimes. Cloud object stores provide multiple storage tiers that come with progressively lower costs in exchange for longer access latencies. Lakehouses can transparently use these tiers, with frequently accessed bronze and silver zones in standard storage and rarely accessed historical partitions migrated to lower-cost tiers. The combination of these techniques can reduce the total cost of ownership of an analytics platform by an order of magnitude relative to a classical warehouse. The cost picture is not purely advantageous, however: unoptimized queries against large tables can produce surprising bills, idle tables can accumulate small files that degrade performance, and versioning produces storage growth that must be managed through vacuum operations. The mature enterprise lakehouse practice therefore includes explicit cost-engineering capabilities such as query-cost monitoring, table-maintenance schedules, and lifecycle policies (Pierce, 2025).

9. Future of Intelligent Data Platforms and Industry-Specific Applications

The lakehouse has matured rapidly enough that the conversation in the enterprise architecture community has shifted from whether to adopt it to where it will go next. Three trajectories are visible in the current literature. The first is the convergence of table formats: Delta Lake, Iceberg, and Hudi have converged substantially in their feature sets, and the industry is investing in interoperability layers that allow organizations to operate across formats without committing to one. The second trajectory is the deeper integration of streaming and batch into a single conceptual model, building on the unified semantics that Flink and Spark Structured Streaming pioneered (Armbrust et al., 2018; Carbone et al., 2015). The third is the continued elaboration of AI-native capabilities, including vector indexes co-located with structured tables and integrated feature stores, that make the lakehouse the natural foundation for the generative AI workloads that increasingly define competitive advantage.

In financial services, lakehouses underpin real-time fraud detection systems that score transactions in milliseconds against features computed from terabytes of historical behavior, regulatory reporting systems that produce auditable views of trades and positions, and AI-driven advisor copilots. In retail and e-commerce, they underpin personalization engines that combine clickstream events, transaction history, and inventory state and supply-chain coordination systems that propagate demand signals from point-of-sale to logistics within seconds. In healthcare, lakehouses serve clinical data warehouses that integrate electronic health record data, claims data, imaging metadata, and genomic data into a single queryable substrate for population health analytics and clinical decision support. The Health Insurance Portability and Accountability Act and FDA Software as a Medical Device guidance impose specific data governance obligations that the lakehouse's lineage, time travel, and audit capabilities are well-suited to meet. In insurance, lakehouses serve actuarial modeling, claims processing, underwriting decision support, and AI-driven first-notice-of-loss triage. In manufacturing and Industry 4.0, lakehouses ingest sensor telemetry from connected equipment and serve predictive-maintenance models, quality assurance pipelines, and production-optimization workflows. In the public sector, lakehouses support benefit administration, public health surveillance, and education analytics, ensuring that auditability and openness meet public-accountability requirements.

Table 1. Industry-specific lakehouse applications and required capabilities

Vertical	Representative Use Cases	Critical Lakehouse Capabilities
Financial services	Real-time fraud detection, regulatory reporting, risk analytics, AI advisor copilots	Sub-second streaming ingest, ACID reliability, auditable lineage, time travel, feature & vector stores
Retail/e-commerce	Real-time personalization, inventory & supply-chain, demand forecasting, merchandising copilots	High-throughput event ingest, streaming features, gold zones for personalization signals
Healthcare	Clinical data warehousing, population health, decision support, claims & regulatory reporting	PHI-aware classification, lineage for FDA SaMD & HIPAA, long-horizon time travel
Insurance	Actuarial modeling, claims triage, underwriting support, first-notice-of-loss AI	Reproducible modeling, fairness-monitoring features, vectorized claims documents
Manufacturing/IoT	Predictive maintenance, quality assurance, production optimization, digital-twin telemetry	Sensor-stream ingest, low-latency aggregation, large-scale time-series tables
Public sector	Benefit administration, public-health surveillance, education analytics, open data platforms	Open-format portability, auditable lineage, governed data accessible to multiple agencies

Conclusion

The lakehouse has emerged as the dominant architectural pattern for enterprise data platforms because it resolves the long-standing trade-offs that defined the previous two generations. Where data warehouses delivered reliability and SQL performance at the cost of scalability and AI-readiness, and data lakes delivered scale and openness at the cost of consistency and quality, the lakehouse delivers both: ACID transactions on cheap, open, cloud-object-store storage, accessed uniformly by a wide ecosystem of analytics, machine learning, and AI engines. The technical foundations are well-established in the past literature. The two mathematical interludes in this article translate the architectural arguments into quantitative ones. Interlude 1 shows that incremental delta processing achieves speedups of 100x to 10,000x over full rebuilds for typical change-rate scenarios, which is what makes streaming-latency lakehouse operations economically feasible. Interlude 2 shows that the combination of column projection, partition pruning, and clustering can reduce query scan volume by four orders of magnitude relative to a naive full-table scan, which is what makes interactive SQL feasible against multi-terabyte tables. Together with the cost-engineering practices of Section 8, these quantitative properties explain why the lakehouse has been adopted not only by data-engineering teams seeking architectural elegance but also by chief financial officers seeking to control the rapidly growing cost of enterprise data infrastructure. The industry-specific applications described in Section 9, across financial services, retail, healthcare, insurance, manufacturing, and the public sector, demonstrate that the value of this consolidation extends well beyond engineering economics.

Table 2. Comparative characteristics of the three major open lakehouse table formats

Dimension	Delta Lake	Apache Iceberg	Apache Hudi
Origin	Databricks, 2016	Netflix, 2017 (ASF 2020)	Uber, 2016
ACID transactions	Transaction log over object store	Metadata tree with snapshots	Timeline-based record-level updates
Schema evolution	Full support	Full support with hidden partitioning	Full support
Time travel	Version-based	Snapshot-based	Timeline-based
Incremental processing	MERGE INTO and Change Data Feed	Row-level deletes and positional deletes	Primary-key-based upserts (MOR & COW)
Streaming integration	Spark Structured Streaming	Flink, Spark	Flink, Spark, Hudi Streamer
Partition evolution	Partition overwrite patterns	Hidden partitioning, no query rewrite needed	Partition pruning via index
Primary deployment	Databricks Lakehouse Platform, Microsoft Fabric	Netflix, AWS, Snowflake	Uber, Amazon EMR, Azure HDInsight
Key citation	Armbrust et al. (2020)	Eswararaj et al. (2025)	Kasireddy (2023)

References

1. Apache Kafka Project. (n.d.). Apache Kafka: Open-source distributed event streaming platform. Apache Software Foundation. Available: https://kafka.apache.org/
2. Apache Parquet Project. (n.d.). Apache Parquet: Open-source columnar storage format. Apache Software Foundation. Available: https://parquet.apache.org/
3. Armbrust, M., Das, T., Sun, L., Yavuz, B., Zhu, S., Murthy, M., Torres, J., et al. (2020). Delta lake: High-performance ACID table storage over cloud object stores. Proceedings of the VLDB Endowment, 13(12), 3411-3424. Available: https://www.eecs.umich.edu/courses/cse584/static_files/papers/p975-armbrust.pdf
4. Armbrust, M., Das, T., Torres, J., Yavuz, B., Zhu, S., Xin, R., Ghodsi, A., Stoica, I., & Zaharia, M. (2018). Structured streaming: A declarative API for real-time applications in Apache Spark. In Proceedings of the 2018 International Conference on Management of Data (pp. 601-613). Available: https://dl.acm.org/doi/abs/10.1145/3183713.3190664
5. Armbrust, M., Ghodsi, A., Xin, R., & Zaharia, M. (2021). Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics. In Proceedings of CIDR 2021. Available: https://15721.courses.cs.cmu.edu/spring2023/papers/02-modern/armbrust-cidr21.pdf
6. Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., & Tzoumas, K. (2015). Apache flink: Stream and batch processing in a single engine. The Bulletin of the Technical Committee on Data Engineering, 38(4). Available: https://research.tudelft.nl/en/publications/apache-flink-stream-and-batch-processing-in-a-single-engine/
7. Chaudhari, A. V., & Charate, P. A. (2025). Optimizing data lakehouse architectures for scalable real-time analytics. International Journal of Scientific Research in Science, Engineering and Technology, 12(2), 809-822. Available: https://doi.org/10.32628/IJSRSET25122198
8. Cloudera. (2024). Global survey: 90% of IT leaders believe unifying the data lifecycle is critical for analytics and AI. Available: https://www.cloudera.com/about/news-and-blogs/press-releases/2024-03-28-global-survey-reveals-90-of-it-leaders-believe-that-unifying-the-data-lifecycle-on-a-single-platform-is-critical-for-analytics-and-ai.html
9. Eken, B., Pallewatta, S., Tran, N., Tosun, A., & Babar, M. A. (2025). A multivocal review of MLOps practices, challenges and open issues. ACM Computing Surveys, 58(2), 1-35. Available: https://dl.acm.org/doi/full/10.1145/3747346
10. Eswararaj, D., Nellipudi, A. B., & Kollati, V. (2025). A comparative study of delta parquet, iceberg, and hudi for automotive data engineering use cases. arXiv preprint arXiv:2508.13396. Available: https://arxiv.org/abs/2508.13396
11. Hewage, N., & Meedeniya, D. (2022). Machine learning operations: A survey on MLOps tool support. arXiv preprint arXiv:2202.10169. Available: https://arxiv.org/abs/2202.10169

12. Jha, V. Y. (2024). Data lakehouse architectures: Bridging structured and unstructured data. International Journal of Advanced Research in Computer Science & Technology (IJARCST), 7(6), 11218-11221. Available: https://www.ijarcst.org/index.php/ijarcst/article/view/67
13. Kasireddy, J. R. (2023). Operationalizing lakehouse table formats: A comparative study of Iceberg, Delta, and Hudi workloads. International Journal of Research Publications in Engineering, Technology and Management (IJRPETM), 6(2), 8371-8381. Available: https://www.ijrpetm.com/index.php/IJRPETM/article/view/273
14. Mishra, S. (2022). Comparing Apache Iceberg and Databricks in building data lakes and mesh architectures. International Journal of AI, BigData, Computational and Management Studies, 3(4), 37-48. Available: https://ijaibdcms.org/index.php/ijaibdcms/article/view/211
15. Pierce, J. K. (2025). Modern data lakehouse architectures: Integrating cloud warehousing, analytics, and scalable data management. International Journal of Advanced Artificial Intelligence Research, 2(12), 12-17. Available: https://aimjournals.com/index.php/ijaair/article/download/466/406/985
16. Serepas, F., Papias, I., Bellos, N., & Marinakis, V. (2024). A comprehensive approach to real-time and batch processing for energy-efficient IoT homes: Leveraging lambda architecture and data lakes. In 2024 15th International Conference on Information, Intelligence, Systems & Applications (IISA) (pp. 1-6). IEEE. Available: https://ieeexplore.ieee.org/abstract/document/10786715
17. Vohra, D. (2016). Apache parquet. In Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools (pp. 325-335). Apress. Available: https://link.springer.com/chapter/10.1007/978-1-4842-2199-0_8
18. Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., et al. (2016). Apache spark: A unified engine for big data processing. Communications of the ACM, 59(11), 56-65. Available: https://dl.acm.org/doi/fullHtml/10.1145/2934664
19. Zhengxin, F., Yi, Y., Jingyu, Z., Yue, L., Yuechen, M., Qinghua, L., Xiwei, X., et al. (2023). MLOps spanning whole machine learning life cycle: A survey. arXiv preprint arXiv:2304.07296. Available: https://arxiv.org/abs/2304.07296